Universidad de Buenos Aires
Facultad de Ingeniería
Tesis de Maestría en Ciencias de la Ingeniería

Desarrollo de circuitos integrados CMOS para aplicaciones de RFID

Andrew Parlane

Director: Dr. Ing. Mariano Garcia-Inza
Codirectores: Ing. Federico G. Zacchigna, Ing. Octavio Alpago

Resumen

Este trabajo presenta el diseño de circuitos integrados digitales para la implementación de un TAG RFID en un chip CMOS. Los requerimientos de los circuitos son definidos por un proyecto de investigación marco cuyo objetivo general es el desarrollo de sensores inalámbricos de radiación ionizante para dosimetría médica. Esto requiere la integración de diferentes subsistemas en un ASIC a fabricar en tecnología CMOS.

Los circuitos presentados en esta tesis consisten en un bloque digital de control, que puede recibir y responder a las tramas definidas en la norma ISO/IEC 14443A, y mediante un protocolo propietario encima de la norma, controlar otros subcircuitos del chip necesarios para la adquisición de la señal dosimétrica y su posterior transmisión inalámbrica. La implementación de la norma es presentada en un núcleo IP genérico. Por lo tanto es apto para uso en otros proyectos y con cualquier lector comercial que trabaje bajo norma. El protocolo propietario consiste en cinco mensajes que permiten el muestreo de hasta quince TAGs de forma sincronizada. El muestreo simultáneo de múltiples TAGs sensores utilizando RFID es una estrategia novedosa desarrollada en esta tesis, el cual representa un potencial avance en el desarrollo de métodos de control dosimétrico en aplicaciones médicas.

El trabajo realizado incluye el diseño de los bloques digitales utilizando HDL SystemVerilog y su verificación funcional. Luego, se utiliza el kit de diseño del proceso (PDK, por sus siglas en Inglés) XH018 (nodo tecnológico de 180 nm) de la foundry XFAB para realizar la síntesis, el place and route y la generación de las máscaras de fabricación (layout).

Las pruebas de verificación llevadas a cabo dan alta confianza en el diseño. Las mismas incluyen: más de cien horas de simulaciones del RTL ejecutando 187 aserciones de SystemVerilog más de cien mil millones de veces y generando informes de cobertura de código, la verificación formal de equivalencia entre lógica RTL y los netlists post síntesis y post implementación, Design Rules Check (DRC), y Layout Vs Schematic (LVS); todas esas pruebas tienen resultados favorables.

El layout final no tiene violaciones de timing, el slack de Setup menor es 1,59 ns y de Hold es 0,02 ns, el área utilizada es 0,087 mm^2 (295,68 µm por 294,00 µm), y la estimación de consumo de potencia promedio es 256 µW.

Aclaraciones

Aclaraciones Por cuestiones de claridad en esta tesis cuándo se dice que una señal está en ‘1’ o ‘0’ significa que la señal tiene un valor lógico igual a ’1’ o ‘0’ respectivamente.

El código RTL, los bancos de pruebas y los scripts de síntesis y place & route están publicados de forma abierta en GitHub bajo la licencia GNU v3.0, de manera que cualquier parte de este trabajo pueda ser utilizada en otros proyectos académicos o comerciales. Para facilitar la adopción y modificación de este trabajo por la comunidad internacional, el código y los scripts están escritos con nombres y comentarios en Inglés. El enlace al repositorio y detalles de su estructura son dados en el Apéndice A.

Para facilitar la lectura de esta tesis las señales, variables y parámetros están escritos en *cursiva*, los módulos, clases y funciones están escritos en **negritas**. Además se utilizan paréntesis para nombrar a las funciones, aunque sin sus argumentos, por ejemplo: **compare()**. Finalmente los nombres de las tramas definidas en los protocolos están escritos en MAYÚSCULAS.

Índice

Introducción

Motivación y Contexto de Trabajo

La radioterapia es un tratamiento médico que consiste en utilizar radiación ionizante para eliminar células cancerígenas que forman tumores. Sin embargo, la radiación puede presentar riesgos a tejido sano, especialmente si la dosis aplicada es mayor a la necesaria o mal localizada. Hay varios incidentes registrados donde algunas personas recibieron una sobredosis durante radioterapia, y en algunos casos dosis letales [9]. Por otro lado, una dosis localizada pero demasiado pequeña reduciría la efectividad del tratamiento. Por esas razones es muy importante adoptar consideraciones de QA (Quality Assurance), la cual puede definirse como:

> Los procedimientos que aseguren el cumplimiento de las prescripciones médicas con respeto a la dosis entregada al volumen deseado, junto con una dosis mínima a tejido sano, exposición mínima al personal, y el monitoreo adecuado del paciente para determinar el resultado del tratamiento. [19, traducción mía]

Una técnica importante en QA es dosimetría in-vivo (IVD), que es la práctica de medir la dosis recibida durante el tratamiento en tiempo real. La IVD es recomendada para su uso en radioterapia [17][6], con un error menor del 3 % por tratamiento. Existen varios tipos de sensores de radiación que pueden ser utilizados en IVD con radioterapia. Entre ellos se encuentran los sensores MOSFET, los cuales tienen varias ventajas, como por ejemplo que pueden ser leídos en tiempo real o posteriormente, son pequeños y robustos, pero también tienen algunas limitaciones [9]. Los investigadores del Laboratorio de Física de Dispositivos - Microelectrónica de la Facultad de Ingeniería de la Universidad de Buenos Aires han trabajando durante los últimos años en mejorar el desempeño de sensores MOSFET de radiación para uso en IVD [4][3]. Otra ventaja de los sensores MOSFETs es que pueden ser integrados en un mismo chip con circuitos adicionales que permitan su lectura, digitalización y posterior transmisión de los resultados en tiempo real. Esto representa una importante ventaja frente al método usual de lectura post irradiación, ya que el seguimiento en tiempo real permitiría ajustar la dosis durante la ejecución del tratamiento.

Un sensor pequeño construido con una cantidad mínima de componentes permitiría

obtener arreglos de sensores con excelente resolución espacial y así realizar un mapeo dosimétrico de una zona de interés. A la hora de implementar esta solución, los TAGs RFID (Radio Frequency Identification) pasivos son una excelente opción, ya que sólo requieren del circuito integrado y una antena externa. La alimentación y la comunicación pueden realizarse a través del campo electromagnético generado por el dispositivo de lectura. Un sistema de estas características, además de ser más simple, sería más cómodo para el paciente y más práctico para su uso en el campo médico.

En su tesis de 2018 [15], Arana analizó la relación entre la frecuencia de operación de un TAG RFID y su distancia máxima de la lectura considerando los límites de exposición de humanos a campos electromagnéticos definidos en IEEE C95.1. Los resultados muestran que una frecuencia en el orden de 10 MHz daría el mejor rango de operación mientras manteniendo el campo electromagnético (EM) dentro de límites seguros. Esta frecuencia se encuentra cercana a los 13.56 MHz de RFID HF (High Frequency). Arana muestra en su tesis el diseño de la antena y de circuitos integrados analógicos para un TAG ISO/IEC 14443 tipo A que puede funcionar a una distancia de 30 cm de la lectura. También en una publicación de 2014 [1], Alcalde et al. presentan el diseño y fabricación de un TAG RFID que implementa parte de la norma ISO/IEC 14443 tipo A. El funcionamiento es verificado a través de mediciones experimentales del sistema funcionando en loopback.

Un atributo del protocolo ISO/IEC 14443A es que permite hasta 15 TAGs activos al mismo tiempo. Esto da la posibilidad de obtener muestras de los sensores de forma sincronizada, con la ventaja de medir en múltiples lugares del cuerpo del paciente. Esto contribuye a mejorar la verificación de la ejecución del tratamiento planificado y por lo tanto se alinea con los criterios de QA.

En un artículo publicado en 2016 [26], Villani et al. se desarrolló un sensor de radiación inalámbrico para uso en IVD por radioterapia. El sensor emite una señal RF con frecuencia que depende de la dosis de radiación recibida. La ventaja de este enfoque es la simplicidad del diseño, el circuito integrado no necesita un bloque digital complejo para soportar el protocolo. Las desventajas son: solo es posible usar solo un sensor a la vez, requiere una fuente de alimentación externa (batería), no permite configurar el sensor inalámbricamente, y por no haber sido diseñado dentro de una norma, no es compatible con otros equipos requiriendo de un lector diseñado ad hoc para la aplicación.

Los circuitos diseñados en esta tesis permitirán implementar una red de sensores inalámbricos cuya finalidad es medir dosis de radiación en un tratamiento de radioterapia en diferentes lugares de interés. La adquisición en múltiples puntos tiene como objetivo mejorar

el control de la irradiación para ajustar con mayor precisión la dosis entregada por el acelerador a la planificación previa. Típicamente, la dosis entregada por los LINAC no es lineal con el tiempo por lo cual, para que la medición en tiempo real sea útil, se requiere sincronizar el muestreo de los sensores.

ISO/IEC 14443A

La norma ISO/IEC 14443 fue desarrollada por la Organización Internacional de Normalización (ISO) y la Comisión Electrotécnica Internacional (IEC), específicamente por el grupo de tareas 2 del grupo de trabajo 8 de la subcomisión del comité técnico mixto 1. Fue publicado primero en 2001 con dos interfaces distintas: tipos A y B. Tipo A fue desarrollado en colaboración con Mikron (adquirida desde entonces por Phillips), basado en su tecnología Mifare. Esta interfaz fue diseñada como una tarjeta para almacenar datos únicamente. Tipo B fue desarrollado en colaboración con varios operadores de sistemas de transporte públicos de Europa e Innovatron. Principalmente diseñado para pagos de tarifas pero también puede funcionar como billetera electrónica y verificación de identidad. Esta interfaz fue diseñada como una tarjeta que al tener un microprocesador, además de almacenar datos, cuenta con capacidad de procesamiento. Al principio estas dos interfaces fueron complementarias, pero luego de los años sus aplicaciones se ampliaron y diversificaron [20].

La norma viene en cuatro partes:

- ISO/IEC 14443-1: Características Físicas.
- ISO/IEC 14443-2: Radiofrecuencia Potencia y Señal Interfaz.
- ISO/IEC 14443-3: Inicialización y Anticolisión.
- ISO/IEC 14443-4: Protocolo de Transmisión.

La parte una define atributos físicas de la PICC (Proximity Card, la tarjeta o TAG) como las dimensiones y el rango de operación a temperatura ambiente. La parte dos define la interfaz inalámbrica de la transmisión de potencia y comunicaciones bidireccionales entre un PCD (Proximity Coupling Device, la lectura) y la PICC. La parte tres define el formato de los bytes, la estructura de las tramas y los mensajes necesarios para el descubrimiento de todas las PICCs presentes en el campo del PCD y para activarlas. Finalmente, la parte cuatro define un protocolo de bloques para la configuración de la PICC y la transmisión de mensajes de nivel aplicación.

Un sistema tiene un PCD que es el maestro y una o más PICCs que son los esclavas.

Una PICC solo responde a solicitudes, no inicia comunicaciones. Durante el proceso de inicialización múltiples PICCs pueden responder a la misma solicitud, así colisiones son posibles y esperadas. Después del proceso de inicialización, en un sistema correctamente configurado, solo una PICC responde a cada solicitud.

ISO/IEC 14443-1: Características Físicas

Esta parte de la norma define las condiciones de operación en cuáles la PICC debe poder operar correctamente. Estos requisitos son: la exposición a rayos X, el límite dinámico de flexión y de torsión, la exposición a campos magnéticos y eléctricos alternos, la exposición a electricidad estática, y la temperatura ambiente[10].

ISO/IEC 14443-2: Radiofrecuencia Potencia y Señal Interfaz

El PCD emite un campo electromagnético con frecuencia $f_c = 13{,}56\,\text{MHz} \pm 7\,\text{kHz}$. Las PICCs dentro de este campo se acoplan inductivamente al campo para la transferencia de potencia. El campo es modulado para la comunicación entre los dispositivos. El PCD usa modulación de amplitud para enviar información a las PICCs, y las PICC envían sus respuestas con modulación de carga mediante un subcarrier. La norma define varias tasas de bits, pero las comunicaciones siempre comienzan con tasa de bits $f_c/128 \approx 106\,\text{Kbps}$ en las dos direcciones. La codificación de los bits y los parámetros de modulación dependen de: la dirección de la comunicación, si las PICCs son tipo A o B, y la tasa de bits. La Figura 1.1 muestra una representación de las dos direcciones de comunicación para PICCs de tipo A y de tipo B.

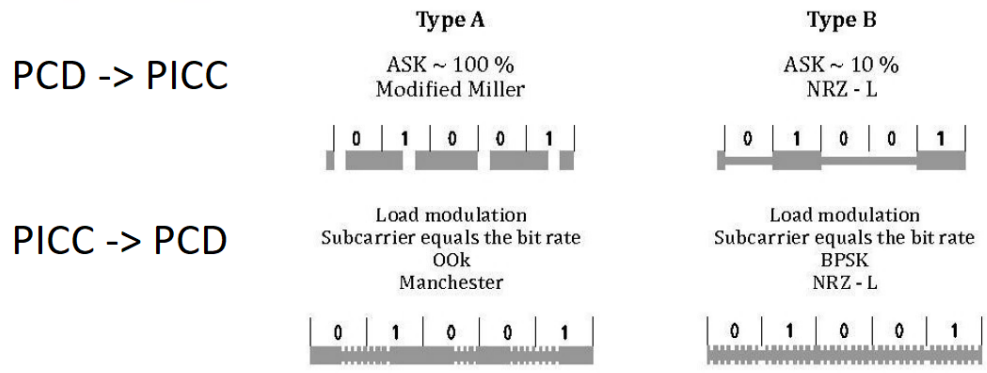

Figura 1.1: Comunicaciones con una tasa de bits de $f_c/128$ [11, Modificaciones Mías]

Para PICCs de tipo A la modulación por comunicaciones desde el PCD hasta la PICC es ASK (Amplitude Shift Keying). Cuándo la amplitud de la portadora baja al 5 % del campo original, la norma define esto como una pausa. Un esquema de la misma se muestra en la Figura 1.2, con valores que se definen en el Cuadro 1.1. Debido a que una PICC pasiva recibe su potencia inalámbricamente desde el campo electromagnético, es responsabilidad de la PICC mantener los rails de alimentación internos a tensiones adecuadas durante las pausas para que el sistema no cambie de estado. La codificación de los bits es Modified Miller. Cada trama comienza con un SOC (Start of Communication), tiene un número de bits de datos y termina con un EOC (End Of Communications). SOC, EOC y los valores lógicos son enviados mediante secuencias. Cada secuencia tiene el largo de una duración de bit, lo que es 128 ciclos de la portadora. La presencia y la ubicación de una pausa dentro de una duración de bit define el tipo de la secuencia, como es mostrado en la Figura 1.3. Una pausa que ocurre al principio de la duración de bit es una secuencia Z, una pausa que ocurre en el medio de la duración de bit es una secuencia X, y una duración de bit sin pausas es una secuencia Y. El SOC es la secuencia Z. Un '1' lógico es una X. Un '0' lógico depende en la última secuencia, si fue una X, se envía una Y, si fue una Y o una Z, se envía una Z. La trama termina con el EOC lo que es un '0' lógico, seguido por una secuencia Y. La Figura 1.4 muestra dos ejemplos de tramas.

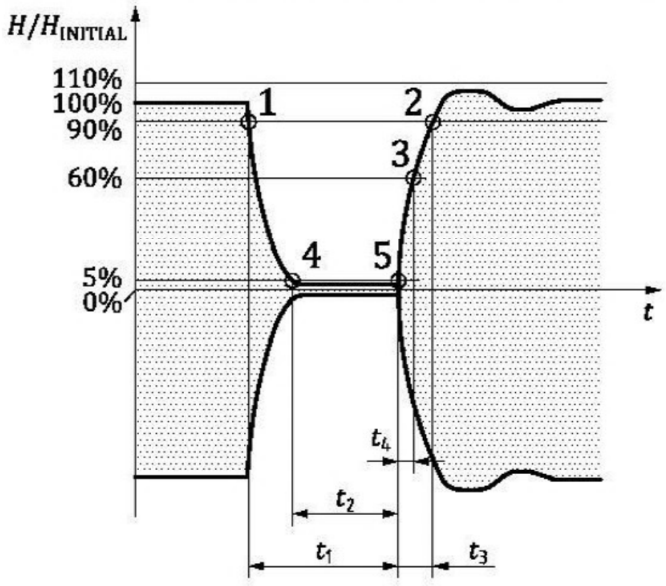

Figura 1.2: El formato de una pausa tipo A [11]

Parámetro	Condición	Mínimo	Máximo
t_1		$^{28}/_{f_c}$	$^{40,5}/_{f_c}$
t_2	$t_1 > {}^{34}/_{f_c}$	$^{7}/_{f_c}$	t_1
	$t_1 \leq {}^{34}/_{f_c}$	$^{10}/_{f_c}$	
t_3		$1,5 \cdot t_4$	$^{16}/_{f_c}$
t_4		0	$^{6}/_{f_c}$

Cuadro 1.1: Parámetros de timing para una pausa [11]

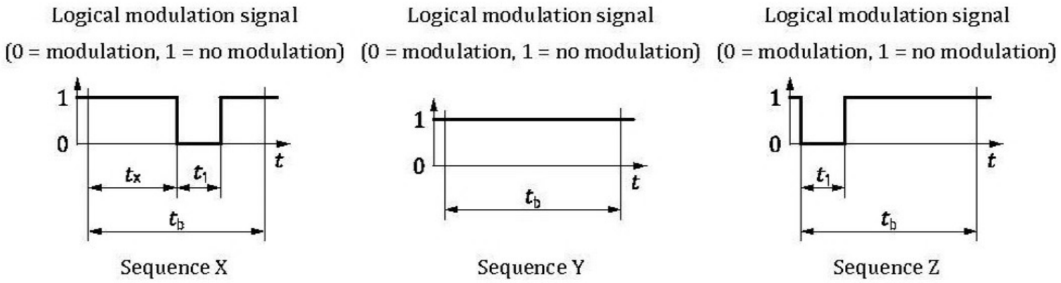

Figura 1.3: Codificación Modified Miller [11]

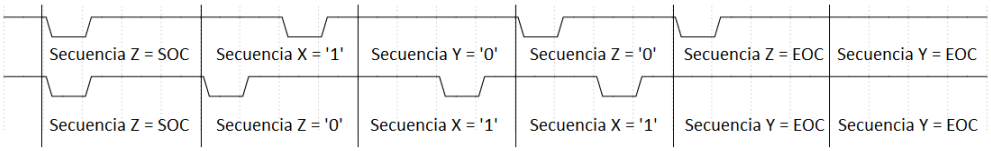

Figura 1.4: Ejemplos de tramas PCD → PICC tipo A

Para responder las PICCs tipo A usan modulación de carga para generar una subportadora. La frecuencia de la subportadora (f_s) depende de la tasa de bit. Para una tasa de bit de $f_c/128$ la frecuencia de la subportadora es $f_s = f_c/16 \approx 848\,\text{kHz}$. Como en el caso de comunicaciones del PCD a la PICC la duración de un bit es $128/f_c = 8/f_s$. La portadora debe ser solo modulada con la subportadora cuando la PICC está enviando datos. Los bits a transmitir tienen codificación Manchester, para representar un '1' lógico la señal es en '1' por la primera mitad de la duración de bit, y en '0' por la segunda mitad, para representar un '0' lógico es al revés. La señal que maneja el modulador de carga es la operación lógica AND entre la señal de codificación Manchester y la subportadora como se muestra en la Figura 1.5. Una trama comienza con un SOC lo que es un '1' lógico, y termina con un EOC lo que es una duración de bit sin modulación, eso es a decir que la señal al modulador de carga es en '0' por toda la duración de bit. La Figura 1.6 muestra un ejemplo de una trama que ingresa a la compuerta AND con la subportadora.

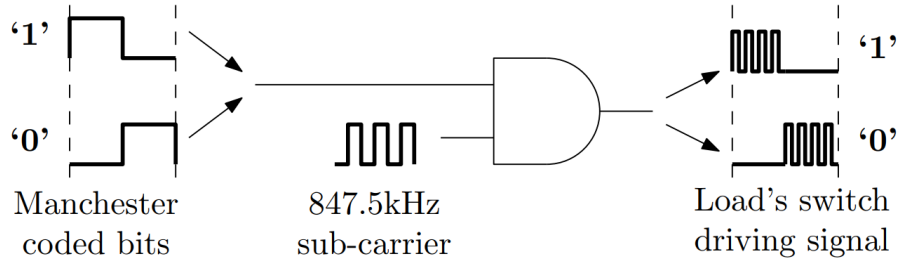

Figura 1.5: Codificación de bits y modulación de la portadora con la subportadora para comunicaciones de la PICC a PCD [1]

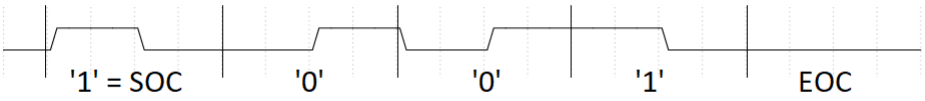

Figura 1.6: Ejemplo de una trama por comunicaciones PICC tipo A → PCD.

ISO/IEC 14443-3: Inicialización y Anticolisión

La información presentada en esta sección es válida considerando una tasa de bit de $f_c/128$ en cada dirección y PICCs tipo A.

La parte 3 de la norma define un FDT (Frame Delay Time), lo que es el tiempo entre dos tramas transmitidas en direcciones opuestas. Los tiempos del FDT definidos en la

norma son especificados en números de ciclos de la portadora, y son medidos entre el último flanco de modulación de la primera trama y el primer flanco de modulación de la segunda trama. El Cuadro 1.2 muestra los FDTs especificados en la norma. En el caso del FDT entre una trama del PCD y una de la PICC, el FDT usado depende en el último bit lógico de la trama, y los valores especificados son absolutos para tramas de inicialización y tiempos mínimos para otras tramas.

Primera Trama	**Segunda Trama**	**Último Bit**	**FDT**
PCD → PICC	PICC → PCD	'0'	1172
		'1'	1236
PICC → PCD	PCD → PICC		1272

Cuadro 1.2: Valores del FDT en número de ciclos de la portadora.

Una trama comienza con un SOC, después se envían los datos empezando por el bit menos significativo, y termina con un EOC. Cada 8 bits de data hay un bit de paridad impar, así que el número de los 1s en cada byte más su bit de paridad es impar. Hay tres tipos de tramas definidas:

Tramas Cortas
: Tienen siete bits de datos sin bit de paridad.

Tramas Estándares
: Tienen un número de bytes enteros, cada uno con un bit de paridad.

Tramas Anticolisión orientada a bits
: Tienen siete bytes cada uno con un bit de paridad, y están divididas en dos: la primera parte es enviada desde el PCD y la segunda parte desde la PICC. La partición puede ser después de un byte entero, incluyendo su bit de paridad (Figura 1.7 arriba), o en medio de un byte (Figura 1.7 abajo). Estas tramas son usadas durante el proceso de inicialización para que el PCD pueda detectar todas las PICCs en su campo electromagnético.

Unas de las tramas definidas en la norma terminan con un CRC16 (Cyclic Redundancy Check de 16 bits). El CRC16 permite el receptor de la trama determinar si el contenido fue corrompido. El polinomio del CRC es $P(x) = x^{16} + x^{12} + x^5 + 1$, y el valor inicial es 0x6363 [12][14].

Cada PICC tiene un UID (Unique Identifier) que el PCD puede utilizar para enumerar todas las PICCs en su campo electromagnético y elegir cuáles activar. A pesar de que el nombre indique que el ID es único, no tiene que ser así. Está permitido usar un ID

aleatorio (RID), o un ID no único (NUID). Los UIDs pueden ser: simples (4 bytes), dobles (7 bytes), o triples (10 bytes). Es requerido que todas las PICCs presentes en el campo de un PCD tienen UIDs únicos. Cuando se usan PICCs con NUIDs o RIDs es posible que haya conflictos. Debido a la gran cantidad de NUIDs posibles, la probabilidad de conflictos es mínima en un sistema bien diseñado.

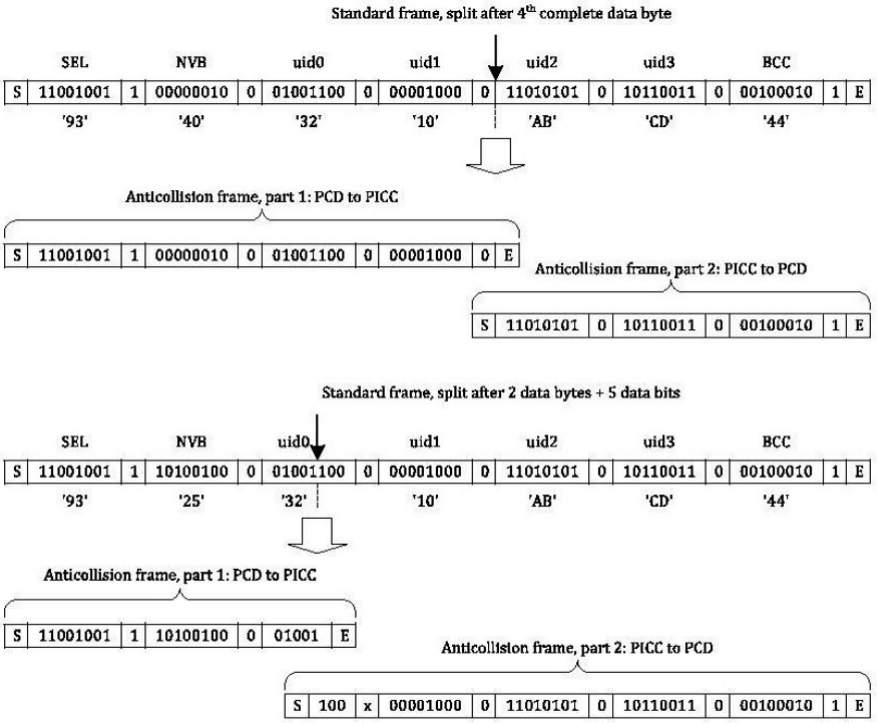

Figura 1.7: Trama Anticolisión orientada a bits [12, Modificaciones Mías]

Para identificar las PICCs presentes y activarlas, la norma define cinco comandos y sus respuestas, mostrados en el Cuadro 1.3. La Figura 1.8 especifica cómo una PICC debería responder cuando recibe un comando dependiendo de su estado actual.

PCD → PICC			PICC → PCD		
Solicitud	**Tipo**	**CRC**	**Respuesta**	**Tipo**	**CRC**
REQA	Corta	No	ATQA	Estándar	No
WUPA	Corta	No	ATQA	Estándar	No
ANTICOLLISION	Anticolisión	No	ANTICOLLISION	Anticolisión	No
SELECT	Estándar	Sí	SAK	Estándar	Sí
HLTA	Estándar	Sí	Sin Respuesta		

Cuadro 1.3: Solicitudes y respuestas definidas en ISO/IEC 14443-3A

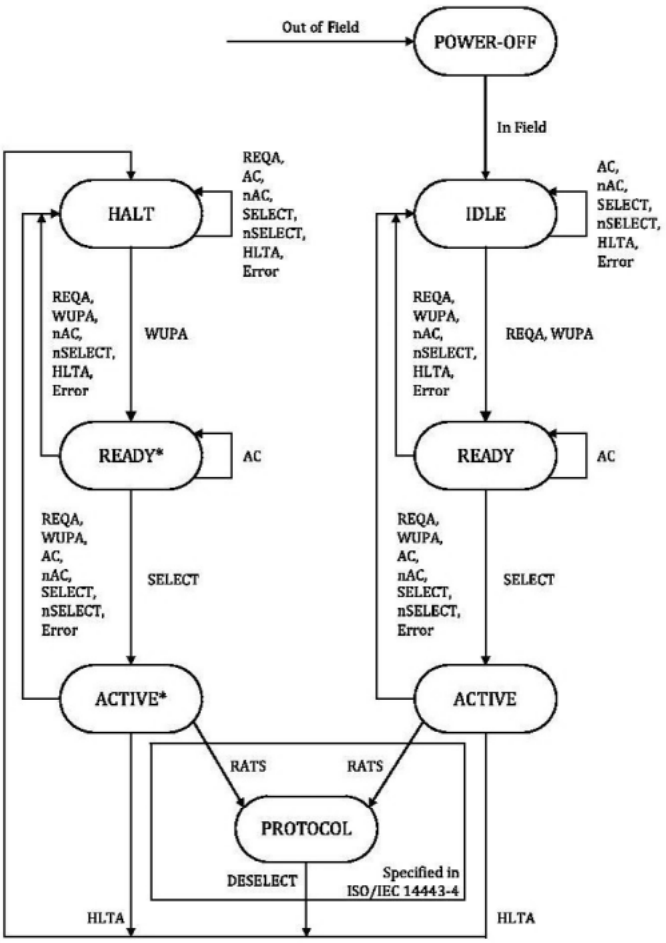

Figura 1.8: Diagrama de transiciones de estados [12]

Cuando una PICC entra en un campo electromagnético, comienza en el estado IDLE. Un comando REQA o WUPA cambia el estado a READY. En ese estado el PCD puede enviar comandos de ANTICOLLISION para determinar el UID de una de las PICCs presentes. Con ese UID el PCD puede enviar un SELECT para mover la PICC al estado ACTIVE. Desde aquí el PCD puede enviar un RATS para terminar activando la PICC. En el caso de recibir un error, o un comando no esperado, la PICC vuelve al estado IDLE. Hay tres otros estados también: HALT, READY* y ACTIVE*. Las diferencias únicas entre estos estados y IDLE, READY y ACTIVE, son: WUPA es el único comando aceptado en HALT y el comando REQA es ignorado, y un error o un comando no esperado recibido en READY* o ACTIVE* causa la PICC volver a HALT en vez de IDLE. La ventaja de esto es que si el PCD decide que no quiere activar una PICC particular, puede ponerla en el estado HALT, y enviando un nuevo REQA comenzará el proceso de inicialización de nuevo en todos las PICCs menos aquellos que están en HALT.

En el caso de una PICC con UID doble el proceso de ANTICOLLISION y SELECT tiene que repetir dos veces antes de que la PICC se mueva al estado ACTIVE. En el primer lazo, los mensajes usan los primeros tres bytes de su UID junto con una etiqueta cascada: CT (Cascade Tag). La PICC responde al SELECT especificando que su UID no está completo todavía, y el lazo comienza de nuevo, esta vez usando los últimos cuatro bytes del UID. Por una PICC con UID triple este proceso tiene tres lazos. La Figura 1.9 muestra este proceso.

El comando de ANTICOLLISION funciona de la siguiente forma: el PCD envía un UID parcial, y todas las PICCs en el campo electromagnético cuyas UIDs corresponden con la parte enviada, responden con los demás bits de sus UIDs. Debido al FDT fijo todas las PICCs comienzan responder de forma sincronizada. Las respuestas tienen codificación Manchester, por lo tanto cuándo dos PICCs envían valores lógicos diferentes el PCD puede detectar la colisión porque hay modulación durante todo el tiempo de bit. De esta manera el PCD sabe qué parte del UID es compartido entre todas las PICCs, y usando una búsqueda binaria puede determinar el UID completo de una de las PICC presentes. Esa PICC entonces puede ser activada o puesta en el estado HALT, y después el PCD puede repetir el proceso para enumerar todas las demás PICCs.

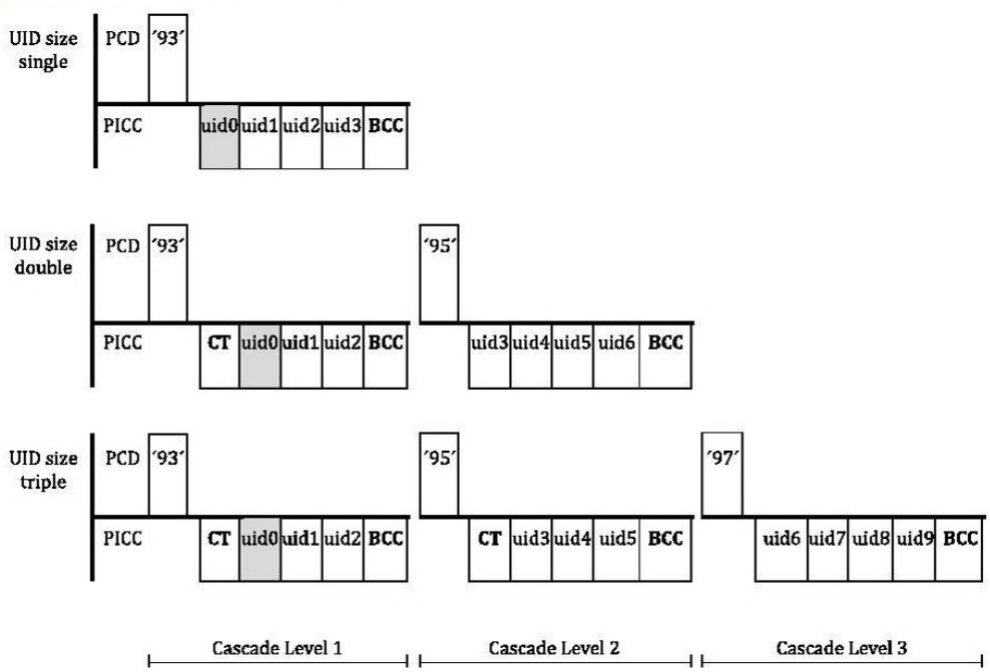

Figura 1.9: Niveles cascadas para UIDs simples, dobles y triples [12]

ISO/IEC 14443-4: Protocolo de Transmisión

La mayor parte de esta subsección es común para PICCs tipo A y B, sin embargo hay dos comandos extras para terminar activando el protocolo para PICCs de tipo A. Estos comandos son RATS y PPS. El PCD debería enviar un RATS como el primer comando después de que una PICC entra en el estado ACTIVE o ACTIVE*. En el RATS el PCD asigna un CID (Card Identifier) a la PICC, esto es un identificador que el PCD puede usar para direccionar un mensaje a una PICC en particular, así que el CID debe ser único para cada PICC activa. En diferencia al UID, el CID solo tiene cuatro bits y son asignados al momento de la inicialización y no en la de fabricación. Este CID es válido sólo hasta que la PICC es desactivada. La respuesta al RATS es la ATS lo que contiene información sobre la capacidad de la PICC, por ejemplo las tasas de bits y el tamaño máximo de una trama que la PICC puede recibir. El comando PPS es opcional, y solo puede ser enviado inmediatamente después de que el PCD recibe la ATS. El PPS permite al PCD configurar las tasas de bits de comunicaciones en cada dirección.

La norma define dos formatos de bloques: estándares y aumentados. Bloques estándares están enviados en tramas estándares con el CRC16 presente. Bloques aumentados

comienzan con el largo del bloque, y terminan con un CRC32, y el campo INF contiene códigos Hammings para la corrección de errores. La Figura 1.10 muestra los dos. Comunicaciones comienza con bloques estándares hasta que el PCD configura lo contrario.

Prologue field			Information field	Epilogue field
PCB	[CID]	[NAD]	[INF]	CRC16
1 byte	1 byte	1 byte		2 bytes

Length field	Prologue field			Information field	Epilogue field
LEN	PCB	[CID]	[NAD]	[INF]	CRC32
2 bytes	1 byte	1 byte	1 byte		4 bytes

Figura 1.10: El formato de un bloque estándar (arriba) y uno aumentado (abajo) [13]

Hay tres tipos de bloques: I (Information), R (Receive Ready) y S (Supervisory). El campo PCB indica que tipo de bloque es. El CID es la dirección de la PICC y solo está presente si la PICC lo soporta. El campo CID en respuestas también contiene dos bits con información sobre el nivel de potencia recibido, el PCD puede usar esa información para controlar la potencia transmitida por el campo. La NAD (Node Address) permite una PICC tener más de una aplicación, direccionado por este campo, también solo está presente si la PICC la soporta. Finalmente el campo INF contiene la información del bloque, y solo es presente por bloques tipos I y S.

Bloques-S son usados para información de control. Hay tres comandos definidos:

S(WTX)
: Waiting Time eXtension. Si la PICC no está lista para responder a una solicitud en el tiempo permitido, puede responder con un S(WTX) pidiendo más tiempo.

S(DESELCT)
: Este comando es enviado por el PCD cuándo quiere desactivar la PICC.

S(PARAMETERS)
: Este comando está usado para leer o setear la configuración de la PICC. Por ejemplo, para cambiar la tasa de bits, o cambiar entre bloques estándares y aumentados.

Bloques-I son usados para transmitir información al nivel de aplicación. El protocolo de la aplicación no está definido en esta norma. Estos bloques pueden ser encadenados para permitir el envío de un mensaje más grande que el soportado por el destino, partiendo el mensaje en partes de tamaños soportados.

Bloques-R son usados para reconocer la recepción de un bloque: R(ACK) o indicar errores: R(NAK). En el caso de bloques-I encadenados un R(ACK) es enviado para pedir la siguiente parte del mensaje. También el PCD puede enviar un R(ACK/NAK) para pedir la PICC retransmita su última respuesta, esto puede ser usado para recuperar de errores. Finalmente el PCD puede enviar un R(NAK) para verificar la presencia continua de la PICC.

Descripción del Proyecto Marco

Este trabajo se enmarca en un proyecto que tiene como objetivo general desarrollar un circuito integrado capaz de tomar muestras de un sensor MOSFET de radiación y transmitirlas a un dispositivo externo mediante un protocolo propietario implementado encima de la norma ISO/IEC 14443A. Este proyecto marco requiere cinco bloques como se muestra en la Figura 2.1:

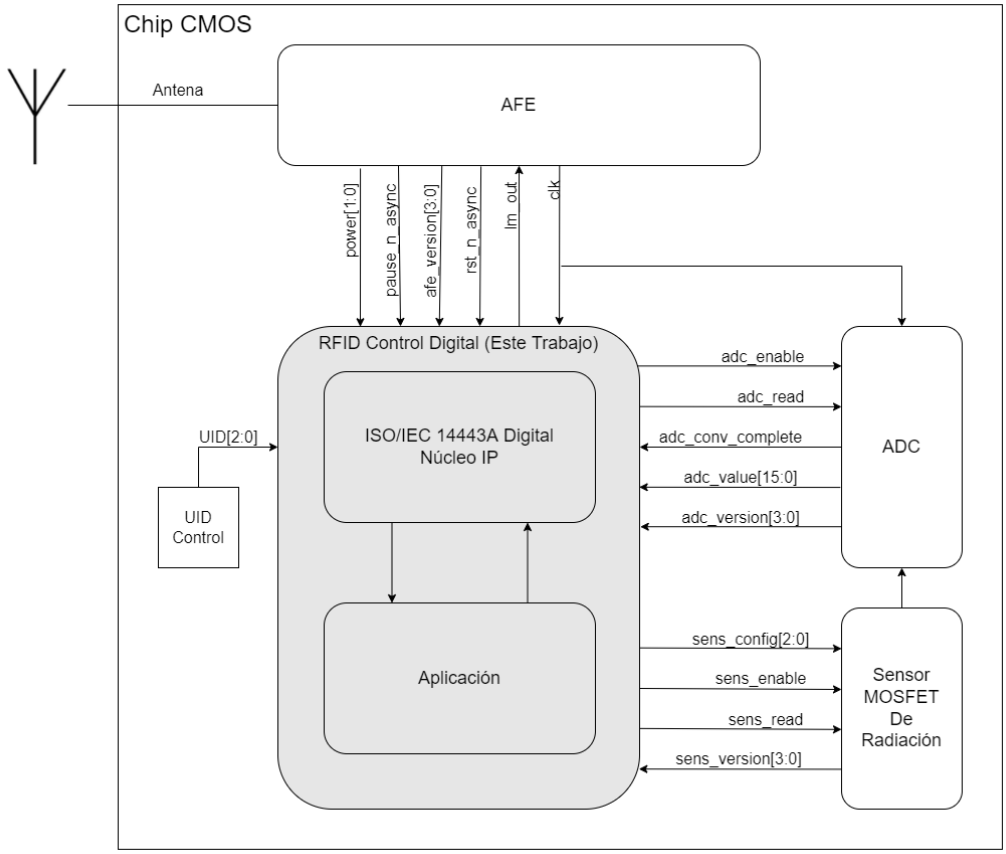

Figura 2.1: Diagrama general del proyecto marco y su relación con el bloque a diseñar en esta Tesis.

El Analogue Front End (AFE)

Es la parte analógica de la norma ISO/IEC 14443A-2.

El sensor MOSFET de radiación

Desarrollado por el Laboratorio de Física de Dispositivos-Microelectrónica (LFDM) de la facultad de ingeniería de la UBA.

El Conversor Analógico Digital (ADC, por sus siglas en inglés)

Encargado de convertir la salida analógica del sensor a una señal digital.

El RFID Control Digital

Es el objetivo de esta Tesis.

El UID Control

Permite asignar la parte única del UID de la PICC.

Algunos de estos bloques no se han desarrollado todavía, por lo que es necesario asumir las especificaciones de los mismos.

El AFE es responsable de:

- Recibir y regular potencia desde el campo electromagnético. La implementación de esta tesis se realiza considerando la tecnología de fabricación de XFAB 180 nm, la cual utiliza celdas estándares que requieren una tensión de 1.8 V, por lo que el AFE debe producir esa tensión, y mantenerla durante las pausas.
- Recuperar la señal de reloj a partir de la portadora. La dificultad en este punto reside en el hecho de que el PCD envía información a la PICC utilizando pausas en la portadora, por lo cual se espera que el reloj se detenga durante estas pausas. Es posible utilizar un PLL para producir un reloj continuo. El circuito de esta tesis es elaborado con habilidad para funcionar con un reloj continuo o con un reloj que se detenga durante las pausas. En la Sección sequence_decode se detalla el número máximo de flancos del reloj que es posible perder conservando una correcta decodificación de las secuencias.
- Manejar la señal de reset (activa baja). Los demás bloques se deben mantener en un estado de reset hasta que la tensión de alimentación sea estable.
- Detectar las pausas enviadas desde el PCD.
- Permitir la transmisión de respuestas al PCD mediante un modulador de carga. El modulador debería ser manejado directamente desde una entrada del bloque, lo cual sería conectado a la salida *lm_out* del bloque RFID Digital Control. La señal se obtiene como el AND lógico de la codificación Manchester y la subportadora.
- Generar una señal indicando el nivel de potencia recibida a través del campo electromagnético (opcional). Si se tiene esta información, la misma puede ser enviada al PCD en el campo CID de mensajes de nivel protocolo. El PCD puede usarla

para ajustar la intensidad del campo electromagnético.

- Proveer una salida de cuatro bits que indique la versión del hardware del AFE. Este valor forma parte de la información enviada al PCD como respuesta al mensaje de protocolo IDENTIFY.

El sensor de radiación tiene tres entradas: *sens_config[2:0]*, *sens_enable* y *sens_read* . Para leer el sensor, primero se debe establecer el valor de *sens_config[2:0]* de manera que se elija la configuración del sensor deseado por el usuario. Posteriormente se debe establecer *sens_enable* en '1' para activar el sensor. Finalmente después del tiempo deseado por el usuario, llevar *sens_read* a '1' configura al circuito del sensor en modo lectura para su muestreo. Por otro lado, el sensor tiene dos salidas, una analógica que está conectada al ADC, y una señal de cuatro bits que indica la versión del sensor. Este valor forma parte de la información enviada al PCD como respuesta al mensaje de protocolo IDENTIFY.

Por su parte el ADC, además de la entrada analógica proveniente desde el sensor, tiene otras dos entradas: *adc_enable* y *adc_read*. El flanco ascendente de la señal *adc_read* indica el momento de muestreo de la salida del sensor. Respecto de las salidas, hay tres: *adc_conversion_complete*, *adc_value[15:0]* y *adc_versión[3:0]*. Cuando el ADC completa la conversión la señal *adc_conversion_complete* toma el valor '1' durante un ciclo del reloj. La señal *adc_value* debe ser estable antes del pulso en *adc_conversion_complete*.

El bloque de UID Control es responsable de especificar los tres bits menos significativos del UID de la PICC. Este valor puede ser configurado mediante una memoria no volátil, pero en esta tesis la intención es configurarlo utilizando wire bonding o resistores pull up/down.

El objetivo de esta tesis es implementar el bloque del RFID Control Digital, que consiste en dos sub-bloques: Una implementación del parte digital de la norma ISO/IEC 14443A y un protocolo a nivel aplicación que permita la lectura del sensor y envíe esta información al PCD. Un requisito de este bloque es que la entrada de UID se mantenga estable mientras el bloque no esté en un estado de reset.

Implementación y Verificación

La implementación de todos los módulos y los bancos de prueba se escriben con el HDL (Hardware Description Language) SystemVerilog, el cual es definido en IEEE 1800 [7]. SystemVerilog es basado en Verilog, formalmente definido en IEEE 1364 [8]. SystemVerilog es conocido principalmente como un HDL para verificación debido a sus extensiones sustanciales a verilog en ese ámbito, por ejemplo en la adicción de conceptos de programación orientados a objetivos (como clases), y aserciones. Además SystemVerilog tiene varias ventajas a Verilog para uso en síntesis [21], por ejemplo:

Enumeraciones (enum) y estructuras (struct)
: Como en el lenguaje de programación C.

always_comb y always_ff
: Estos permiten al ingeniero especificar su intención a implementar lógica combinatoria o secuencial respectivamente. Las herramientas pueden verificar que el circuito inferido cumple con esa intención, por ejemplo, que los bloques combinatorios no contienen latches.

Interfaces
: Colecciones de señales que son frecuentemente usadas juntas para reducir la replicación de código.

La implementación de esta tesis está dividida en dos: 1) El núcleo IP genérico para ISO/IEC 14443A que es la lógica digital necesaria para recibir, decodificar y actuar sobre los mensajes definidos en la norma, y construir, codificar y transmitir las respuestas adecuadas. Este núcleo IP está dividido en tres partes principales, uno para cada parte de la norma (excepto ISO/IEC 14443-1). 2) El sistema de control del sensor y del ADC. La Figura 3.1 muestra todos esos bloques, y el flujo de datos entre ellos.

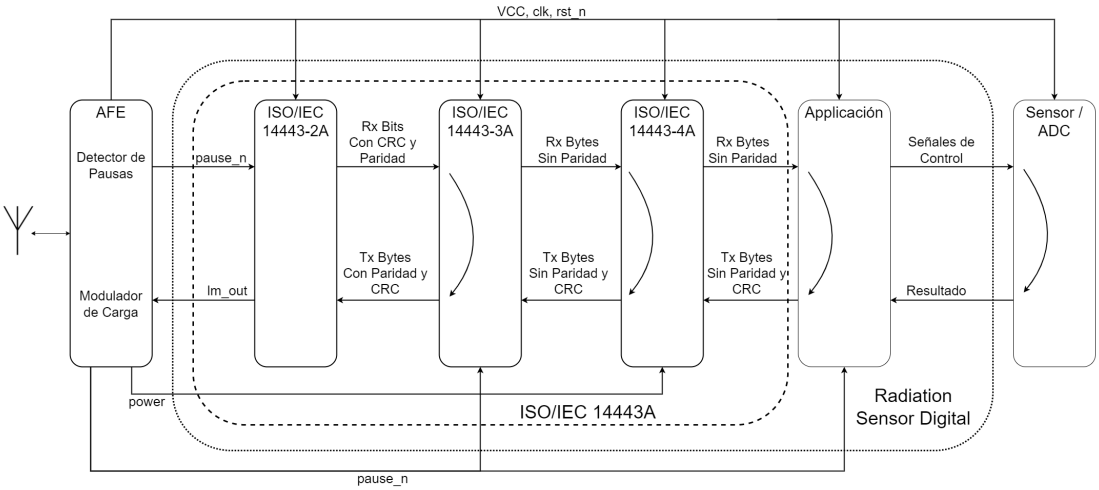

Figura 3.1: Los bloques principales de la implementación.

Interfaces

La norma IEEE de SystemVerilog define un interfaz como:

> A su nivel más bajo, un interfaz es un conjunto nombrado de nets o variables. La interfaz es instanciada en un diseño y puede ser accedida por un puerto como un ítem sencillo; las nets o variables componentes [pueden ser] referidos cuando sea necesario. Una proporción significativa de un diseño verilog frecuentemente consiste en listas de puertos y listas de conexiones, los que son simplemente repeticiones de nombres. La habilidad de reemplazar un grupo de nombres con un nombre simple puede reducir significativamente el tamaño de una descripción y mejorar su mantenibilidad. [7, traducción mía]

Además de esas ventajas, una interfaz puede contener funcionalidad, sea para síntesis o verificación, por ejemplo se puede agregar aserciones que verifican el comportamiento de los nets internos en vez de tener que duplicar esas pruebas en cada sitio que la interfaz es usada. Una interfaz puede tener uno o más modports, los que especifican las direcciones de las señales. Esos modports pueden ser usados en la lista de puertos por un módulo en vez de especificar cada señal y su dirección individualmente.

Este trabajo consiste en varios módulos que contienen uno o más sumideros para recibir tramas desde otros módulos, y una o más fuentes para enviar tramas a otros módulos. Por ejemplo el módulo: **frame_decode** recibe tramas desde el módulo: **sequence_decode**,

quita los bits de paridad, y reenvía las tramas modificadas al módulo: **deserialiser**. Por lo tanto muchos de los módulos manejan el mismo conjunto de señales, lo que es el uso principal por interfaces. Hay dos interfaces definidas en este trabajo, uno para la recepción de tramas (*rx_interface*), y otra para la transmisión de las respuestas (*tx_interface*). Las dos son parametrizadas para funcionar con series de bits o de bytes. Las interfaces son usadas para conectar una fuente en un módulo a un sumidero en otro módulo.

La *rx_interface* contiene:

- *soc*: Un indicador que indica el comienzo de una trama.
- *eoc*: Un indicador que indica el fin de una trama.
- *data*: Un bit / byte de la trama.
- *data_valid*: Un indicador que indica si los datos en *data* son válidos.
- *data_bits*: La cantidad de bits válidos. Por una interfaz configurada a bytes, esta señal permite la recepción de tramas cortas que tienen solo 7 bits, o de tramas anticolisión orientada a bits que pueden terminar con entre uno y ocho bits.
- *error*: Indica la detección de un error en la trama, por ejemplo: un bit de paridad equivocado.

La Figura 3.2 muestra una simulación de la recepción de una trama corta con una *rx_interface* de bits, y su conversión a una serie de bytes. Arriba están las señales en una *rx_interface* de bits, *data_valid* está en '1' siete veces durante la trama, indicando que el dato recibido es: 1,1,1,1,0,1,0. Debido a que el bit menos significativo es enviado primero, este serie representa 7'b0101111. Abajo están las señales en una *rx_interface* de bytes representando la misma trama. El dato tiene valor 8'bX0101111 cuando *data_valid* tiene valor lógico igual a '1'.

La *rx_interface* incluye varias aserciones para verificar el comportamiento de las señales en la interfaz. Las aserciones consideran: Las señales son correctas en el estado de reset, los indicadores nunca son desconocidos, *soc* y *eoc* no están en '1' en el mismo ciclo, y que solo están en '1' por la duración de un solo ciclo del reloj a la vez.

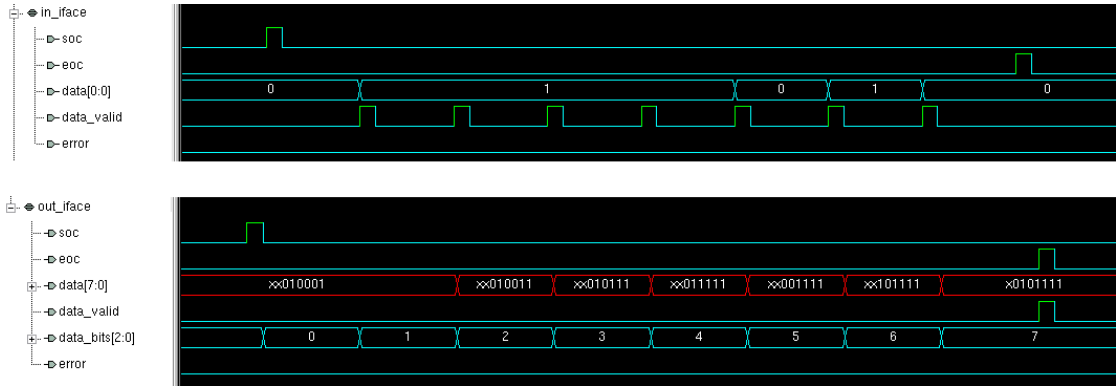

Figura 3.2: Una simulación de la recepción de una trama corta representada con una *rx_interface* de bits (arriba) y su conversión a una *rx_interface* de bytes (abajo).

La *tx_interface* contiene:

- *data*: Un bit / byte de la trama.
- *data_valid*: Un indicador que indica que *data* es válida.
- *data_bits*: La cantidad de bits válidos. Para una interfaz configurada a bytes, esta señal permite la transmisión de tramas de anticolisión orientada a bits, las que pueden comenzar con un byte parcial.
- *last_bit_in_byte*: Esta señal solo existe en interfaces de bits. Indica que el bit actual es el último bit en un byte. Es usada para conocer dónde agregar bits de paridad dentro de una serie de bits.
- *req*: El sumidero usa esta señal para pedir que la fuente envíe el siguiente bit / byte de la trama. La norma ISO/IEC 14443A-2 define la duración de bits como 128 ciclos de la portadora, por lo tanto esta señal es necesaria para limitar la tasa de envío de datos.

Cuando una fuente está lista para transmitir una trama se establece el primer bit / byte de la trama en la señal *data* y fija *data_valid* en '1'. El sumidero puede utilizar la señal *data_valid* para ver si la fuente tiene data enviar. Después de haber leído la primera bit / byte de data, el sumidero fija la señal *req* en '1' durante un ciclo del reloj. La fuente detecta ese pulso y si hay más data a enviar, se actualiza la *data* con el siguiente bit / byte, dejando *data_valid* en '1'. Este proceso repite hasta que la fuente no tiene más data a enviar, y después del último pulso en la *req*, se fija *data_valid* en '0'. La Figura 3.3 muestra una simulación de la transmisión de una trama de 16 bits con una *tx_interface* de bytes, y su conversión a una serie de bits. Arriba están las señales en la *tx_interface*

de bytes enviando la data 0xAE, 0x42. Abajo están las señales en una *tx_interface* de bits representando la misma trama.

La *tx_interface* también incluye tres aserciones que verifican: Las señales son correctas en el estado de reset, *req* es '1' solo para la duración de un ciclo del reloj, y que *data*, *data_valid*, *data_bits* y *last_bit_in_byte* solo cambian estado cuando *data_valid* está '0' o en los cuatro ciclos después de un pulso de req. La última aserción es para verificar que la fuente puede proveer datos cuándo es pedido con la señal *req* antes de que el sumidero la necesita.

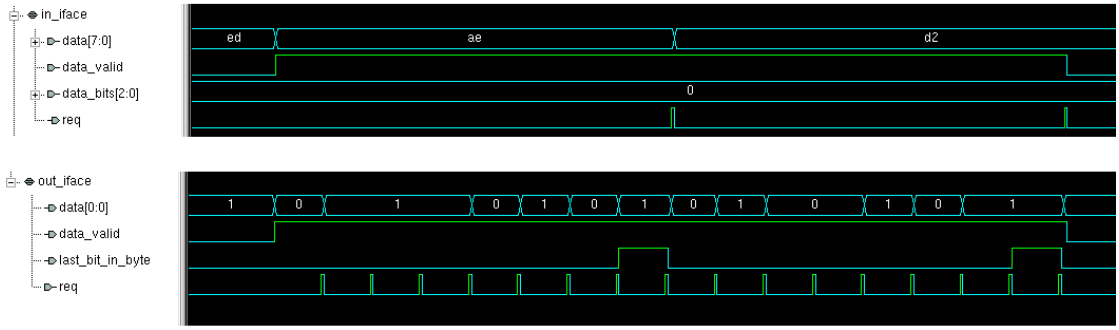

Figura 3.3: Una simulación de la transmisión de una trama de 16 bits representada con una *tx_interface* de bytes (arriba) y su conversión a una *tx_interface* de bits (abajo).

Marco de Verificación

Simulación y verificación es una parte muy importante del diseño digital. En un informe de 2020 sobre las tendencias de diseño y verificación de ICs y ASICs [5], los autores encontraron que en promedio más de 50 % del tiempo de un proyecto es utilizado en verificación. También encontraron que en promedio hay un ingeniero de verificación por cada ingeniero de diseño, además los ingenieros de diseño gastan aproximadamente la mitad de su tiempo verificando sus diseños.

En esta tesis se utiliza verificación funcional mediante simulaciones con la herramienta VCS de Synopsys. Las simulaciones son completamente automatizadas y son ejecutadas con un comando sencillo utilizando un Makefile, por ejemplo: "make serialiser_tb".

UVM (Universal Verification Methodology) es una metodología de verificación funcional mediante un conjunto de clases de SystemVerilog. Aproximadamente 75 % de proyectos

mundiales son verificados con UVM [5]. Una de las ventajas principales de UVM es la modularidad. El código es partido en bloques separados facilitando la reutilización de los componentes en varios bancos de pruebas sin tener que duplicar código. La desventaja principal es la complejidad, requiere mucho código para armar un banco de prueba. UVM no es usada en esta tesis porque su implementación sería más compleja que el propio diseño. Sin embargo el marco de verificación está basado en las técnicas de UVM.

El proceso de verificación es estimular las entradas del DUT (Design Under Test) y verificar que las salidas son las esperadas. Frecuentemente no es posible verificar un diseño para todas las combinaciones y secuencias posibles de las entradas, por lo tanto es común usar estímulo aleatorio. Con estímulo aleatorio siempre hay el riesgo de no verificar parte del diseño, por la posibilidad de no elegir una de las combinaciones o secuencias de entradas necesarias para estimular esa parte del diseño. Una técnica para ayudar a mitigar esto, es: aleatorio restringido (constrained random), esta técnica permite la generación de estímulo aleatorio mediante constraints para restringir el estímulo a un rango interesante en particular. Por ejemplo, en vez de generar tramas completamente aleatorias, pueden ser limitadas: a tramas válidas, una colección de tramas en particular, o tramas con errores.

Una técnica para asegurar que todas las partes de un diseño son verificadas suficientemente, es generar informes de cobertura. La herramienta VCS está habilitada para generar, de forma automática, informes de cobertura de código con varias métricas, y un resultado total de qué proporción del diseño fue verificado. Estos informes pueden ser analizados por el diseñador para verificar cuáles partes de su diseño fue suficientemente estimulado y cuáles partes necesitan más trabajo. Las métricas de cobertura de código habilitadas en VCS son:

- Línea: Muestra las líneas del RTL que fueron ejecutadas.
- Condición: Muestra las sub expresiones booleanas que fueron evaluados a verdadero y falso. Por ejemplo en la declaración: “res = (A == 0) ? B : C”, el informe de cobertura indicará si la expresión “A == 0” fue evaluado al menos una vez a verdadero y al menos una vez a falso.
- Cambio de Estado (Toggle): Muestra cuáles señales y puertas cambiaron de estado en las dos direcciones.
- Branch: Muestra cuáles branches fueron tomados.
- FSM (Finite State Machine): Muestra cuáles estados en un FSM fueron utilizados, y las transiciones entre ellos.

- Aserción: Muestra cuáles aserciones fueron: ejecutadas, aprobadas y falladas, y cuántas veces por cada uno.

El marco de verificación es implementado con varios componentes, la mayoría son clases de SystemVerilog. Los componentes pueden ser divididos en siete grupos distintos:

- Transacciones: Una transacción representa una trama, puede ser una trama de bits, de bytes o de secuencias (como definido en ISO/IEC 14443-2).
- Controladores: Un controlador envía una transacción sobre una interfaz.
- Monitores: Un monitor monitoriza una interfaz, y construye transacciones representando las tramas detectadas.
- Generador de transacciones: Genera transacciones de bytes para representar las tramas definidas en la norma. También puede generar transacciones aleatoriamente.
- Convertidor de transacciones: Produce una transacción en un formato desde una transacción en otro formato. Por ejemplo puede convertir una transacción de bytes a una transacción de bits, opcionalmente agregando los bits de paridad.
- Secuencias: Código compartido para verificar diseños que reciben tramas, actúan sobre ellos, y generan las respuestas. Por ejemplo: cuándo el DUT está en el estado READY, responde a un SELECT con un SAK con los valores esperados, y se transiciona al estado ACTIVE. Ese ejemplo es una prueba utilizada para verificar el comportamiento de cuatro módulos: **initialization**, **iso14443_3a**, **iso14443a**, **radiation_sensor_digital_top**.
- Otros: Este grupo incluye: modelos de los bloques analógicos, una clase para guardar la dirección de una PICC (UID, CID y NAD), y unas interfaces que solo están usadas para verificación.

Transacciones

La Figura 3.4 muestra un diagrama UML (Unified Modelling Language) de las clases de transacciones. La clase base es **Transaction**, es abstracta y contiene dos métodos abstractos: **compare()** que evalúa si dos transacciones son iguales, y **to_string()** que devuelve un string describiendo la transacción. La clase **QueueTransaction** extiende **Transaction**, tiene un parámetro de tipo: *ElemType*, que especifica el tipo de una cola declarada en la clase. La cola es usada para contener los datos de la trama representada por la transacción. Los dos métodos abstractos de **Transaction** son sustituidos, y hay métodos extras para ayudar con la construcción de transacciones. Hay tres clases que

extienden **QueueTransaction**: **ByteQueueTransaction**, **BitQueueTransaction** y **PCDPauseNTransaction**, que definen el parámetro *ElemType* como: logic[7:0] (byte), logic (bit) y **PCDBitSequence** respectivamente. **PCDBitSequence** es una enum definida en un paquete, y consiste en las secuencias X, Y y Z describidas en la Sección ISO/IEC 14443-2. **ByteQueueTransaction** tiene métodos para calcular y agregar CRCs a la transacción, y para convertir la transacción a una cola de bits. **BitQueueTransaction** tiene un método para agregar los bits de paridad.

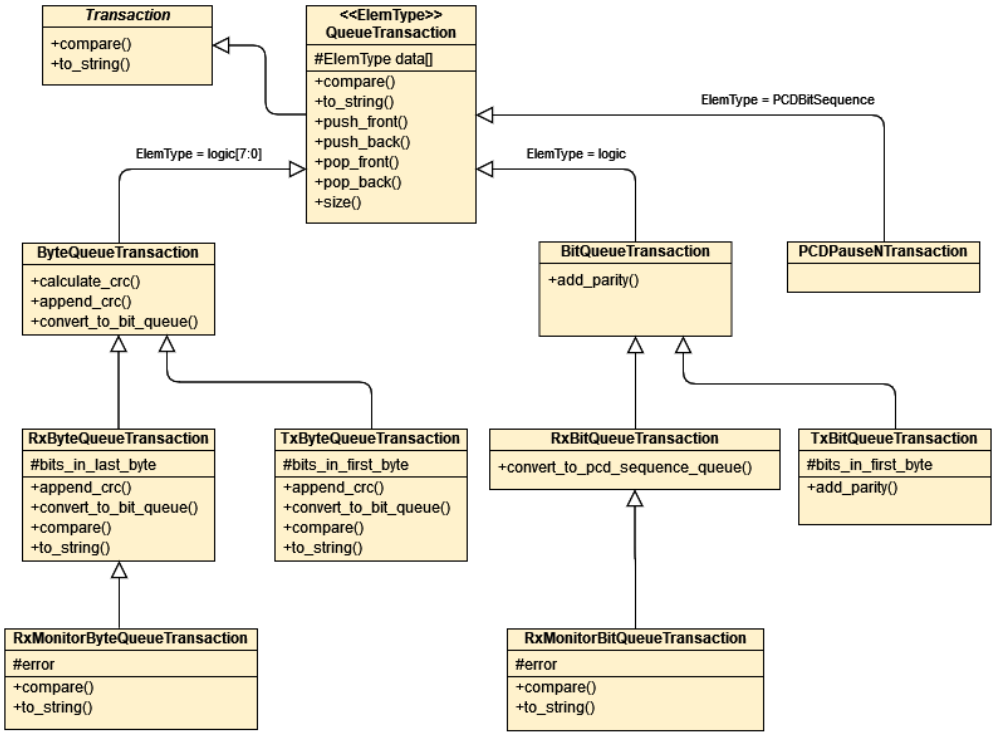

Figura 3.4: Diagrama UML de clases de transacciones.

Para transacciones de transmisión (PICC a PCD), hay dos clases **TxByteQueueTransaction** y **TxBitQueueTransaction**, las dos tienen un atributo especificando cuántos bits hay en el primer byte, eso es porque respuestas a mensajes de ANTICOLISIÓN pueden comenzar con un byte parcial, como se muestra en la Figura 1.7.

Para transacciones de recepción (PCD a PICC), hay cuatro clases en adición a **PCDPauseNTransaction**: **RxByteQueueTransaction**, que tiene un atributo especificando cuántos bits hay en el último byte para tramas cortas y de anticolisiones que pueden terminar con un byte parcial. **RxBitQueueTransaction**, que tiene un método para

convertir su data a una cola de **PCDBitSecuences**. Estas dos clases son usadas en los controladores para enviar transacciones a los DUTs. Las otras dos clases son **RxMonitorByteQueueTransaction** y **RxMonitorBitQueueTransaction**, las que son son las transacciones construidos por los Monitores de recepción. Los dos tienen un atributo para indicar si el monitor ha detectado un pulso en la señal *error* de las interfaces de recepción.

Controladores

La Figura 3.5 muestra un diagrama UML de clases para los controladores. La clase base principal es **Driver**, es abstracta y tiene dos tipos parametrizados: *IfaceType*, que es el tipo de la interfaz que maneja, y *TransType*, que es el tipo de las transacciones que el controlador puede envíar sobre esa interfaz. El método **start()** lleva una referencia a una cola de elementos de tipo **TransType** y comienza un nuevo hilo de ejecución. Ese hilo monitoriza la cola, cuándo una transacción es empujada a la cola desde el banco de prueba, la controladora la quita, y la pasa al método **process()**. En esta clase base **process()** es abstracto, las clases hijas lo sustituyen para definir cómo envíar la transacción sobre su interfaz.

RxIfaceDriver es la clase base para los controladores de las interfaces de recepción (PCD a PICC). **RxByteIfaceDriver** y **RxBitIfaceDriver** la extienden para uso con las interfaces de bytes y de bits respectivamente. Las implementaciones del método **process()** envían el SOC, los datos y el EOC. Esas clases tienen atributos para controlar el timing de las transmisiones, por ejemplo el tiempo entre el SOC y el primer dato, y entre los datos mismos. También esas clases pueden introducir errores en la transmisión pulsando la señal *error* de la *rx_interface* en un sitio aleatorio.

El controlador **PCDPauseNDriver** es parecido al **RxIfaceDriver**, pero en vez de envíar transacciones sobre una *rx_interface*, las envía mediante una señal *pause_n* como haría el detector de pausas en el AFE real.

Los controladores de transmisión (PICC a PCD) están divididos en dos partes: las fuentes y los sumideros. Los sumideros son necesarios porque la señal *req* de la *tx_interface* es manejada por el sumidero. La clase **TxIfaceSinkDriver** es la clase base de los sumideros, contiene un método **start()** que espera por la activación de la señal *data_valid* de la interfaz, y después comienza a pulsar *req* periódicamente hasta que *data_valid* baja a '0'.

Unos atributos controlan los timings de los pulsos. Las clases hijas **TxByteIfaceSinkDriver** y **TxBitIfaceSinkDriver** son usadas con una *tx_interface* de bytes y de bits respectivamente. La clase **TxIfaceSourceDriver** y sus clases hijas son los controladores de las fuentes, y funcionan de la misma manera de los controladores de recepción.

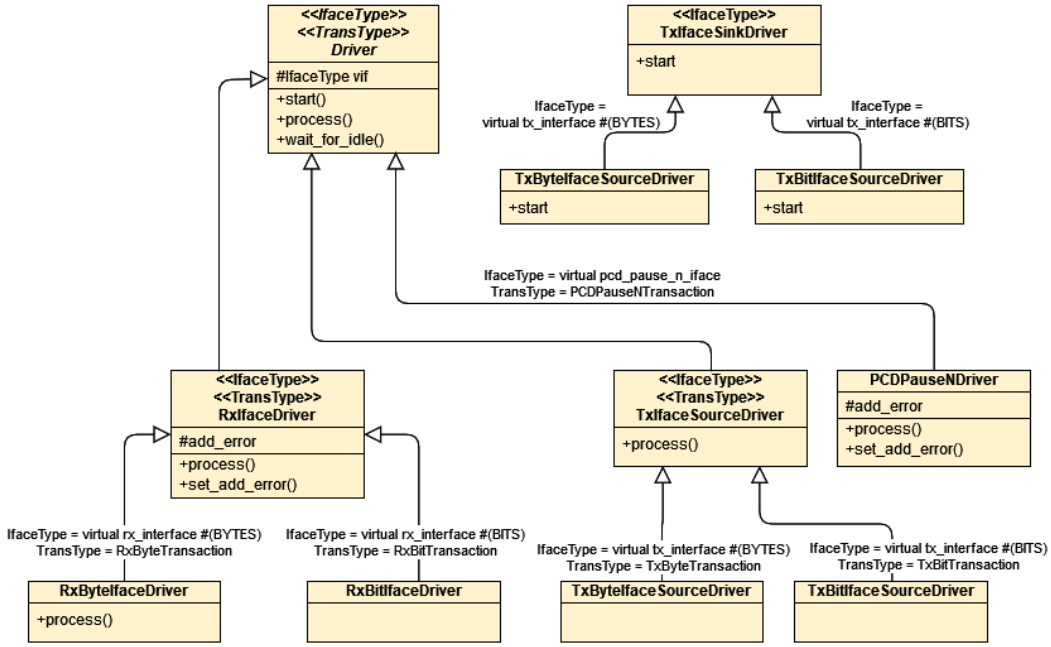

Figura 3.5: Diagrama UML de clases para los controladores.

Monitores

La Figura 3.6 muestra el diagrama UML de las clases para los monitores. La clase base es **Monitor**, es abstracta y tiene dos tipos parametrizados: *IfaceType*, que es el tipo de la interfaz que monitoriza, y *TransType*, que es el tipo de las transacciones que construye. El método **start()** lleva una referencia a una cola de elementos de tipo *TransType* y comienza un nuevo hilo de ejecución. Ese hilo llama el método **process()**, si esa tarea devuelve una transacción válida, es empujada a la cola, para que el banco de prueba pueda verificarla, y el método **process()** es llamado de nuevo. Ese proceso se repite por siempre. En esta clase base **process()** es abstracto, las clases hijas lo sustituyen para definir cómo armar una transacción desde las señales de la interfaz.

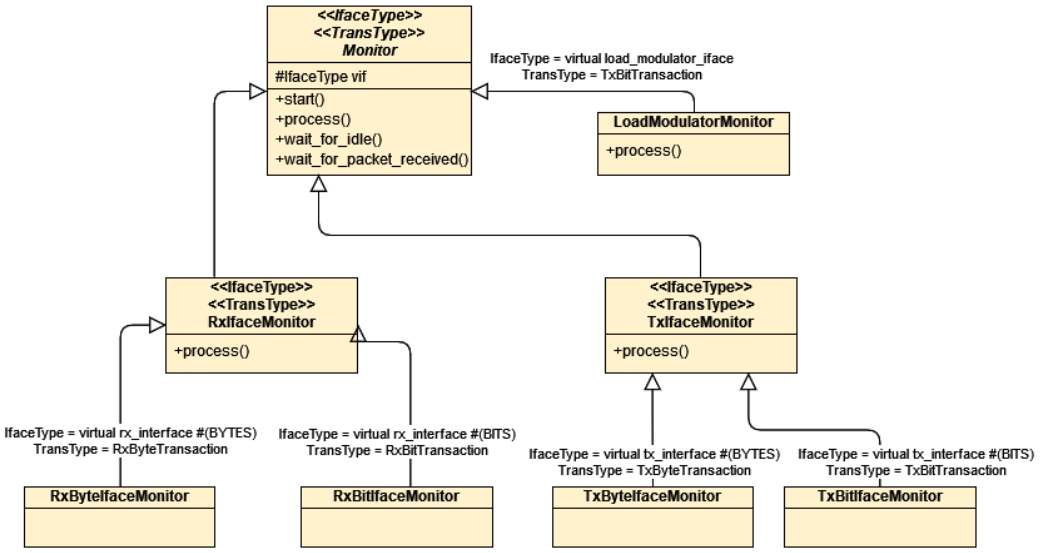

Figura 3.6: Diagrama UML de clases para los monitores.

Para la *rx_interface*, las clases son **RxIfaceMonitor** y sus hijas **RxByteIfaceMonitor** y **RxBitIfaceMonitor**. El método **process()** monitoriza la *rx_interface* para una de las señales: *SOC*, *EOC*, *error* o *data_valid* tener valor '1'. Cuándo *SOC* está en '1' comienza construir una nueva transacción. Cuándo *data_valid* está en '1' el bit o byte actual en la señal *data* es empujado a la cola de datos en la transacción. Cuándo *error* está en '1' la transacción es actualizada para indicar la detección del error. Finalmente cuándo *EOC* está en '1' indicando el fin de la trama, la transacción es devuelta. Si detecta un error, por ejemplo un SOC dentro de una trama, o un EOC fuera de una trama, un error es indicado al usuario mediante la función de sistema **$error()**, la transacción es abandonada, y un error es devuelto.

Los monitores para la *tx_interface* son parecidos a los monitores de recepción, salvo por el método *process()*. Este método primero espera hasta que la señal *data_valid* esté en '1', que indica el comienzo de una trama. Cuándo la señal *req* está en '1', significa que el sumidero ha leído el último dato, así el monitor agrega el bit o byte actual en la señal *data* a la transacción. Cuándo *data_valid* baja a '0', significa que la fuente no tiene más datos a enviar siendo el fin de la trama, y la transacción es devuelta.

Finalmente la clase **LoadModulatorMonitor** es diseñada para monitorear y decodificar la señal *lm_out* que es conectada al modulador de carga en el AFE. El método

process() primero espera por un flanco ascendente de la señal *lm_out*, y después toma una muestra de la señal cada ciclo para 128 ciclos, lo que es la duración para enviar un bit. Si las muestras son iguales al patrón definido en la norma para envíar un '1' o '0' lógico, ese valor lógico es agregado a la transacción. Cuándo todas las muestras están '0's indica el fin de la trama y la transacción es devuelta. Las muestras teniendo cualquier otro patrón indican un error, y el monitor llama la función de sistema **$error()** para avisar al usuario.

Generador y Conversores de Transacciones

El generador de transacciones es una clase **TransactionGenerator**, que tiene métodos para generar transacciones para todos las tramas definidas en la norma. Por tramas de recepción (PCD a PICC) las transacciones generadas son de tipo **RxByteTransactions**. Transacciones de tipo **TxByteTransactions** son generadas por tramas de transmisión (PICC a PCD). También hay métodos para generar transacciones desde una cola de bytes, y para generar transacciones aleatorias restringidas a no ser las tramas definidas en la norma. Las transacciones pueden tener CRCs agregadas automáticamente si es deseado por el usuario. Además es posible pedir que el CRC agregado sea corrompido para verificar como un módulo maneja tramas con errores de CRC.

Los conversores de transacciones son clases que facilitan la conversión de un tipo de transacción a otro. Un ejemplo del uso de un conversor de transacciones es: convirtiendo una **RxByteTransaction** generada por el generador de transacciones a una **RxBitTransaction** que el controlador **RxBitIfaceDriver** puede enviar sobre una *rx_interface* de bits. La clase base abstracta es **TransactionConverter**. Esta clase tiene dos tipos parametrizados: *InputTransType* y *OutputTransType* que especifican los tipos de las transacciones involucradas en la conversión. Las clases son:

- **RxByteToByteTransactionConverter**: Convertir una **RxByteTransaction** a una **RxByteTransaction** (identidad).
- **RxByteToBitTransactionConverter**: Convertir una **RxByteTransaction** a una **RxBitTransaction**, opcionalmente agregando los bits de paridad.
- **RxBitToPCDPauseNTransactionConverter**: Convertir una **RxBitTransaction** a una **PCDPauseNTransaction**.
- **RxByteToPCDPauseNTransactionConverter**: Convertir una **RxByteTransaction** a una **PCDPauseNTransaction**, opcionalmente agregando los bits de paridad.

- **TxByteToByteTransactionConverter**: Convertir una **TxByteTransaction** a una **TxByteTransaction** (identidad).
- **TxByteToBitTransactionConverter**: convertir una **TxByteTransaction** a una **TxBitTransaction**, opcionalmente agregando los bits de paridad.

Secuencias

Las clases de secuencias no deberían ser confundidas con el método que el PCD usa para codificar tramas mediante pausas, estas clases son diseñadas para verificar el comportamiento de módulos que reciben tramas desde el PCD, y responden con las respuestas adecuadas. Contienen una serie de pruebas que: ponen el DUT en un estado conocido, envían una o más tramas, verifican que las respuestas son las esperadas, y que el estado del DUT cambia como se espera. Cada banco de prueba que necesita usar una secuencia debería extender una de las secuencias genéricas para proveer el funcionamiento específico para el DUT.

La clase base es **Sequence**, y contiene atributos:

- *rx_trans_gen* y *tx_trans_gen*, los generadores de transacciones de recepción y transmisión.
- *rx_trans_conv* y *tx_trans_conv*, los conversores de transacciones para recepción y transmisión.
- *rx_driver*, el controlador para enviar tramas sobre la *rx_interface*.
- *rx_send_queue*, una referencia a una cola conectada al *rx_driver*. Las transacciones empujadas a esa cola son enviadas por el controlador.
- *tx_monitor*, el monitor para monitorizar la *tx_interface*.
- *tx_recv_queue*, una referencia a una cola conectada al *tx_monitor*. Tramas detectadas sobre la *tx_interface* son empujadas a esa cola.

Y estos métodos:

- **send_transaction()**: Lleva una **RxByteTransaction**, la convierte a una transacción del tipo requerido mediante el conversor de transacciones *rx_trans_conv*, la empuja a la cola *rx_send_queue* y espera para el controlador *rx_driver* terminar enviarla.
- **send_*()**: Hay un método para cada trama definida en la norma: por ejemplo **send_reqa()** o **send_select()**. Generan una transacción para la trama mediante el generador de transacciones *rx_trans_gen*, y la pasa a **send_transaction()**.

- **wait_for_reply()**: Espera para el monitor *tx_monitor* a indicar que ha recibido una transacción, quita la transacción de la cola *tx_recv_queue* y la devuelve. Hay una aserción para confirmar que la transacción es recibida dentro de un periodo de tiempo especificado.
- **verify_no_reply()**: Espera un periodo de tiempo especificado para confirmar que el DUT no envía una trama.
- **verify_trans()**: Verifica que una transacción recibida es igual a la que es esperada.
- **sequence_callback()**: Los bancos de pruebas pueden sustituir ese método para permitir actuar sobre eventos de forma específica al banco de prueba.

La clase **SpceficTargetSequence** extiende **Sequence** con la intención de verificar el comportamiento de una PICC en particular, o parte de su diseño. El UID de la PICC es conocido, permitiendo la predicción de sus acciones y respuestas. Esa clase sustituye unos de los métodos de la clase base para seguir y verificar el estado de la PICC, por ejemplo si la PICC es en estado IDLE y una REQA es enviada, la PICC debería transicionar al estado READY y enviar la respuesta ATQA. También hay unos métodos definidos que envían las tramas necesarias para inducir al DUT a transicionar a un estado especificado.

Siguiente la clase **CommsTestsSequence** extiende **SpecificTargetSequence**. Esta clase contiene una serie de pruebas para verificar el comportamiento de una PICC, o parte de su diseño. Las pruebas específicas son descritas más adelante en la sección de implementación cuándo son usados. Un ejemplo de una prueba típica es:

- Transicionar al estado READY o READY* aleatoriamente.
- Enviar la trama SELECT con el UID de la PICC pero con CRC corrompida.
- Verifica que el DUT no responde.
- Verifica que el DUT está en estado IDLE o HLTA dependiendo si el estado original estuvo READY o READY* respectivamente.

Otros

Los controladores y monitores siendo clases no pueden utilizar señales de forma directa, tienen que usar una interfaz, por lo tanto hay dos interfaces más definidas para uso solo en verificación:

- *pcd_pause_n_iface*: Es usada con el controlador **PCDPauseNDriver**. Contiene una señal de un bit: *pause_n*, que representa la entrada desde el detector de pausas del AFE.

- **load_modulator_iface**: Es usada con el monitor **LoadModulatorMonitor**. También consiste en una señal de un bit: *lm*, que representa la salida hasta el modulador de carga del AFE.

La clase de ayuda **UID** está diseñada para ayudar con operaciones relacionadas con el UID de una PICC. Puede ser usada para guardar un UID simple, doble o triple. También puede ser usada para comparar los datos en una trama ANTICOLLISION o SELECT con el UID de la PICC, para conocer si es esperado que la PICC respondería a esa trama. La clase **FixedSizeUID** extiende **UID**, y tiene un parámetro para definir si el UID es simple, doble o triple. También tiene dos parámetros para especificar cuántos de los bits más significativos son constantes, y el valor de esos bits. Los bits menos significativos pueden ser generados aleatoriamente, considerando unas restricciones para asegurar que el UID generado es válido. Esta clase está diseñada en esta forma, porque la implementación del núcleo IP en este trabajo opera de la misma manera.

La clase **StdBlockAddress** guarda detalles sobre los campos CID y NAD de los bloques estándares definidos en la parte cuatro de la norma. Esta clase es usada para generar transacciones que representan tramas de bloques estándares.

La clase **Target** contiene información sobre una PICC, y es usada para ayudar las secuencias seguir el estado de la PICC. La información guardada consiste en: La **UID**, la **StdBlockAddress** actual, y el nivel de lazo de anticolisión actual. Además contiene métodos ayudantes para operaciones comunes, por ejemplo para decodificar si una **StdBlockAddress** es direccionada a esta PICC o no.

Modelos Analógicos

Un PCD genera una señal portadora de 13,56 MHz y la usa para enviar información a PICCs en su campo electromagnético mediante la modulación de la amplitud de esa portadora. El detector de pausas en el AFE de la PICC detecta esas pausas y genera la salida digital *pause_n* para indicar cuándo las pausas son detectadas. La pausa detectada por el AFE es una distorsión de la pausa ideal enviada del dominio digital del PCD. Los dos flancos de la pausa son retrasados debido a las implementaciones del circuito de modulación de amplitud del PCD y del detector de pausas del AFE. También esa distorsión puede retardar los flancos descendente y ascendente por tiempos diferentes, cambiando la duración de la pausa. Además el AFE de la PICC recupera el reloj desde la portadora, y por lo tanto el reloj se detiene durante las pausas. Debido a las implementaciones del circuito de modulación de amplitud del PCD y de la recuperación del reloj del AFE, el

reloj puede sigue andando por unos ciclos después de que el PCD comienza la pausa, y el reloj puede seguir detenido por unos ciclos después de que el PCD termina la pausa. Para simular este comportamiento, un modelo del PCD y del AFE fue implementado. La Figura 3.7 muestra ese modelo.

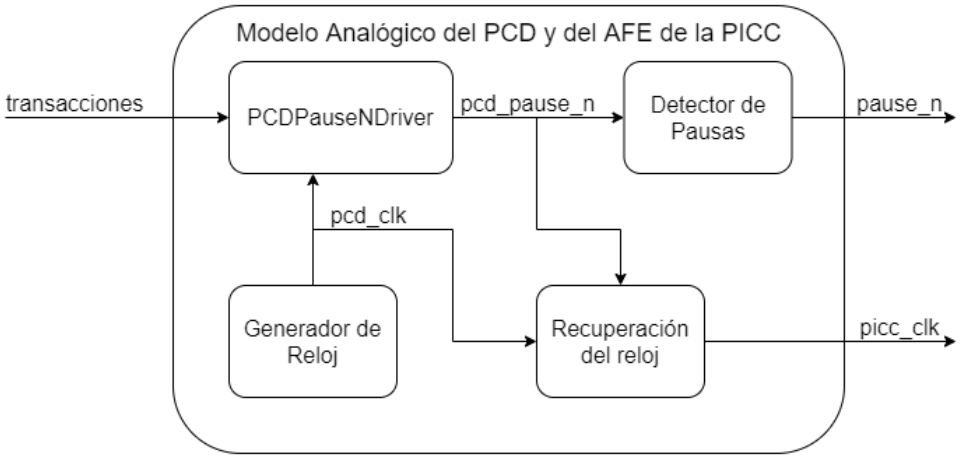

Figura 3.7: Modelo del PCD y del AFE de la PICC para uso en simulaciones.

El modelo consiste en cuatro bloques:

- Generador de Reloj: Genera un reloj continuo de 13,56 MHz que es un modelo del reloj del PCD.
- **PCDPauseNDriver**: El controlador que envía transacciones mediante pausas. Genera una señal *pcd_pause_n* que es la pausa ideal dentro del PCD. La duración de la pausa puede ser elegida por el usuario.
- Detector de Pausas: Un modelo del bloque con el mismo nombre en el AFE de la PICC. Genera una señal: *pause_n* que es una versión retrasada de la señal *pcd_pause_n*. Los retardos del flanco ascendente y el flanco descendente pueden ser elegidos por el usuario individualmente.
- Recuperación del Reloj: Un modelo del bloque con el mismo nombre en el AFE de la PICC. Genera un reloj basado en lo del PCD, que se detiene durante las pausas. Los retardos entre el comienzo de una pausa y el reloj deteniéndose, y entre el fin de una pausa y el reloj comenzando de nuevo, pueden ser elegidos por el usuario individualmente. Además el usuario puede especificar que el reloj no se detiene,

por lo tanto este modelo puede simular el comportamiento de un AFE dónde el reloj es continuo.

La Figura 3.8 muestra dos simulaciones de una pausa con argumentos de timing diferentes. En el momento de escritura de esta tesis los timings por defectos son basados en simulaciones SPICE del AFE implementado por Alcalde Bessia por otro proceso de fabricación [1]. Cuándo exista una implementación del AFE para el proceso de fabricación XFAB XH018 es altamente sugerido que los valores en este modelo sean actualizados para adecuarse la situación real.

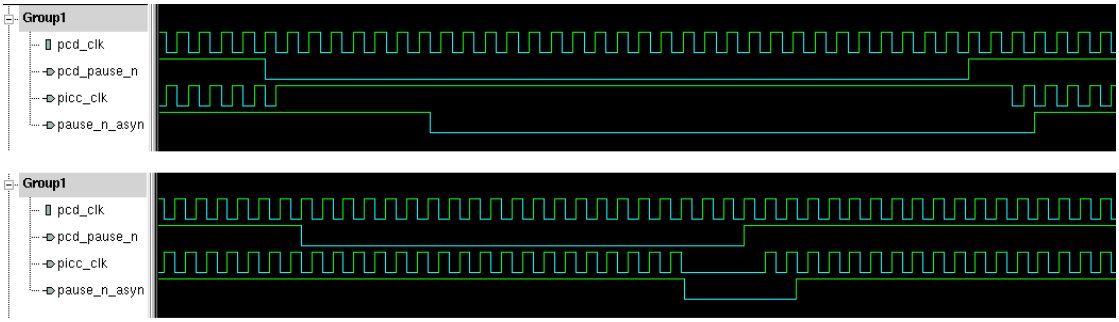

Figura 3.8: Dos ejemplos con timings diferentes del modelo analógico del PCD y el AFE de la PICC.

Estructura de los Bancos de Pruebas

La mayoría de los módulos en esta tesis son verificados con un banco de prueba que tiene una de las dos estructuras mostrado en la Figura 3.9.

Los módulos que simplemente reciben tramas, operan sobre ellas, y las reenvían, son verificados con un banco de prueba sin una secuencia (izquierda). El bloque de control genera una transacción en el formato adecuado y la pasa al controlador mediante un FIFO. El DUT la recibe por su interfaz de sumidero, la procesa, y la reenvía sobre su interfaz de fuente. El monitor detecta la trama reenviada, construye una transacción, y la pasa al bloque de control mediante otro FIFO. Finalmente el bloque de control compara esas transacciones para verificar que la trama fue modificada correctamente.

Los módulos que reciben solicitudes del PCD y envían respuestas, son verificados con un banco de prueba que incluye una secuencia (derecha). El banco de prueba define una nueva clase de secuencia que extiende una de las secuencias genéricas, y sustituye varios métodos para implementar funcionamiento específico a ese banco de prueba. Esos bancos

de pruebas funcionan de una manera parecida a los bancos de prueba sin secuencias, pero las tramas enviadas son elegidas por las pruebas en la secuencia, y son generadas por el generador de transacciones. Además las respuestas enviadas por el DUT son comparadas en la secuencia con las respuestas esperadas.

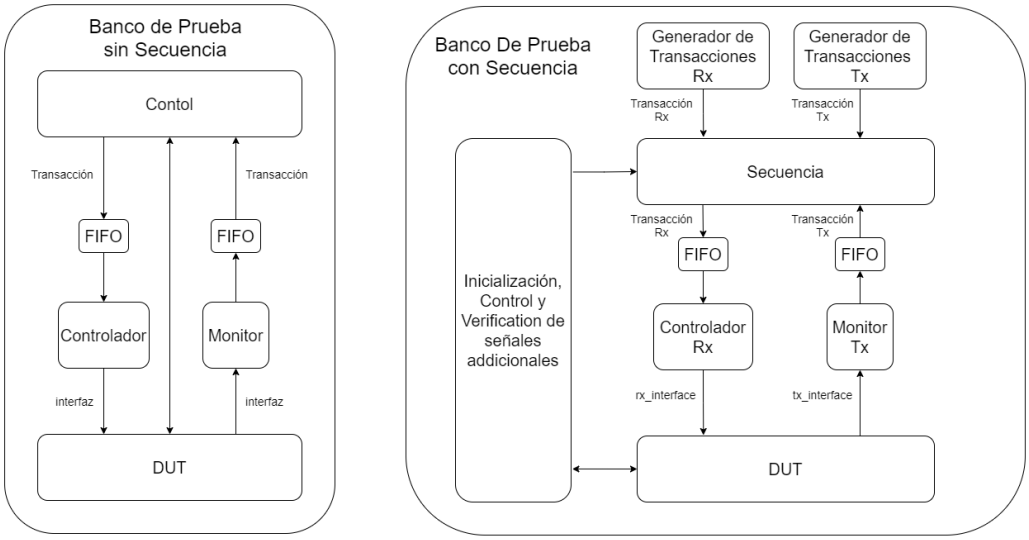

Figura 3.9: Dos estructuras comunes de los bancos de pruebas.

ISO/IEC 14443A núcleo IP

Está parte de la tesis está diseñada para ser una implementación digital genérica de la norma ISO/IEC 14443 para PICCs tipo A. Cuándo se combina con una AFE puede ser usado para cualquier proyecto que requiera funcionar como una PICC tipo A. La implementación se divide en tres partes principales para las partes dos, tres, y cuatro de la norma.

ISO/IEC 14443-2A

Está parte de la implementación consiste en cinco módulos y sus bancos de pruebas correspondientes. Para la ruta de recepción de tramas hay un solo módulo: **sequence_decode**, que convierte una serie de pausas a tramas de bits. La ruta de transmisión consiste en tres módulos que convierten tramas de bits a una señal apta para manejar

el modulador de carga. Las módulos son:

- **subcarrier**: Genera la subportadora.
- **bit_encoder**: Genera la codificación manchester desde una trama de bits.
- **tx**: Recibe la trama a transmitir, agrega el SOC, y produce la salida al modulador de carga mediante los módulos: **subcarrier** y **bit_encode**.

Finalmente hay un módulo: **iso14443_2a** que es el módulo más alto de esta parte de la norma.

subcarrier

Este módulo genera la señal de la subportadora cuándo pedido. Las entradas y salidas son descritas en el Cuadro 3.1. La norma dicta que la subportadora debería tener frecuencia igual a $f_c/16$, y que el periodo de bit debería comenzar con el estado cargado de la portadora [11]. Esto significa que la salida *subcarrier* es una onda cuadrada que comienza con valor '1', y tiene un periodo de 16 ciclos del reloj.

Nombre	Dirección	Descripción
clk	Entrada	Reloj.
rst_n	Entrada	Reset activo bajo.
en	Entrada	Señal de activación.
subcarrier	Salida	Subportadora.

Cuadro 3.1: Entradas y Salidas del módulo **subcarrirer**.

La implementación consiste en un contador de tres bits, cuándo el módulo no está activado el contador es detenido con valor cero, y la salida *subcarrier* está '0'. Cuándo la señal *en* sube a '1', la salida *subcarrier* es asignada a '1' y el contador comienza a contar. Cada ocho ciclos del reloj el contador completa su cuenta, y la salida *subcarrier* invierte su estado lógico.

El banco de prueba consiste en activar el DUT tres veces. La primera vez es activado por 256 ciclos (dos duraciones de bits). La segunda vez es activado por seis ciclos, para que la subportadora es '1' cuando se desactiva el DUT. Finalmente la tercera vez el módulo es activado para trece ciclos, para que la subportadora es '0' cuando se desactiva el DUT. Hay cuatro aserciones que verifican el comportamiento:

- La subportadora está '0' mientras el DUT está en reset.
- La subportadora está '0' mientras el DUT está desactivado.

- Un ciclo después del flanco ascendente de la señal *en* el subcarrier está '1'.
- Cuando el DUT está activado la subportadora tiene un periodo de 16 ciclos del reloj con un ciclo de trabajo de 50 %.

El informe de cobertura da un resultado de 100 % cobertura.

bit_encoder

Este módulo es un sumidero por la *tx_interface*, recibe tramas de bits y las convierte a su codificación Manchester con duración de bit de 128 ciclos del reloj. Las entradas y salidas son descritas en el Cuadro 3.2.

Nombre	**Dirección**	**Descripción**
clk	Entrada	Reloj.
rst_n	Entrada	Reset activo bajo.
en	Entrada	Señal de activación.
in_iface	Entrada	Sumidero de la, *tx_interface*. Las tramas a transmitir.
encoded_data	Salida	Codificación Manchester de cada bit.
last_tick	Salida	Indica el último ciclo de cada periodo de bit.

Cuadro 3.2: Entradas y Salidas del módulo **bit_encoder**.

La implementación de este módulo consiste en un contador de siete bits para contar la duración de bit de 128 ciclos. Cuando el módulo no está activado el contador tiene valor cero, y cuando está activado cuenta repetitivamente. Al principio de cada duración de bit, eso es decir cuándo el contador tiene valor cero y el módulo está activado, la salida *encoded_data* es asignada al valor del bit actual presente en la *in_iface*. En el medio del periodo del bit, cuando el contador tiene valor 64, la salida *encoded_data* se invierte. Esto significa que para envíar el bit 0, *encoded_data* está '0' por 64 ciclos y después '1' por 64 ciclos más. La señal *tx_interface.req* está pulsada por un ciclo del reloj en el medio del periodo de bit, para pedir que la fuente prepare el siguiente bit. Finalmente la salida *last_tick* está pulsada cuándo el contador tiene valor 127, esta salida es usada para determinar cuándo desactivar este módulo después del último bit de la trama.

El banco de prueba tiene un **TxBitIfaceSourceDriver** para envíar tramas al DUT. No utiliza un monitor, ya que es el único módulo que genera una salida en este formato. Antes de envíar cada trama, la transacción es convertida a una cola que consiste en los valores esperados por cada ciclo de la trama. Una aserción verifica que la salida *encoded_data* del DUT corresponde con esa cola. Las pruebas consisten en:

- Enviar dos transacciones de un solo bit cada una, con valores ‘0’ y ‘1’.
- Enviar dos transacciones de dos bit cada una, con valores “00” y “10”.
- Enviar mil transacciones de data y tamaño aleatorios.

Las aserciones son:

- Cuándo el DUT está en reset o no está activado, las salidas: *in_iface.req* y *last_tick* están ‘0’s.
- La salida *last_tick* solo está ‘1’ para un ciclo del reloj a la vez.
- Mientras el DUT está activado, las salidas *in_iface.req* y *last_tick* pulsan por un ciclo del reloj, cada 128 ciclos.
- Después del flanco ascendente de la entrada *en*, la salida *last_tick* está ‘0’ por 127 ciclos del reloj y ‘1’ en el siguiente ciclo.

El informe de cobertura da un resultado de 100 % por el DUT.

tx

El módulo **tx** instancia los módulos: **subcarrier** y **bit_encoder**, e incluye una máquina de estados sencilla que: agrega el SOC a las tramas, activa los submódulos, y manejar la señal *tx_interface.req* para pedir que la fuente prepare el siguiente bit. Finalmente la señal *lm_out* es el AND lógico de la subportadora y la señal *encoded_data*, esta salida es registrada para prevenir glitches llegando al modulador de carga. Las entradas y salidas son descritas en el Cuadro 3.3.

Nombre	**Dirección**	**Descripción**
clk	Entrada	Reloj.
rst_n	Entrada	Reset activo bajo.
in_iface	Entrada	Sumidero de la *tx_interface*.
lm_out	Salida	Señal hacia el modulador de carga.

Cuadro 3.3: Entradas y Salidas del módulo **tx**.

El banco de prueba tiene un controlador **TxBitIfaceSourceDriver** y un monitor **LoadModulatorMonitor**. Las pruebas son:

- El SOC es agregado correctamente. Esto es hecho en el monitor, que verifica que todas las tramas comienzan con un SOC.
- La salida *lm_out* está en ‘0’ cuándo la fuente no está enviando una trama.
- Tramas son enviadas correctamente. Cien transacciones aleatorias son enviadas y

las transacciones detectadas por el monitor son comparadas a las enviadas dentro de una aserción.

El informe de cobertura da un resultado de 92 % por el DUT. Las partes que faltan cobertura no son relevantes, por ejemplo no hay una transición desde el estado State_SOC al estado State_IDLE, ya que no está permitido transmitir una trama SOC aislada.

sequence_decode

El módulo **sequence_decode** convierte series de pausas a tramas de bits, y las envía al siguiente módulo mediante una fuente de la *rx_interface*. La norma ISO/IEC 14443-2 [11] especifica que la duración de bit por recepción es 128 ciclos de la portadora. La existencia de una pausa dentro de una duración de bit y su ubicación determina qué tipo de secuencia es: X, Y o Z. Una pausa al principio de la duración de bit es una secuencia Z, una pausa en el medio de la duración de bit es una secuencia X, y una duración de bit sin pausas es una secuencia Y. La Figura 1.3 muestra estas secuencias. Cada trama comienza con una secuencia Z (SOC). Un '1' es transmitido con una secuencia X. La secuencia usada para transmitir un '0' depende en la última secuencia. Una secuencia Y es usada cuando la última secuencia fue una X, y una secuencia Z es usada en otros casos. Cada trama termina con un EOC, que es transmitido con un '0' lógico seguido por la secuencia Y. Finalmente una secuencia X seguida por una secuencia Z no es permitido y debería estar contado como un error.

En teoría el proceso de convertir una serie de pausas a secuencias es: contar el número de ciclos del reloj en cada duración de bit para determinar dónde comienza la pausa. En la práctica es más complicado debido a que el reloj puede detenerse durante las pausas. Además el tiempo que el reloj está detenido depende de la implementación del bloque de la recuperación del reloj del AFE, y las propiedades del PCD. Debido a que este núcleo IP está diseñado para ser genérico, tiene que funcionar con diferentes AFEs y diferentes PCDs, por lo tanto este módulo fue implementado para funcionar con un amplio rango de comportamientos de relojes. La estrategia propuesta para el módulo **sequence_decode** consiste en contar ciclos de clock entre flancos ascendentes de la señal *pause_n*. Luego a partir del valor de la cuenta se infiere la secuencia recibida. A continuación se analizan las condiciones para aplicar este método de decodificación.

La Figura 3.10 muestra dos secuencias Z consecutivas. *PCD CLK* y *PICC CLK* son los relojes en el PCD y la PICC respectivamente, *pcd_pause_n* es la pausa digital que el PCD transmite, *picc_pause_n* es la pausa detectada en el AFE, *PICC CLK ACTIVE* indica

cuándo el reloj de la PICC está activo. El tiempo entre los dos flancos ascendentes de las pausas medido en ciclos del reloj del PCD es $T_b = 128$. Si el reloj de la PICC se detiene por T_{nc} ciclos durante las pausas, la PICC mediría $T_{z2z} = T_b - T_{nc}$ ciclos entre los flancos ascendentes de las pausas.

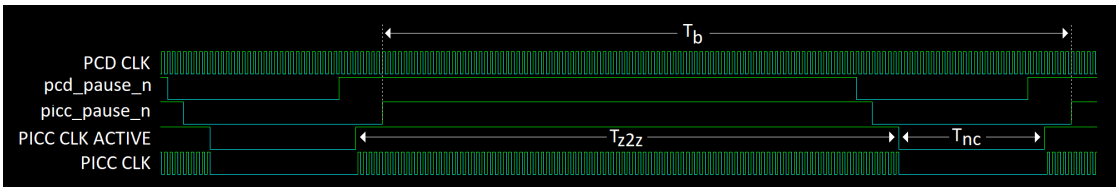

Figura 3.10: Diagrama de timing de dos pausas y el comportamiento del reloj de la PICC.

La Figura 3.11 muestra el número de ciclos entre dos pausas para todas las combinaciones de secuencias posibles. Usa la misma convención de la Figura 3.10, T_{z2x} por ejemplo representa la cantidad de ciclos que la PICC mide entre los flancos ascendentes de las pausas cuándo el PCD transmite una secuencia Z seguida por una secuencia X. Esta convención sigue válida aun cuándo el AFE produce un reloj continuo, y por lo tanto $T_{nc} = 0$.

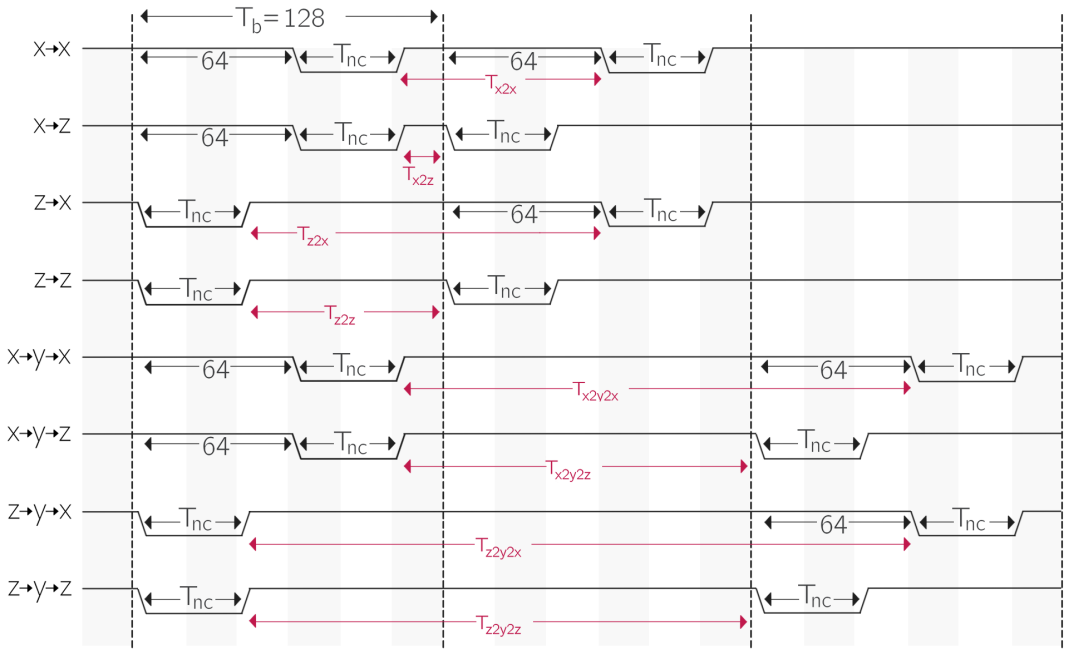

Figura 3.11: Números de ciclos posibles entre dos pausas.

Se define T_{nc_min} y T_{nc_max} como el número de ciclos mínimos y máximos que el reloj puede estar detenido para que las secuencias puedan ser decodificadas correctamente. Este análisis es necesario para considerar variaciones en el comportamiento de los PCDs y los AFEs. Considerando las combinación de secuencias: Z $\rightarrow$ Z, y Z $\rightarrow$ X y observando la Figura 3.11, se puede calcular:

$$T_{z2z\_max} = 128 - T_{nc\_min}$$
$$T_{z2x\_min} = 192 - T_{nc\_max}$$

Conociendo que $T_{z2x\_min} > T_{z2z\_max}$, para que se pueda distinguir entre una Z $\rightarrow$ X y una Z $\rightarrow$ Z, se puede derivar:

$$192 - T_{nc\_max} > 128 - T_{nc\_min}$$
$$T_{nc\_max} < T_{nc\_min} + 64$$

Para soportar AFEs que usan un PLL para proveer un reloj continuo, debería aceptarse T_{nc_min} igual a 0. Además es posible que haya jitter en la detección de los flancos ascendentes de las las pausas en el AFE. Ese jitter puede causar que una PICC cuente un número de ciclos ligeramente diferente entre dos secuencias cada vez que ocurren. Ese comportamiento es igual a si el jitter fue en el número de ciclos que el reloj está detenido. En esta implementación es requerido que ese jitter es a lo máximo tres ciclos, por lo tanto se usa un $T_{nc\_min} = -3$, lo que da un $T_{nc\_max} < 61$.

Considerando el caso de una secuencia X seguido por una secuencia Y se requiere que: $T_{x2y\_min} > T_{x2x\_max}$, es decir que después de una secuencia X, el momento que este módulo emite una secuencia Y tiene que ser después del último momento que puede llegar una pausa que indicaría otra secuencia X. Por lo tanto:

$$T_{x2y\_min} > 128 - T_{nc\_min}$$
$$T_{x2y\_min} > 131$$

Eligiendo $T_{x2y\_min} = 132$, significa que si no hay una pausa detectada por más de 131 ciclos después de una secuencia X, se asume el arribo de una secuencia Y. Finalmente considerando el serie de secuencias: X $\rightarrow$ Y $\rightarrow$ Z:

$$\begin{aligned} T_{x2y2z\_min} &= 192 - T_{nc\_max} \\ &= T_{x2y\_min} + T_{y2z\_min} \\ &= 132 + T_{y2z\_min} \end{aligned}$$

$$\begin{aligned} 192 - T_{nc\_max} &= 132 + T_{y2z\_min} \\ T_{y2z\_min} &= 60 - T_{nc\_max} \end{aligned}$$

Una $T_{y2z\_min} = 0$, implica que la secuencia Y puede ser detectada en el mismo ciclo de un flanco ascendente de una pausa por una secuencia Z. Para simplificar el diseño esto no se permite, así:

$$\begin{aligned} T_{y2z\_min} &> 0 \\ 60 - T_{nc\_max} &> 0 \\ T_{nc\_max} &< 60 \end{aligned}$$

En resumen es posible diseñar un sistema que decodifica una serie de pausas a secuencias, siempre que el AFE pueda garantizar que el reloj nunca se detiene por más de 59 ciclos. Para soportar hasta ± 3 ciclos de jitter en la detección de las pausas como especificado arriba es recomendado que el reloj no se detenga por más de 56 ciclos. El mismo diseño

va a funcionar con un AFE que produce un reloj continuo. Estos requisitos por el AFE son razonables, debido a que la duración de pausa máxima determinada con la Figura 1.2 y el Cuadro 1.1 es 56,5 ciclos del reloj, medido entre los puntos 1 y 2, dónde él amplitud de la portadora es a 90 % de su original.

Usando $T_{nc\_min} = -3$, y $T_{nc\_max} = 59$, es posible diseñar un sistema que decodifica pausas a secuencias, dependiendo de la última secuencia detectada y el número de ciclos contados entre los flancos ascendentes de las pausas. La decodificación usada es mostrada en el Cuadro 3.4. Valores fuera de estos rangos son considerados errores.

Última Secuencia	**Mín Ciclos**	**Máx Ciclos**	**Proxima Secuencia**
X	69	131	X
	132 (timeout)	-	Y
Z	69	131	Z
	132	195	X
	196 (timeout)	-	Y
Y	1	64	Z
	65	127	X
	128 (timeout)	-	Y

Cuadro 3.4: Decodificación de pausas basado en la última secuencia y el tiempo entre pausas.

El Cuadro 3.5 muestra las entradas y salidas de este módulo. La implementación consiste en dos partes. La primera parte determina la secuencia desde las pausas detectadas. Comienza en un estado inactivo, cuándo detecta la primera pausa emite una secuencia Z (SOC). Después emite secuencias cuándo pausas están detectadas o cuándo timeouts ocurren usando el método descrito arriba. Después de detectar dos secuencias Y consecutivas, eso es dos duraciones de bits sin pausas, se vuelve al estado inactivo, y espera por la próxima trama. En el caso de detectar un error, no emite más secuencias y después de haber detectado tres duraciones de bits sin pausas vuelve al estado inactivo. La segunda parte de este módulo decodifica las secuencias a una trama de bits y lo envía sobre la *out_iface* para el posterior procesamiento en otros módulos.

Nombre	**Dirección**	**Descripción**
clk	Entrada	Reloj.
rst_n	Entrada	Reset activo bajo.
pause_n_synchronised	Entrada	Pausa, activa baja, sincronizada.
out_iface	Salida	Fuente de la *rx_interface*.

Cuadro 3.5: Entradas y Salidas del módulo **sequence_decode**.

Para verificar este módulo el banco de prueba envía varias transacciones mediante el controlador **PCDPauseNDriver** en el modelo del AFE, que maneja la entrada del DUT *pause_n_synchronised*. El monitor **RxBitIfaceMonitor**, monitoriza la salida *out_iface* y emite transacciones por las tramas detectadas. Esas transacciones son comparadas con las enviadas para verificar que el DUT decodifica correctamente las pausas. Las transacciones enviadas son:

- La transacción ZZXXYZXYXYY, y la transacción ZXYZY. Estas dos transacciones verifican todas las filas en el Cuadro 3.4.
- Cincuenta transacciones aleatorias sin errores.
- La transacción: ZXZZXYXYY para verificar que la transición Z → X es detectada como un error, y que secuencias después del error son ignoradas.

Estas pruebas son repetidas varias veces con diferentes timings configurados en el modelo del AFE. Los parámetros de timings son:

- La duración de la pausa.
- Si el reloj se detiene durante las pausas o no.
- El retardo entre el principio de una pausa del PCD y el reloj deteniéndose.
- El retardo entre el final de una pausa del PCD y el reloj activándose de nuevo.
- El retardo entre el principio de una pausa del PCD y el AFE detectándola.
- El retardo entre el final de una pausa del PCD y el AFE detectándola.

Para asegurar que un amplio rango de las condiciones son verificadas estos tiempos son elegidos aleatoriamente con restricciones sobre cuántos flancos el reloj de la PICC pueden faltar comparado al reloj del PCD. El código que elige esos valores toma un argumento *missing_edges* que define el número de flancos que faltan al reloj. Este argumento es usado con la sintaxis de SystemVerilog "std::randomize() with {}" que permite especificar una lista de variables a randomizar y una lista de condiciones que los resultados deben cumplir. Antes fue especificado que el DUT funciona si el reloj se detiene entre 0 ciclos y 59 ciclos. Esos valores corresponden con un *missing_edges* entre 0 y 118. Las pruebas son repetidos cien veces para cada uno de *missing_edges* igual a 0 a 3 y 115 a 118, y después mil veces con *missing_edges* elegido aleatoriamente entre 4 y 114.

El informe de cobertura da un resultado de 95 % por el DUT. Los tres sitios que faltan cobertura son condiciones imposibles de ocurrir, por ejemplo emitiendo una secuencia Z después de una secuencia X. La norma define una X → Z como un error, por lo tanto cuando el DUT detecta la Z emite un error mediante *out_iface.error*, en vez de tratarla como una secuencia Z normal.

iso14443_2a

Este módulo simplemente instancia los módulos: **tx** y **sequence_decode** como se muestra la Figura 3.12. Las entradas y salidas son descritas en el Cuadro 3.6.

Nombre	Dirección	Descripción
clk	Entrada	Reloj.
rst_n	Entrada	Reset activo bajo.
pause_n_synchronised	Entrada	Pausa, activa baja, sincronizada.
rx_iface	Salida	Fuente de la *rx_interface*.
tx_iface	Entrada	Sumidero de la *tx_interface*.
lm_out	Salida	Señal hacia el modulador de carga.

Cuadro 3.6: Entradas y Salidas del módulo **iso14443_2a**.

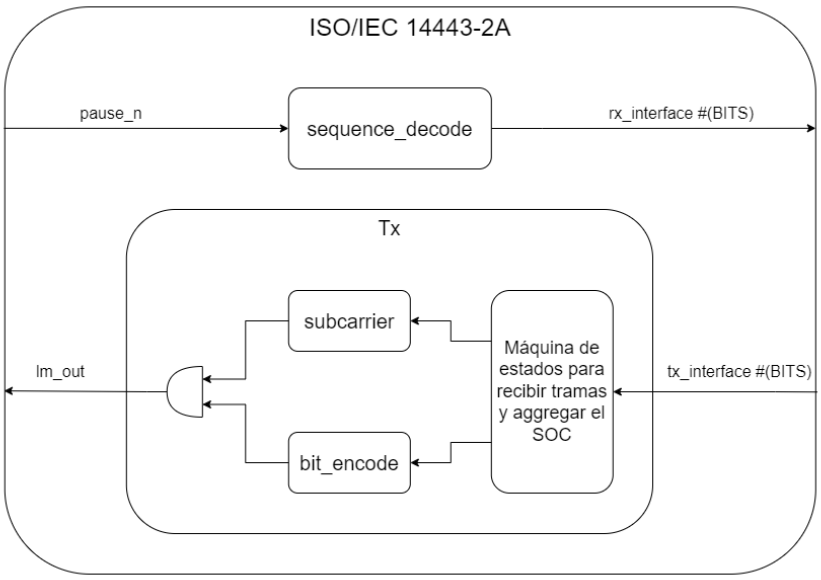

Figura 3.12: Diagrama en bloque del módulo **iso14443_2a**.

El banco de prueba funciona de forma loopback, genera mil transacciones aleatoriamente, que son transmitidos mediante el controlador **PCDPauseNDriver** en el modelo del AFE. El monitor **RxBitIfaceMonitor** recibe las tramas procesadas por el DUT sobre la *rx_iface*. Cada transacción recibida por el monitor es convertida a una **TxBitTransaction** y es reenviado sobre la *tx_iface* con un controlador **TxBitIfaceSourceDriver**. Finalmente un monitor **LoadModulatorMonitor** detecta las tramas transmitidas por el DUT sobre su salida *lm_out*. Cada transacción recibida por ese monitor es comparada

con las transacciones enviadas en una aserción. La Figura 3.13 muestra un diagrama en bloques del banco de prueba.

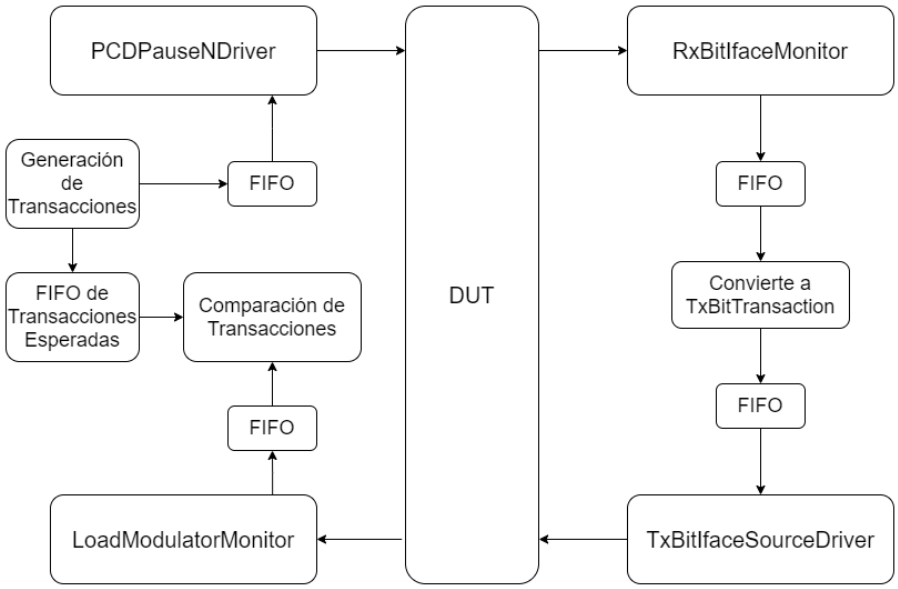

Figura 3.13: Diagrama en bloques del banco de prueba para el módulo **iso14443_2a**

Debido a que este módulo solo consiste en la instanciación de dos módulos, la única métrica de cobertura generada por el DUT, no incluyendo los submódulos auxiliares, es de cambio de estado de las señales y puertos (toggle), que tiene 100 % cobertura.

ISO/IEC 14443-3A

Esta parte de la implementación consiste en once módulos y sus bancos de pruebas correspondientes.

- **frame_decode**: Verifica y quita los bits de paridad de tramas recibidas desde el PCD.
- **desreialiser**: Convierte una trama de bits a una trama de bytes. Usado en la ruta de recepción.
- **FDT**: Genera una señal para activar la transmisión de una respuesta, para que se cumpla con el FDT definido en la norma.
- **CRC_A**: Calcula el CRC16 de una serie de bits.

- **crc_control**: Verifica el CRC16 de tramas recibidas, y genera el CRC16 de tramas a transmitir.
- **serialiser**: Convierte una trama de bytes a una trama de bits. Usado en la ruta de transmisión.
- **frame_encode**: Comienza la transmisión de una trama al PCD cuándo recibe la señal de activación del módulo FDT, y agrega los bits de paridad y el CRC16 a la trama.
- **framing**: Instancia los módulos de arriba.
- **initialisation**: Recibe tramas de inicialización y anticolisión desde el PCD, actúa sobre ellas, y responde con las respuestas adecuadas.
- **routing**: Rutea tramas recibidas desde el PCD al módulo apto para procesarlas, y rutea las respuestas desde ese módulo para transmitir al PCD. El ruteo es controlado basado en el estado actual de la PICC.
- **iso14443_3a**: El módulo más alto para esta parte de la norma, instancia los módulos: **framing**, **initialisation** y **routing**.

frame_decode

Este módulo tiene un sumidero de la *rx_interface* que está conectado a la fuente del módulo **iso14443_2a**, y una fuente, también de la *rx_interface* que está conectado al próximo módulo. Las Tramas son recibidas en una serie de bits, después de cada 8 bits hay un bit de paridad. Esos bits de paridad son verificados y quitados antes de reenviar la trama por la fuente. Las entradas y salidas son descritas en el Cuadro 3.7.

Nombre	Dirección	Descripción
clk	Entrada	Reloj.
rst_n	Entrada	Reset activo bajo.
in_iface	Entrada	Sumidero de la *rx_interface*.
out_iface	Salida	Fuente de la *rx_interface*.
last_bit	Salida	Último bit recibido.

Cuadro 3.7: Entradas y Salidas del módulo **frame_decode**.

El tipo de paridad definido por la norma es impar, eso es a decir que el número de unos en el byte más el bit de paridad debería ser impar. Por ejemplo el byte: 8'b0101_1100, tiene cuatro '1's, cuatro es par, así el bit de paridad debe ser '1', para que el total de los '1's sea impar.

Los bits de paridad son verificados y quitados mediante el siguiente proceso. Al principio

de cada trama, un indicador *expected_parity* es inicializado a '1'. Después cuándo cada uno de los primeros ochos bits son recibidos, son reenviados, y ese indicador es actualizado con el resultado de: *expecetd_parity* XOR *in_iface.data*. Ese cálculo invierte el valor de *expected_parity* cada vez que un '1' es recibido. El noveno bit es el bit de paridad, así es comparado con *expected_parity* para verificar que es correcto. Este proceso se repite hasta el fin de la trama, o hasta un error es detectado.

Hay tres errores que pueden ser detectados en este módulo: Un error indicado en la *in_iface*, un error en un bit de paridad, y una trama que le falta su último bit de paridad. Para soportar tramas cortas y de anticolisiones que pueden terminar con un byte parcial sin un bit de paridad, ese último error solo es emitido si una trama termina con un byte entero.

Finalmente la salida *last_bit* indica el valor del último bit recibido, incluyendo bits de paridad. Esta señal es usada en el módulo **FDT** para determinar cuándo comenzar a enviar la respuesta, debido a que el FDT usado depende en el valor de ese último bit.

El banco de prueba para este módulo consiste en un controlador **RxBitIfaceDriver** y un monitor **RxBitIfaceMonitor**. Por cada prueba una transacción es generada aleatoriamente, los bits de paridad son agregados, la transación es enviada por el controlador, el resultado es recibido por el monitor y es comparado con la transacción enviada. Las pruebas son:

- Enviar una transacción de 8 bits más el bit de paridad correcto.
- Enviar una transacción de 8 bits más el bit de paridad equivocado.
- Enviar una transacción de 8 bits que falta el bit de paridad.
- Enviar mil transacciones de 8 bits cada uno, con un error enviado por el controlador.
- Enviar ocho transacciones con cero a siete bits en orden.
- Envíar mil transacciones cada uno con un número de bits aleatorio entre uno y ochenta, más los bits de paridad correctos.
- Envíar mil transacciones cada uno con un número de bits aleatorio entre ocho y ochenta, con un bit de paridad equivocado y los demás correctos.
- Enviar mil transacciones con un número de bytes entero entre 1 y diez, más los bits de paridad por cada byte menos el último.

Hay dos aserciones concurrentes para verificar que la señal *last_bit* es correcta. La primera confirma que la señal corresponde con el último bit enviado. La segunda verifica que esa señal no cambia entre tramas. El informe de cobertura da el DUT un resultado de 100 %.

deserialiser

Este módulo recibe tramas de bits por un sumidero de la *rx_interface* desde el módulo **frame_decode**. Esas tramas son convertidas a tramas de bytes y reenviadas por una fuente de la *rx_interface*. La norma define que el bit menos significativo en cada byte es transmitido primero [12]. Las entradas y salidas son descritas en el Cuadro 3.8.

Nombre	**Dirección**	**Descripción**
clk	Entrada	Reloj.
rst_n	Entrada	Reset activo bajo.
in_iface	Entrada	Sumidero de la *rx_interface*.
out_iface	Salida	Fuente de la *rx_interface*.

Cuadro 3.8: Entradas y Salidas del módulo **deserialiser**.

El banco de prueba para este módulo consiste en un controlador **RxBitIfaceDriver** y un monitor **RxByteIfaceMonitor**. Por cada prueba una transacción es generada aleatoriamente, es enviada por el controlador, el resultado es recibido por el monitor y es comparado con la transacción enviada. Las pruebas son:

- Enviar 17 transacciones con número de bits entre cero y 16 en orden.
- Enviar mil transacciones con número de bits elegido aleatoriamente entre cero y ochenta.
- Enviar mil transacciones con errores insertados por el controlador.

El informe de cobertura da un resultado de 100 % por el DUT.

FDT

La norma define el FDT (Frame Delay Time) como el tiempo entre la última pausa transmitida por el PCD y el primer flanco de modulación de la respuesta enviada por la PICC [12]. El valor del FDT requerido depende en el último bit enviado por el PCD.

Último Bit	**FDT (en ciclos del reloj)**
'0'	$128 \cdot n + 20$
'1'	$128 \cdot n + 84$

El valor de 'n' depende del tipo de trama enviada. Las tramas de inicialización: REQA, WUPA, ANTICOLLISION y SELECT requieren un $n = 9$, dando FDTs de 1172 ciclos y 1236 ciclos. Cualquier otra trama requiere un $n \geq 9$. Sin embargo para simplicidad esta implementación siempre usa un $n = 9$. La norma requiere que una PICC envíe su

respuesta entre este tiempo y este tiempo más 0,4 µs, lo que da aproximadamente 5,4 ciclos de flexibilidad.

Este mecanismo es usado para que si más de una PICC responde a la misma solicitud, las respuestas estén sincronizadas, y si responden con respuestas diferentes el PCD puede detectar una colisión y en qué bit ocurre.

El FDT puede ser dividido en tres partes:

1. El tiempo entre el instante cuando el PCD termina la pausa y cuando este módulo detecta el flanco ascendente de la pausa. Este incluye el retardo en el detector de pausas en el AFE y el número de ciclos necesarios para que la señal se propague por el diseño digital hasta llegar a este módulo.
2. La demora entre el instante en cual el flanco ascendente de la pausa es detectado en este módulo y la activación de la salida *trigger* que dispara la transmisión de la respuesta.
3. El tiempo entre la activación de la salida *trigger* y el primer flanco de la respuesta llegando al modulador de carga.

Si los puntos 1, y 3 son conocidos, es posible elegir un valor por el punto 2, para obtener el FDT total deseado. Este núcleo IP está diseñado para ser genérico así que puede ser usado en varios sistemas con propiedades diferentes. Con este fin, este módulo es parametrizado con un parámetro *TIMING_ADJUST*, lo que especifica el retardo externo de este módulo (punto 1 + punto 3), en números de ciclos. Debido a que está permitido comenzar la respuesta hasta 0,4 µs tarde pero no está permitido comenzar antes, es importante elegir el valor mínimo para *TIMING_ADJUST* del rango de valores posibles. También la conversión entre un tiempo y el número de ciclos debería usar la frecuencia de la portadora mínima: $f_{c\_min} = 13{,}56\,\text{MHz} - 7\,\text{kHz}$, para usar el período máximo, y el número de ciclos debería ser redondeado por abajo.

La Figura 3.14 muestra la ruta del FDT en este trabajo. El parámetro *TIMING_ADJUST* es calculado de forma jerárquica. Este módulo es instanciado en el módulo **framing**, que es instanciado en el módulo **iso14443_3a**, que es en turno instanciado en el módulo más alto de este núcleo IP: **iso14443a**. Cada uno de estos módulos lleva un parámetro *FDT_TIMING_ADJUST*, el cual es pasado a la próxima módulo en la jerarquía después de sumar el retardo introducido en ese módulo. Por ejemplo el módulo **framing** instancia el módulo **frame_encode** que tiene un ciclo de retardo entre el flanco ascendente de la señal *trigger* y el principio de la respuesta. Por lo tanto el módulo **framing** suma uno a su parámetro *FDT_TIMING_ADJUST* y pasa ese valor al parámetro del módulo **FDT**.

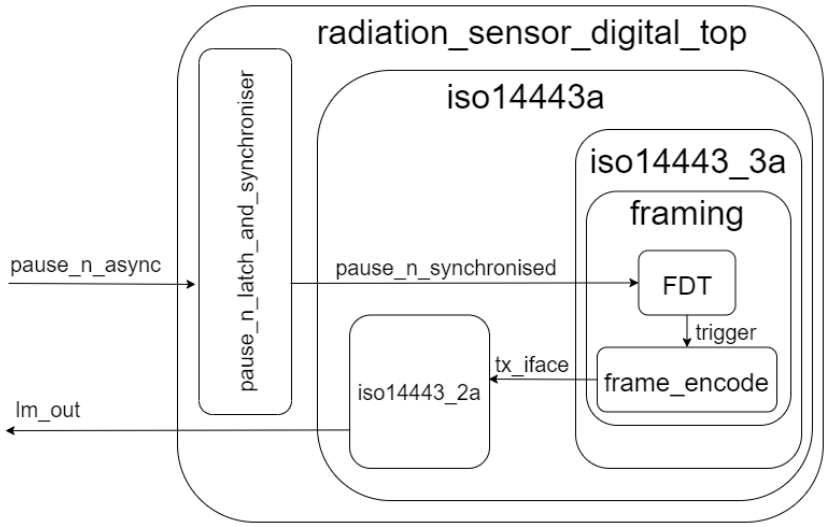

Figura 3.14: La ruta del FDT en este trabajo.

Las entradas y salidas son descritas en el Cuadro 3.9.

Nombre	Dirección	Descripción
clk	Entrada	Reloj.
rst_n	Entrada	Reset activo bajo.
pause_n_synchronised	Entrada	Pausa sincronizada.
last_rx_bit	Entrada	Último bit transmitido por el PCD.
trigger	Salida	Señal de activación para comenzar la transmisión de la respuesta.

Cuadro 3.9: Entradas y Salidas del módulo **FDT**.

La implementación de este módulo consiste en un detector de flancos ascendentes y un contador. El contador es reseteado a cero cada vez que un flanco ascendente es detectado en la entrada *pause_n_synchronised*. Cuándo el contador llega al valor adecuado, calculado con la entrada *last_rx_bit* y el parámetro *TIMING_ADJUST*, la salida *trigger* es pulsada para un ciclo del reloj, y el contador es desactivado hasta que la próxima pausa es detectada.

Para verificar este módulo el banco de prueba instancia el DUT con el parámetro *TIMING_ADJUST* = 6. Este valor es arbitrario, ya que en este banco de prueba no es necesario compensar por retrasos externos. El valor 6 fue elegido porque es el valor usado en la síntesis de este trabajo. Después hay un bloque de código que mide el tiempo entre

el último flanco ascendente de la entrada *pause_n_synchronised* y la activación de la salida *trigger*, verificando que es precisamente el número de picosegundos esperados. Las pruebas son:

- Verifica que si no hay pausas la señal *trigger* no se activa.
- Verifica el FDT cuando la entrada *last_bit* es '0', mediante una sola pausa.
- Verifica que si no hay más pausas la señal *trigger* no se activa una segunda vez.
- Verifica el FDT cuando la entrada *last_bit* es '1', mediante una sóla pausa.
- Envía un número aleatorio de pausas con un tiempo aleatorio entre cada una, y verifica que la señal *trigger* sólo se activa después de la última pausa, y que el FDT es correcto.

El informe de cobertura da un resultado de 98 % por el DUT. La única parte que falta cobertura es una condición imposible de ocurrir.

CRC_A

Este módulo calcula el CRC16 definido por el polinomio $x^{16}+x^{12}+x^5+1$ y el valor inicial 0x6363, por una serie de bits. Esta implementación está diseñada para ser utilizada por tramas de recepción y de transmisión, por lo tanto no usa una interfaz. Las salidas y entradas son descritas en el Cuadro 3.10.

La implementación usa un Registro de Desplazamiento con Realimentación Lineal (LFSR, por sus siglas en Inglés) y es basada en el código generado mediante un generador de LFSRs por CRCs [18]. Un LFSR es un registro de desplazamiento pero unos de los bits son realimentados con el XOR de otros bits y la entrada. Para calcular el CRC16 con el polinomio definido en la norma, los bits con realimentación son los de índices: 0, 5 y 12. La Figura 3.15 muestra el LFSR. Las operaciones para calcular los bits con realimentación son:

$$\begin{aligned} \text{lfsr}[12] &= \text{lfsr}[15] \text{ XOR } \text{lfsr}[11] \text{ XOR data} \\ \text{lfsr}[5] &= \text{lfsr}[15] \text{ XOR } \text{lfsr}[4] \text{ XOR data} \\ \text{lfsr}[0] &= \text{lfsr}[15] \text{ XOR data} \end{aligned}$$

Cuándo la entrada *start* es '1', el registro *lfsr* es asignado el valor 0x6363. Cuándo la entrada *sample* es '1' el valor del registro es actualizado con el algoritmo descrito arriba. La salida *crc* es asignado el valor del registro *lfsr*.

Nombre	Dirección	Descripción
clk	Entrada	Reloj.
rst_n	Entrada	Reset activo bajo.
start	Entrada	Comenzar un nuevo cálculo.
data	Entrada	Próximo bit.
sample	Entrada	El dato es válida.
crc[15:0]	Salida	Resultado.

Cuadro 3.10: Entradas y Salidas del módulo **CRC_A**.

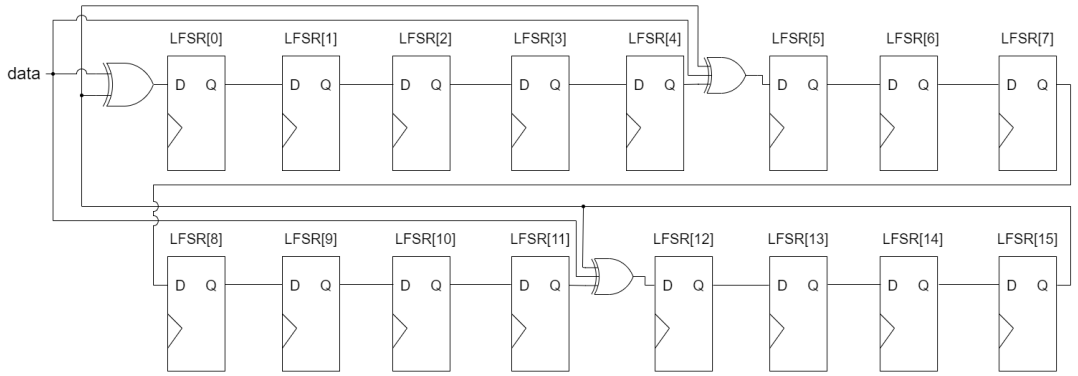

Figura 3.15: El LFSR del módulo **CRC_A**.

El banco de prueba para este módulo calcula el CRC de varias series de bytes, mediante dos métodos: con otra implementación del LFSR, y mediante un algoritmo que opera sobre bytes en vez de bits. La implementación de este algoritmo de bytes es provisto en el apéndice B de la parte tres de la norma [12]. Estos dos resultados son comparados con la salida del DUT en dos aserciones. Además el apéndice B de la norma específica los CRCs por dos series de bits. Estas series de bits son verificadas como descrita arriba, y también los resultados son comparados con los especificados en la norma. Las series de bytes probados son:

- Sin data: El CRC es el valor inicial 0x6363.
- {0x00, 0x00}: Especificado en la norma con CRC 0x1EA0.
- {0x12, 0x34}: Especificado en la norma con CRC 0xCF26.
- Mil transacciones aleatorias entre 0 y 10 bytes enteros.

El informe de cobertura da un resultado de 100 % por este banco de prueba.

crc_control

Este módulo instancia el módulo **CRC_A** y lo usa para verificar los CRCs de tramas recibidas y para calcular los CRCs de tramas a transmitir. Las entradas y salidas son descritas en el Cuadro 3.11. Este módulo contiene un monitor por la *rx_interface* y uno por la *tx_interface*. En este caso el término "monitor" se refiere a un modport en las interfaces dónde todas las señales son entradas. No se debe confundir con los monitores usados en los bancos de pruebas.

Nombre	Dirección	Descripción
clk	Entrada	Reloj.
rst_n	Entrada	Reset activo bajo.
rx_iface	Entrada	Monitor por la *rx_interface*.
rx_crc_ok	Salida	Indica si el CRC de la última trama tiene un CRC válido.
tx_iface	Entrada	Monitor por la *tx_interface*.
tx_append_crc	Entrada	Calcular el CRC para esta trama de transmisión
fdt_trigger	Entrada	La señal de activación de la transmisión de una trama, desde el módulo FDT.
crc[15:0]	Salida	El resultado

Cuadro 3.11: Entradas y Salidas del módulo **crc_control**.

Una propiedad de este algoritmo de CRC es: si se agrega el CRC de una serie de bits al final de esa serie, el CRC de la nueva serie de bits es cero. Por lo tanto para verificar la integridad de una trama recibida, es suficiente comprobar que el CRC calculado en este módulo es cero.

La entrada *start* del módulo **CRC_A** es pulsado al detectar el principio de una trama en una de las interfaces. Esto causa que el módulo **CRC_A** reinicie su LFSR. En el caso de tramas recibidas desde el PCD el principio de una trama es indicado con la señal *rx_iface.soc*. En el caso de tramas a transmitir el principio de una trama es indicado cuando las entradas *fdt_trigger*, y *tx_iface.data_valid* están en '1's, también en este caso solo se calcula el CRC cuándo la entrada *tx_append_crc* está en '1'. La entrada *sample* del módulo **CRC_A** es pulsado por cada bit de la trama, lo que ocurre cuándo una de las señales *rx_iface.data_valid* o *tx_iface.req* están en '1'. Este módulo no es responsable de agregar CRCs a las respuestas, eso ocurre en el módulo **frame_encode**. La salida *rx_crc_ok* es '1' cuando el resultado del módulo **CRC_A** es cero.

El banco de prueba por este módulo consiste en tres controladores: **RxBitIfaceDriver**, **TxBitIfaceSourceDriver** y **TxBitIfaceSinkDriver**. El controlador **TxBitIfaceSinkDriver** es necesario para manejar la señal *tx_iface.req* porque el DUT no contiene un sumidero. Varias tramas son enviadas sobre ambos interfaces, y las salidas del DUT *crc* y *rx_crc_ok* son verificadas mediante aserciones:

- Verifica que la salida *rx_crc_ok* es como esperado al final de una trama de recepción.
- Verifica que la salida *rx_crc_ok* no cambia después del fin de una trama.
- Verifica que la salida *crc* es igual al CRC esperado al final de una trama de transmisión.
- Verifica que la salida *crc* no cambia después del fin de una trama.
- Verifica que la salida *crc* no cambia mientras un indicador *check_crc_stable* está '1'.
- Verifica que *rx_crc_ok* es '1' cuando y sólo cuando la salida *crc* es cero.

Las pruebas ejecutados son:

- Enviar tres tramas sobre la *rx_iface*: {0x00, 0x00, 0xA0, 0x1E}, {0x12, 0x34, 0x26, 0xCF} y {0x63, 0x63}. Estas tramas son las series de bits con sus CRCs especificados en el apéndice B de la norma [12].
- Enviar mil tramas sobre la *rx_iface* con sus CRC agregados.
- Enviar mil tramas sobre la *rx_iface* con sus CRC agregados y un bit de la trama corrompido.
- Enviar dos tramas sobre la *tx_iface*: {0x00, 0x00}, y {0x12, 0x34}, para verificar la salida *crc* es 0x1EA0 y 0xCF26 respectivamente, como especificado en el apéndice B de la norma.
- Enviar mil tramas sobre la *tx_iface* y verificar que la salida *crc* es como se espera en cada caso.

El informe de cobertura da un resultado de 100 % por el DUT.

serialiser

Este módulo es parecido al módulo **deserialiser** pero al revés, convierte tramas de bytes a tramas de bits para transmisión. Las salidas y entradas son descritas en el Cuadro 3.12.

Este módulo espera por el principio de una trama, lo que pasa cuándo la entrada *in_iface.data_valid* está en '1' y después fija la salida *out_iface.data_valid* en '1' y la salida *out_iface.data* en el valor del primer bit de la entrada *in_iface.data*. Cuando el módulo conectado a la fuente *out_iface* pide el siguiente bit mediante la señal *out_iface.req*,

la salida *out_iface.data* se actualiza. Después de haber enviado el último bit del byte, la salida *out_iface.last_bit_in_byte* pasa a '1', y otro byte es pedido mediante la salida *in_iface.req*. Ese proceso continúa hasta el final de la trama. Debido a que las respuestas pueden comenzar con un byte parcial en el caso de tramas de anticolisión, la entrada *in_iface.data_bits* se usa para determinar el número de bits en el primer byte.

Nombre	Dirección	Descripción
clk	Entrada	Reloj.
rst_n	Entrada	Reset activo bajo.
in_iface	Entrada	Sumidero para la *tx_interface*.
out_iface	Salida	Fuente para la *tx_interface*.

Cuadro 3.12: Entradas y Salidas del módulo **serialiser**.

El banco de prueba para este módulo consiste en dos controladores y un monitor: un **TxByteIfaceSourceDriver** por la entrada *in_iface*, un **TxBitIfaceSinkDriver** por la salida *out_iface*, y un **TxBitIfaceMonitor** por la salida *out_iface*. Para verificar el comportamiento del DUT, transacciones de bytes son generadas y enviadas. Después las transacciones enviadas son convertidas a transacciones de bits, y comparadas con las recibidas por el monitor. Hay dos pruebas: Enviar 16 transacciones de tamaños 1 a 16 bits en orden, y envíar mil transacciones de tamaños aleatorios.

El informe de cobertura da un resultado de 100 % por el DUT.

frame_encode

Este módulo es responsable de agregar el CRC16 y los bits de paridad a las respuestas. Además solo comienza envíar la trama cuando está activado mediante una entrada *fdt_trigger* que viene del módulo **FDT**. Las salidas y entradas son descritas en el Cuadro 3.13.

Nombre	Dirección	Descripción
clk	Entrada	Reloj.
rst_n	Entrada	Reset activo bajo.
fdt_trigger	Entrada	Señal para activar el envío de la respuesta.
in_iface	Entrada	Sumidero de la *tx_interface*.
append_crc	Entrada	Indica que la respuesta tiene CRC.
crc[15:0]	Entrada	CRC.
out_iface	Salida	Fuente de la *tx_interface*.

Cuadro 3.13: Entradas y Salidas del módulo **frame_encode**.

La implementación espera la señal de activación desde el módulo **FDT**. En ese momento si hay una trama lista para enviar, indicado con la entrada *in_iface.data_valid*, comienza reenviar la trama sobre la *out_iface*. El bit de paridad por cada byte es calculado en la misma manera del módulo **frame_decode**. Cuándo la entrada *in_iface.last_bit_in_byte* está en '1', indicando el último bit en el byte actual, el bit de paridad calculado es enviado. Después del fin de la trama, si la entrada *append_crc* está en '1', el CRC y su bits de paridad son enviados.

El banco de prueba consiste en: el controlador de fuente **TxBitIfaceSourceDriver** por la *in_iface*, el controlador de sumidero **TxBitIfaceSinkDriver** y el monitor **TxBitIfaceMonitor** por la *out_iface*. Para verificar el comportamiento del DUT, transacciones son generadas y enviadas. Por cada prueba, si la trama tiene un CRC, está agregado a la transacción. Los bits de paridad siempre son agregados. Esta nueva transacción es comparada con la trama detectada por el monitor. Debido a que el DUT solo comienza enviar una trama cuándo su entrada *fdt_trigger* está '1', esa señal es pulsada un número de ciclos aleatorios después de que la transacción fue enviada al controlador. Las pruebas son:

- Envía cien tramas, pero no pulsa la señal *fdt_trigger*, y verifica que el monitor no recibe ninguna transacción.
- Pulsa *fdt_trigger* sin envíar una transacción y verifica que el monitor no recibe ninguna transacción.
- Envía tramas de tamaño entre uno y ocho bits, sin CRCs.
- Envía mil tramas de tamaño aleatorio, sin CRC.
- Envía mil tramas cada una con un número de bytes entero, con CRC.

El informe de cobertura por el DUT da un resultado de 98 %. La única parte que falta cobertura es una condición imposible de ocurrir.

framing

Este módulo simplemente instancia los siguientes módulos: **frame_decode**, **deserialiser**, **FDT**, **crc_control**, **deserialiser** y **frame_encode**, como se muestra la Figura 3.16. Las entradas y salidas están descritas en el Cuadro 3.14.

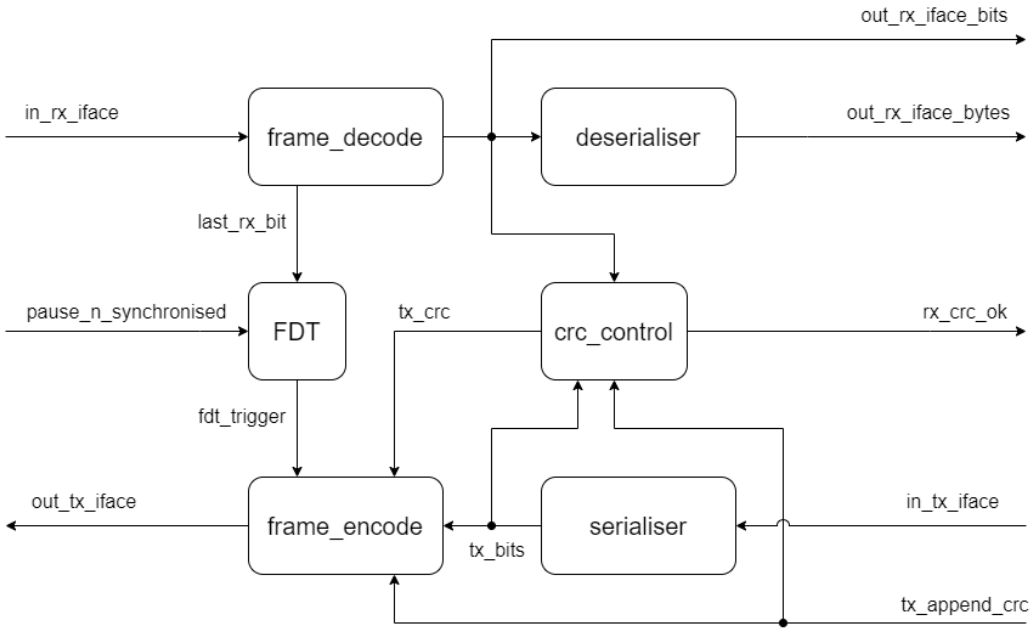

Figura 3.16: Diagrama en bloques del módulo **framing**.

Nombre	**Dirección**	**Descripción**
clk	Entrada	Reloj.
rst_n	Entrada	Reset activo bajo.
pause_n_synchronised	Entrada	Pausa, activa bajo, sincronizada.
in_rx_iface	Entrada	Sumidero de bits por la *rx_interface*.
out_rx_iface_bytes	Salida	Fuente de bytes por la *rx_interface*.
out_rx_iface_bits	Salida	Fuente de bits por la *rx_interface*.
rx_crc_ok	Salida	Indica que la última trama recibida tiene CRC válido.
in_tx_iface	Entrada	Sumidero de bytes por la *tx_interface*.
tx_append_crc	Entrada	Indica que la respuesta debe tener CRC.
out_tx_iface	Salida	Fuente de bits por la *tx_interface*.

Cuadro 3.14: Entradas y Salidas del módulo **framing**.

Para la ruta de recepción el flujo de datos es:

- Los bits de paridad están verificados y quitados.
- El CRC es verificado.
- La trama de bits es convertida a una trama de bytes.
- El temporizador del FDT comienza a contar después del último flanco ascendente de la pausa.

Hay dos salidas por las tramas recibidas desde el PCD, *out_rx_iface_bytes* y *out_rx_iface_bits*. La fuente de bits es para facilitar la comparación del UID en tramas de anticolisión en el módulo **initialisation**. Este tema es descrito más en la Sección initialisation.

Para la ruta de transmisión el flujo de datos es:

- La transmisión comienza cuando el temporizador del FDT vence.
- Las tramas de bytes son convertidas a tramas de bits.
- Los bits de paridad son insertados.
- El CRC es calculado y agregado, incluyendo sus bits de paridad, si es pedido con la entrada *tx_append_crc*.

Este módulo tiene un parámetro *FDT_TIMING_ADJUST* que especifica el retardo en la ruta del FDT externo a este módulo. El módulo **frame_decode** introduce un ciclo de retardo en la transmisión de la respuesta, por lo tanto el valor pasado al parámetro *TIMING_ADJUST* del módulo **FDT** es *FDT_TIMING_ADJUST* + 1.

El banco de prueba para este módulo consiste en:

- Un controlador **RxBitIfaceDriver** por la *in_rx_iface*.
- Un monitor **RxBitIfaceMonitor** por la *out_rx_iface_bits*.
- Un monitor **RxByteIfaceMonitor** por la *out_rx_iface_bytes*.
- Un controlador de fuente **TxByteIfaceSourceDriver** por la *in_tx_iface*.
- Un controlador de sumidero **TxBitIfaceSinkDriver** por la *out_tx_iface*.
- Un monitor **TxBitIfaceMonitor** por la *out_tx_iface*.

Para simular la recepción de una trama desde el PCD hay una tarea **send_rx_frame()** que construye una transacción aleatoriamente con el número de bits especificado en un argumento. Si tiene un número de bytes entero, el CRC es calculado y agregado. Después los bits de paridad son insertados. Otro argumento especifica si la trama debería contener un error, y que tipo de error. Los tipos de errores son: bit de paridad equivocado, falta el último bit de paridad, CRC equivocado, y un error en la trama enviada mediante la

señal *in_rx_iface.error*. Las transacciones recibidas por los dos monitores son comparadas con las esperadas. Hay una aserción concurrente que verifica que la salida *rx_crc_ok* es como esperado por cada trama.

La ruta de transmisión es parecida. La tarea **send_tx_frame()** arma una transacción de bytes con el número de bits especificado en un argumento. La transacción recibida por el monitor es comparada con la enviada. El DUT sólo comienza transmitir una trama cuando el temporizador del módulo **FDT** vence. Para comenzar este temporizador se requiere al menos un flanco ascendente en la entrada *pause_n_synchronised*. Por lo tanto esta tarea pulsa esa señal, después de haber pasado una transacción al controlador.

Como en el banco de prueba para el módulo **FDT**, el tiempo entre el último flanco ascendente de la pausa y el principio de la transmisión de la respuesta es medido y verificado.

Las pruebas son listadas acá, cada uno es repetida mil veces:

- Envía una trama sobre la *in_rx_iface* de número de bytes entero, con CRC.
- Envía una trama sobre la *in_rx_iface* terminando con un byte parcial, sin CRC.
- Envía una trama sobre la *in_rx_iface* de número de bytes entero, con CRC equivocado.
- Envía una trama sobre la *in_rx_iface* con un bit de paridad equivocado.
- Envía una trama sobre la *in_rx_iface* que falta su último bit de paridad.
- Envía una trama sobre la *in_rx_iface* con un error mediante la señal *in_rx_iface.error*.
- Envía una trama sobre la *in_tx_iface* sin CRC.
- Envía una trama sobre la *in_tx_iface* de número de bytes enteros con CRC.
- Envía una trama sobre la *in_rx_iface* y después una trama sobre la *in_tx_iface* como una respuesta.

El informe de cobertura por el DUT da un resultado de 100 %.

routing

La norma ISO/IEC 14443 divide las definiciones de las tramas entre las partes tres y cuatro de la norma, y el diseño de este núcleo IP siguiendo esta división, procesa esas tramas en módulos distintos. La norma específica cuáles tramas son aceptadas en cada estado de la PICC, la Figura 1.8 muestra esto. En los estados: IDLE, READY, HALT y READY*, la PICC solo soporta las tramas de inicialización y anticolisión. En el estado PROTOCOL, la PICC solo soporta las tramas definidas en la parte cuatro de la norma.

En los estados: ACTIVE y ACTIVE*, la PICC puede recibir la trama de inicialización HLTA o la trama RATS de la parte cuatro de la norma. Este módulo es diseñado para dirigir tramas recibidas desde el PCD a los módulos que correspondan: **initialisation** y **iso14443_4a** dependiendo en el estado actual de la PICC, y además dirige las respuestas generadas desde cada módulo. Las entradas y salidas están descritas en el Cuadro 3.15.

Nombre	**Dirección**	**Descripción**
route_rx_to_initialisation	Entrada	Dirigir tramas recibidas al módulo **initialisation**.
route_rx_to_14443_4	Entrada	Dirigir tramas recibidas al módulo **iso14443_4a**.
route_tx_from_14443_4	Entrada	Aceptar respuestas desde el módulo **iso14443_4a** (valor '1'), o desde el módulo **initialisation** (valor ''0').
in_rx_iface	Entrada	Sumidero por la *rx_interface*.
out_rx_iface_init	Salida	Fuente por la *rx_interface*, conectado al módulo **initialisation**.
out_rx_iface_14443_4	Salida	Fuente por la *rx_interface*, conectado al módulo **iso14443_4a**.
in_tx_iface_init	Entrada	Sumidero por la *tx_interface*, desde el módulo **initialisation**.
in_tx_append_crc_init	Entrada	La respuesta desde el módulo **initialisation** debe tener CRC agregado.
in_tx_iface_14443_4	Entrada	Sumidero por la *tx_interface*, desde el módulo **iso14443_4a**.
in_tx_append_crc_14443_4	Entrada	La respuesta desde el módulo **iso14443_4a** debe tener CRC agregado.
out_tx_iface	Salida	Fuente por la *tx_interface*.
out_tx_append_crc	Salida	La respuesta enviada debe tener CRC agregado.

Cuadro 3.15: Entradas y Salidas del módulo **routing**.

La implementación es un circuito combinatorio para evitar el uso de registros y reducir el área del diseño. La ruta de recepción consiste en una entrada: *in_rx_iface*, dos salidas: *out_rx_iface_init* y *out_rx_iface_14443_4*, y dos señales de control: *route_rx_to_initialisation* y *route_rx_to_14443_4*. Las señales *in_rx_iface.data* y *in_rx_iface.data_bits* simplemente son conectadas a las señales correspondientes de las dos salidas. Las demás de las señales: *error*, *soc*, *eoc*, y *data_valid* pasan por compuertas ANDs con la señal de control correspondiente, como se muestra la Figura 3.17. De esta forma, si una salida no está activada los indicadores en la interfaz de esa salida serían '0'.

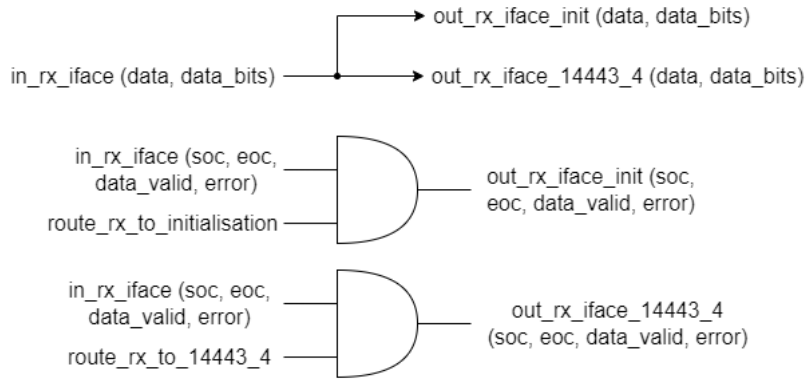

Figura 3.17: La ruta de recepción para las señales en el módulo **routing**.

La ruta de transmisión tiene dos entradas: *in_tx_iface_init* y *in_tx_iface_14443_4*, una salida: *out_tx_iface*, y una señal de control: *route_tx_from_14443_4*. Cuando *route_tx_from_14443_4* está '0', respuestas son dirigidas desde la entrada *in_tx_iface_init*, y cuándo está '1' las respuestas son dirigidas desde la entrada *in_tx_iface_14443_4*. Las señales *in_tx_iface_init.req* y *in_tx_iface_14443_4.req* deben ser '0' cuando su entrada está desactivada. Por lo tanto un par de compuertas AND son usadas. Las otras señales son conectadas mediante un multiplexor. La Figura 3.18 muestra está implementación.

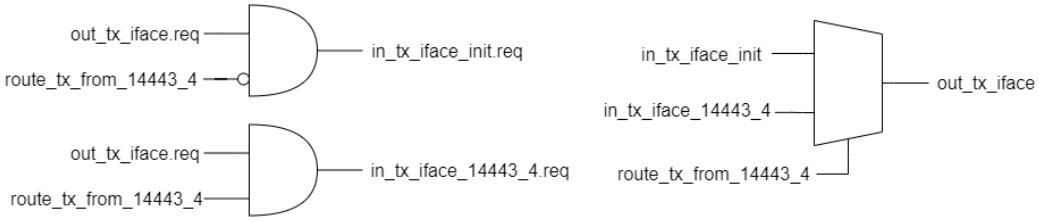

Figura 3.18: La ruta de transmisión en el módulo **routing**.

El banco de prueba por este módulo simplemente fija las señales de control en los modos definidos abajo, cambia todas las entradas aleatoriamente y verifica que todas las salidas son como esperadas. Este proceso es repetido diez mil veces por cada uno de estos modos:

- Rx solo a **initialisation**, Tx desde **initialisation**.
- Rx solo a **iso14443_4a**, Tx desde **iso14443_4a**.
- Rx a **initialisation** y **iso14443_4a**, Tx desde **initialisation**.

- Rx a **initialisation** y **iso14443_4a**, Tx desde **iso14443_4a**.

El informe de cobertura da el DUT un resultado de 100 %.

initialisation

Este módulo es el primero que actúa sobre las tramas recibidas y genera las respuestas requeridas por la norma. Las tramas soportadas son las de inicialización y anticolisión: REQA, WUPA, ANTICOLLISION, SELECT, y HLTA. Las salidas y entradas son descritas en el Cuadro 3.16. Este es un módulo parametrizado, cuyos parámetros son descritos en el Cuadro 3.17.

Nombre	**Dirección**	**Descripción**
clk	Entrada	Reloj.
rst_n	Entrada	Reset activo bajo.
uid_variable	Entrada	Bits menos significativos del UID.
rx_iface	Entrada	Sumidero de bytes por la *rx_interface*.
rx_iface_bits	Entrada	Sumidero de bits por la *rx_interface*.
rx_crc_ok	Entrada	Los últimos dos bytes de la trama recibida son un CRC válido.
tx_iface	Salida	Fuente de bytes por la *tx_interface*.
tx_append_crc	Salida	La respuesta debe tener un CRC agregado.
iso14443_4a_deselect	Entrada	El bloque ISO/IEC 14443-4A recibió el bloque estándar S(DESELECT).
iso14443_4a_rats	Entrada	El bloque ISO/IEC 14443-4A recibió el mensaje RATS.
iso14443_4a_tag_active	Salida	La PICC está en el estado ACTIVE.
route_rx_to_initialisation	Salida	Tramas de recepción deberían estar dirigidas a este módulo.
route_rx_to_14443_4a	Salida	Tramas de recepción deberían estar dirigidas al módulo **iso14443_4a**.
route_tx_from_14443_4a	Salida	El módulo **iso14443_4a** es la fuente de las respuestas (valor '1'), o este módulo es la fuente (valor '0').

Cuadro 3.16: Entradas y Salidas del módulo **initialisation**.

Cómo descrito en la Sección ISO/IEC 14443-3, durante el proceso de inicialización hay un lazo de anticolisión, dónde el UID es dividido en hasta tres partes y el PCD tiene que realizar el proceso de anticolisión y enviar una trama SELECT para cada parte del UID en turno para que la PICC transiciona al estado READY o READY*. Esto es conocido como los niveles cascadas.

Nombre	Tipo	Descripción
UID_SIZE	UIDSize (enum)	La PICC tiene un UID simple, doble o triple.
UID_INPUT_BITS	int	El ancho de la entrada *uid_variable*.
UID_FIXED	logic [] (un vector)	Bits más significativos del UID.

Cuadro 3.17: Parámetros del módulo **initialisation**.

El UID de la PICC es la concatenación del parámetro *UID_FIXED* y la entrada *uid_variable*, el número de bits total y el número de niveles cascadas dependen en el parámetro *UID_SIZE*:

UIDSize_SINGLE: 32 bits, 1 nivel cascada.
UIDSize_DOUBLE: 56 bits, 2 niveles cascadas.
UIDSize_TRIPLE: 80 bits, 3 niveles cascadas.

Un bloque "generate" es usado para partir el UID y agregar las tags cascadas al tiempo de elaboración, de la forma que muestra la Figura 1.9. Hay una señal *current_cascade_level* que indica en qué lazo de anticolisión la PICC está actualmente. Usando esa señal y un mux, una señal *current_cascade_uid_data* está asignada con los datos correspondientes para el nivel de cascada actual. Finalmente cada nivel de cascada termina con un byte BCC (Block Check Character), lo que es el XOR de los últimos cuatros bytes, este valor es calculado y incluido en la señal *current_cascade_uid_data*. Esta señal es usada con las tramas: ANTICOLLISION y SELECT para comparar el UID enviado en la trama con lo de esta PICC. En el caso de la trama ANTICOLLISION, esta señal también es usada para armar la respuesta. La Figura 3.19 muestra el circuito implementado para una PICC con UID doble.

Las tramas recibidas desde el PCD son guardadas en un buffer. La trama más larga es la SELECT lo que tiene siete bytes más dos bytes de CRC. Debido a que el CRC es verificado en el módulo **framing**, no es necesario guardar el CRC en el buffer, así el tamaño del buffer implementado es siete bytes. Si la señal *in_rx_iface_bytes.error* indica un error los demás datos de la trama son ignorados.

La implementación de este módulo incluye una máquina de estados basada en la especificación de la parte tres de la norma. Esta máquina de estados se muestra en la Figura 1.8. Debido a que las tramas: RATS, y S(DESELECT) son procesadas en el módulo **iso14443_4a**, las transiciones correspondientes son realizadas cuando ese módulo indica la recepción de esas tramas mediante las entradas: *iso14443_4a_rats* y

iso14443_4a_deselect. En el estado ACTIVE si la PICC recibe cualquier trama menos una RATS, esa trama es tratada como un error, y se transiciona al estado IDLE o HALT.

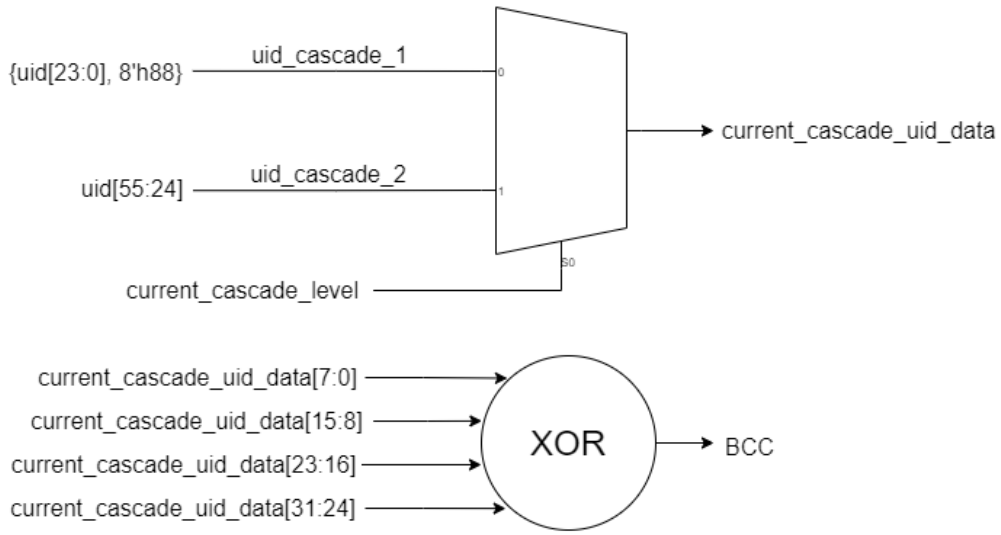

Figura 3.19: La implementación de los niveles cascadas por una PICC con UID doble en el módulo **initialisation**.

Las respuestas son construidas en un buffer de cinco bytes. Cuando la máquina de estados determina que una respuesta es necesaria, el buffer es llenado con los datos de la respuesta, y la salida *tx_iface.data_valid* se fija en '1' para indicar al sumidero que una respuesta está lista para transmitir. Después, cada vez que la señal *tx_iface.req* está pulsada, el buffer es desplazado un byte, para transmitir el próximo byte. Cuando no hay más bytes a enviar, la señal *tx_iface.data_valid* vuelve a '0' para indicar el fin de la trama. Hay tres respuestas posibles:

- ATQA: respuesta a REQA / WUPA. Tiene dos bytes que solo dependen del parámetro: *UID_SIZE*.
- SAK: respuesta a SELECT. Tiene un byte, que depende de si el nivel de cascada actual es la última.
- ANTICOLLISION: respuesta a ANTICOLLISION. Tiene entre un bit y cinco bytes del UID, que corresponden a la parte no enviada en la solicitud.

La Figura 1.7 muestra la solicitud y la respuesta por la trama ANTICOLLISION. El PCD envía parte de un UID y todas las PICCs quienes UIDs corresponden, responden con los de más de sus UIDs. La trama SELECT es igual a una ANTICOLLISION pero el PCD envía el UID entero y un CRC es agregado. Para determinar si el UID en la solicitud corresponde con el UID de esta PICC es necesario implementar un circuito digital que puede comparar entre cero y cinco bytes y en el caso de que el último byte es un byte parcial, entre 1 y 7 bits de ese byte. El circuito necesario para hacer esa comparación de forma combinatoria es un circuito grande. Una opción mejor y que es usada en esta implementación es comparar cada bit del UID en turno en el momento que es recibido. La entrada *rx_iface_bits* existe por esta razón. Después de haber recibido dos bytes de una trama mediante la entrada *rx_iface_bytes* como normal, si la data indica que la trama es una ANTICOLLISION o una SELECT, el diseño comienza comparando cada bit recibida sobre la *rx_iface_bits* con su UID. Otra ventaja de este método es que se puede construir la respuesta usando un registro de desplazamiento iniciado con el UID completo para el nivel de cascada actual, y desplazando lo con cada bit recibido.

El banco de prueba para este módulo consiste en:

- El controlador **RxBitIfaceDriver** para simular tramas desde el PCD
- El módulo **deserialiser** para convertir la trama de bits a una trama de bytes, esto permite el envío de la misma transacción sobre las dos *rx_interfaces*.
- El monitor **TxByteIfaceMonitor** para recibir las respuestas.
- El controlador de sumidero **TxByteIfaceSinkDriver** por la *tx_iface*.
- Una secuencia **InitCommsTbSequence**. Esta es una clase que extiende la secuencia **CommsTestsSequence**. Esta clase provee habilidades específicas (customisations) a este banco de prueba. Las habilidades incluyen:
  - Una tarea **do_reset()**. Esta tarea resetea el DUT a un estado conocido.
  - Antes de enviar una trama de recepción, actualiza la entrada *rx_crc_ok* del DUT para indicar si el CRC de la trama es válido.
  - Maneja las entradas *iso14443_4a_rats* y *iso14443_4a_deselect* para simular el módulo **iso14443_4a** recibiendo una RATS y una S(DESELCT).
  - Verifica que la salida *tx_append_crc* del DUT es correcta durante el envío de una respuesta.
  - Verifica el estado actual del DUT después del envío de cada trama.

El banco de prueba ejecuta todas las pruebas diez veces, con UIDs diferentes. Dentro de este lazo cada prueba es repetida cien veces. Las pruebas ejecutadas son las pruebas de

inicialización definidas en la secuencia **CommsTestSequence**, y se parten en cuatro grupos:

- Pruebas de transiciones de estados. Estas pruebas verifican la implementación de la máquina de estados mostrado en la Figura 1.8. Cada prueba consiste en los pasos:
  - Entra el estado inicial querido.
  - Envía una trama.
  - Verifica la respuesta o que no hay respuesta.
  - Verifica que el DUT está en el estado esperado.
- Pruebas con tramas de ANTICOLLISION y SELECT.
  - Verifica que las tramas de ANTICOLLISION son procesadas correctamente cuando el UID enviado corresponde a la del DUT.
  - Verifica que las tramas de ANTICOLLISION y SELECT son procesadas correctamente cuando el UID enviado no corresponde a la del DUT.
- Pruebas con tramas que contienen errores de CRC. Verifica que el DUT no responde a tramas que contienen CRCs inválidos, y transiciona al estado IDLE o HALT como es esperado.
- Pruebas que verifican que las tramas definidas en la parte cuatro de la norma no son aceptadas cuando la PICC no está en el estado correspondiente. En los estados: IDLE, READY, HALT y READY*, la PICC no debería responder a ninguna de esas tramas, y deberían ser tratadas como errores. En los estados ACTIVE y ACTIVE*, se aplica lo mismo, salvo por la trama RATS.

No es posible verificar el comportamiento del diseño entero con una sola simulación. Esto se debe a que el DUT está parametrizado en el tamaño del UID y el valor de este parámetro determina qué circuito es implementado mediante un bloque "generate". Por lo tanto el banco de prueba también es parametrizado en el tamaño del UID, y una simulación es ejecutada por cada tamaño de UID.

El informe de cobertura por estas tres simulaciones dan resultados por el DUT de: 94 %, 96 % y 97 % por UIDs simples, dobles y triples respectivamente. La diferencia es debido a que algunas partes del circuito no son usadas si solo hay uno o dos niveles cascadas. En el caso del UID triple las partes que faltan cobertura son mayormente casos imposibles de ocurrir (por ejemplo la parte fija del UID falta cobertura de toggle).

ISO/IEC 14443-3A

Este módulo instancia los módulos: **framing**, **routing** y **initialisation**, como se muestra en la Figura 3.20. Las entradas y salidas son descritas en el Cuadro 3.18.

Nombre	Dirección	Descripción
clk	Entrada	Reloj.
rst_n	Entrada	Reset activo bajo.
uid_variable	Entrada	Bits menos significativos del UID.
pause_n_synchronised	Entrada	Pausa, activa baja, sincronizada.
rx_iface_from_14443_2a	Entrada	Sumidero de bits de la *rx_interface*, conectado al módulo **ios14443_2a**
tx_iface_to_14443_2a	Salida	Fuente de bits de la *tx_interface*, conectado al módulo **ios14443_2a**.
rx_iface_to_14443_4a	Salida	Sumidero de bytes de la *rx_interface*, conectado al módulo **ios14443_4a**.
rx_crc_ok	Salida	Indica que la última trama recibida tiene CRC válido, conectado al módulo **ios14443_4a**.
tx_iface_from_14443_4a	Entrada	Fuente de bytes de la *tx_interface*, conectado al módulo **ios14443_4a**.
tx_append_crc_14443_4a	Entrada	La respuesta enviada del módulo **ios14443_4a** debe tener un CRC agregado.
iso14443_4a_deselect	Entrada	El módulo **ios14443_4a** recibió un S(DESELECT).
iso14443_4a_rats	Entrada	El módulo **ios14443_4a** recibió una trama RATS.
iso14443_4a_tag_active	Salida	La PICC está en el estado ACTIVE o ACTIVE*.

Cuadro 3.18: Entradas y Salidas del módulo **iso14443_3a**.

Este módulo es parametrizado y los parámetros son descritos en el Cuadro 3.19. Los parámetros del UID son pasados sin modificaciones al módulo **initialisation** y el parámetro *FDT_TIMING_ADJUST* es pasado al módulo **framing**, también sin modificación.

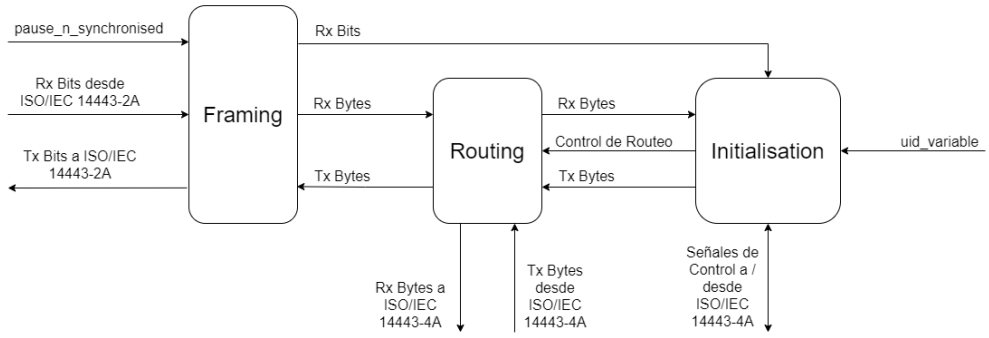

Figura 3.20: Diagrama en bloques del módulo **iso14443_3a**.

Nombre	Tipo	Descripción
UID_SIZE	UIDSize (enum)	La PICC tiene un UID simple, doble o triple.
UID_INPUT_BITS	int	El ancho de la señal *uid_variable*.
UID_FIXED	logic [] (un vector)	Bits más significativos del UID.
FDT_TIMING_ADJUST	int	Específica retardos en las rutas de recepción y transmisión externas a este módulo.

Cuadro 3.19: Parámetros del módulo **iso14443_3a**.

El banco de prueba para este módulo es muy parecido a lo del módulo **initialisation**. Las diferencias son:

- No usa una **deserialiser**, ya que eso está incluido en el DUT.
- Usa un monitor **TxBitIfaceMonitor** en vez de **TxByteIfaceMonitor**, porque el DUT incluye el módulo **serialiser** que convierte las respuestas a bits.
- Tiene un monitor **RxByteIfaceMonitor** para la salida *rx_iface_to_14443_4a*.
- Tiene un controlador **TxByteIfaceSourceDriver** para la entrada *tx_iface_from_14443_4a*.
- El FDT es verificado de la misma manera que en el banco de prueba del módulo: **framing**.
- La clase **CommsTestsSequence** es extendida por **ISO14443_3aTbSequence**, que incluye habilidades específicas por este banco de prueba. Mayormente las diferencias entre esta clase y la usada por el módulo **initialisation** son relacionadas con la verificación de las señales al módulo **iso14443_4a**.

Como en el banco de prueba del módulo **initialisation**, este banco de prueba es ejecutado tres veces, una vez para cada tamaño de UID. El informe de cobertura por estas

simulaciones dan resultados por el DUT de: 91 %, 92 % y 92 % por UIDs simples, dobles y triples respectivamente. Esos valores son para el DUT y sus submódulos. El DUT solo tiene 100 % cobertura en cada caso. Aunque unos de los submódulos faltan cobertura acá, todos tienen un porcentaje de cobertura alta en sus propios bancos de pruebas.

ISO/IEC 14443-4A

En este bloque de la implementación, solo hay un módulo **iso14443_4a**, que es responsable para recibir, actuar sobre, y responder a las tramas definidas en la parte cuatro de la norma. Esas tramas son: RATS, PPS, y bloques estándares. El protocolo de nivel aplicación es encapsulado un campo de información en bloques estándares de tipo I. Ese campo es reenviado a la aplicación, y las respuestas de la aplicación son empaquetadas y enviadas hacia el PCD. Las salidas y entradas son descritas en el Cuadro 3.20.

Nombre	Dirección	Descripción
clk	Entrada	Reloj.
rst_n	Entrada	Reset activo bajo.
power	Entrada	Señal desde el AFE para indicar si la potencia recibida es apta.
rx_iface	Entrada	Sumidero por la *rx_interface*, conectado al módulo **iso14443_3a**.
rx_crc_ok	Entrada	La última trama recibida tiene CRC válido.
tx_iface	Salida	Fuente por la *tx_interface*, conectada al módulo **iso14443_3a**.
tx_append_crc	Salida	Indica que la respuesta debe tener CRC agregado.
tag_active	Entrada	La PICC está en el estado ACTIVE.
rx_rats	Salida	Indica que la trama RATS fue recibida.
rx_deselect	Salida	Indica que la trama S(DESELECT) fue recibida y la respuesta fue enviada.
app_rx_iface	Salida	Fuente por la *rx_interface*, conectada a la aplicación.
app_tx_iface	Entrada	Sumidero por la *tx_interface*, conectado a la aplicación.
app_resend_last	Salida	La aplicación debe enviar su última respuesta de nuevo.

Cuadro 3.20: Entradas y Salidas del módulo **iso14443_4a**.

Cuándo una PICC recibe la trama SELECT con su UID, se transiciona al estado ACTIVE o ACTIVE*. En estos estados para terminar el proceso de inicialización el PCD envía una RATS. Cuando este módulo recibe esa trama mientras la entrada *tag_active* está en '1', pulsa su salida *rx_rats*. El módulo **initialisation** usa esa señal para transicionar al

estado PROTOCOL. La trama RATS contiene dos campos:

- CID (Card Identifier): El PCD asigna un CID a la PICC para poder direccionar tramas a esta PICC si hay más de una inicializada. Este campo es cuatro bits, con el valor 15 reservado para uso futuro. El CID asignado es válido hasta que la PICC es desactivada.
- FSDI: Una codificación de 4 bits que representa el tamaño máximo de tramas que el PCD puede recibir. Este campo es ignorado porque todas las tramas usadas en este trabajo tienen tamaños menores que lo mínimo que los PCDs tienen que soportar.

La respuesta a RATS es ATS que consiste en al menos un byte, pero hay algunos campos de bytes opcionales. Estos bytes opcionales no son implementados en este trabajo, los argumentos presentes en estos bytes tienen valores por defectos adecuados para este trabajo.

Después de la RATS y ATS, el PCD opcionalmente puede enviar la trama PPS para cambiar las tasas de transmisión y recepción. El PCD solo es permitido enviar la PPS inmediatamente después de la recepción de la respuesta ATS, por lo tanto la PICC ignora tramas PPS en cualquier otro momento. Debido a que este trabajo no soporta otras tasas de bits, este módulo solo responde a la PPS si las tasas de bits especificadas son: $f_c/128$ en cada dirección.

Un bloque estándar comienza con una cabecera de entre uno y tres bytes:

PCB:
: Contiene: el tipo del bloque (I, R, o S), unos campos específicos del tipo de bloque, y dos indicadores que especifican si los campos de CID y NAD están presentes.

CID:
: Usado para direccionar una trama a una PICC en particular. Este campo en las tramas enviadas desde la PICC también contiene dos bits que indican si la PICC está recibiendo suficiente potencia desde el campo electromagnético. Estos bits son asignados al valor de la entrada *power*, que es controlado por el AFE.

NAD (Node Address):
: Una PICC puede contener más de una aplicación y este campo permite el PCD direccionar tramas a una en particular. Este trabajo no soporta este campo, bloques estándares con una NAD son ignorados, como especificado en la norma.

Un objetivo de este trabajo es tomar muestras de múltiples de sensores discretos, así

que es necesario tener más de una PICC activada a la vez, por lo tanto este diseño soporta CIDs. Una PICC que soporta el campo CID debe aceptar un bloque estándar que contiene su CID, y en el caso que su CID está 0, debe también aceptar bloques estándares que no contienen un CID. Otras tramas deben ser ignoradas [13]. Este módulo incluye lógica combinatoria que determina si una trama es direccionada a esta PICC o no.

Bloques estándares de tipo I pueden contener un campo de información después de la cabecera. Esta información es un mensaje de nivel de aplicación. Cuándo este módulo recibe un bloque estándar tipo I, el campo de información es reenviado a la aplicación sobre la *app_rx_iface*. La respuesta de la aplicación es recibida sobre la *app_tx_iface*. Este módulo empaqueta la respuesta en un bloque estándar tipo I con los campos adecuados y envía la trama completa al PCD. Bloques estándares de tipo I pueden ser encadenados, esto permite el envío de tramas más grandes que el receptor soporta. La norma requiere que todos los PCDs y PICCs soporten tramas de al menos 16 bytes, y debido a que el protocolo de nivel aplicación de este trabajo solo usa tramas con tamaños menores que eso, bloques estándares encadenados no son soportados en esta implementación. Si este módulo recibe un bloque estándar tipo I encadenado, es ignorado.

Hay tres bloques estándares de tipo S definidos en la norma:

S(DESELECT):
: La PICC debe transicionar al estado HALT. Este módulo pulsa la salida *rx_deselect* después de haber enviado la respuesta, que causa el módulo **initialisation** transicionar al estado HALT.

S(WTX):
: La PICC puede enviar esta respuesta a un bloque estándar tipo I si necesita más tiempo para procesar la solicitud. Este mensaje no es soportado en este trabajo debido a que todas las respuestas de nivel aplicación pueden ser calculado de forma inmediata.

S(PARAMETERS):
: El PCD puede usar este mensaje para cambiar la configuración de la PICC, por ejemplo para cambiar la tasa de bits. Este mensaje es opcional y no es soportado en esta implementación.

Bloques estándares de tipos I y R tienen un bit en el campo PCB que se llama el número de bloque. La PICC también tiene un registro de un bit que guarda un número de bloque. Cuándo la PICC envía un bloque estándar de tipo I o R, el número de bloque

en la cabecera debe ser igual al valor en su registro. Las reglas de cómo manipular este registro en la PICC son:

> - La PICC debe iniciar su número de bloque a '1' cuando está activada.
> - Cuando la PICC recibe un bloque estándar de tipo I, debe invertir el valor en su registro antes de enviar la respuesta.
> - Cuando la PICC recibe un R(ACK) con número de bloque no igual a lo actual de la PICC, debe invertir el valor en su registro, antes de enviar la respuesta.
>
> [13, traducción mía]

Hay dos mensajes que usan bloques estándares de tipo R: R(ACK), y R(NAK). Las reglas de cómo la PICC debe responder a esos mensajes son:

> - Cuando un R(ACK) o un R(NAK) es recibido, si el número de bloque [en la cabecera] es igual al número de bloque actual de la PICC, la PICC debe reenviar su última respuesta.
> - Cuándo un R(NAK) es recibido, si el número de bloque [en la cabecera] no es igual al número de bloque actual de la PICC, la PICC debe responder con un R(ACK).
> - Cuándo un R(ACK) es recibido, si el número de bloque [en la cabecera] no es igual al número de bloque actual de la PICC, y la última respuesta de la PICC fue encadenada, la PICC debe seguir con el siguiente bloque en la cadena.
>
> [13, traducción mía]

Este mecanismo es diseñado para facilitar el PCD recuperar después de la pérdida de una trama. Debido a que esta implementación no soporta bloques encadenados, solo las primeras dos reglas son implementadas, y los bloques R(ACK) son ignorados cuándo su número de bloque no es igual al valor guardado en la PICC. En el caso de recibir un R(ACK) o R(NAK) con el número de bloque igual al valor guardado en la PICC, la última respuesta es reenviada, y si la última respuesta fue un bloque estándar de tipo I, la salida *app_resend_last* es pulsada para pedir la aplicación repetir su última respuesta también.

La ruta de recepción de tramas es parecida a la del módulo **initialisation**, la trama es guardada en un buffer. Después de la recepción de la cabecera, si este módulo determina que la trama es un bloque estándar tipo I direccionada a esta PICC, los bytes del campo de información son reenviados a la aplicación. El buffer de recepción tiene cinco bytes para poder guardar la trama de tamaño máximo: PPS. Aunque bloques estándares de tipo I pueden ser más largos, este módulo solo guarda los primeros dos bytes de la cabecera: PCB y CID.

La ruta de transmisión también es parecida a la del módulo **initialisation**. Un buffer de dos bytes es usado para construir las respuestas. Cuando una respuesta está lista para enviar, la salida *tx_iface.data_valid* es fijada en '1', y el primer byte de la respuesta es asignado a la salida *tx_iface.data*. Cuando el sumidero pide el próximo byte mediante la señal *tx_iface.req*, la salida *tx_iface.data* es actualizada con el próximo byte. En el caso de que la respuesta sea un bloque estándar tipo I, el buffer es llenado con la cabecera correspondiente. Después del envío de la cabecera este módulo actúa como un pass-through por la respuesta de la aplicación. Debido a que las respuestas: ATS, PPS y todos los bloques estándares tienen un CRC, la salida *tx_append_crc* es un '1' constante.

El banco de prueba para este módulo es parecido a los bancos de pruebas de los módulos **initialisation** y **iso14443_3a**, pero ejecuta pruebas distintas que son diseñadas para verificar la parte cuatro de la norma. El banco de prueba incluye la clase **ISO14443_4a-TbSequence** que extiende **CommsTestSequence** para proveer habilidades específicas a este banco de prueba:

- Verifica que los campos de información de bloques estándares de tipos I son reenviados a la aplicación, y que otras tramas no son reenviadas a la aplicación.
- Maneja las entradas *rx_crc_ok*, *power* y *tag_active* del DUT y verifica el comportamiento de sus salidas: *rx_rats*, *rx_deselect*, y *rx_app_resend_last*.
- Simula respuestas desde la aplicación.
- Sustituye las tareas **go_to_state_*()**. La secuencia usa estas tareas para poner el DUT en el estado apto para cada prueba, pero en este caso el DUT no contiene la máquina de estados de la parte tres de la norma, por lo tanto las transiciones entre estos estados tienen que ser simuladas.

La parte cuatro de la norma especifica algunas reglas sobre el comportamiento de una PICC, las pruebas en este banco de prueba verifican que esas reglas se cumplen, entre otras cuestiones. Las pruebas ejecutadas son en la tarea **run_all_part4_tests()** de la clase **CommsTestSequence**. Unos ejemplos de las pruebas son:

- Verifica que cuando una PICC en el estado ACTIVE o ACTIVE* recibe una trama RATS con CID válido, envía la respuesta ATS esperada, y si la PICC es en otro estado o el CID asignado no es válido, la PICC no envía una respuesta.
- Verifica que el DUT solo responde a la trama PPS cuándo es la primera trama después del envío de la respuesta ATS.
- Verifica que cuando la PICC está en el estado PROTOCOL, los campos de información en los bloques estándares de tipo I, son enviados a la aplicación.

- Verifica que el DUT cumple con las reglas descritas arriba sobre los números de bloques.

El informe de cobertura da el DUT un resultado de 98 %. Las partes que faltan cobertura son situaciones imposibles de ocurrir, por ejemplo: la salida *tx_append_crc* es un '1' constante, por lo tanto falta cobertura de toggle.

ISO/IEC 14443A

Este módulo es el módulo más alto del núcleo IP. Debería ser instanciado en la aplicación y conectado a un AFE conforme con parte dos de la norma. El módulo simplemente instancia los módulos: **iso14443_2a**, **iso14443_3a**, **iso14443_4a**, y los conecta de forma adecuada. Este módulo es parametrizado y los parámetros son descritos en el Cuadro 3.21. Las salidas y entradas son descritas en el Cuadro 3.22.

Nombre	Tipo	Descripción
UID_SIZE	UIDSize (enum)	La PICC tiene un UID simple, doble o triple.
UID_INPUT_BITS	int	El ancho de la señal *uid_variable*.
UID_FIXED	logic [] (un vector)	Bits más significativos del UID.
FDT_TIMING_ADJUST	int	Específica retardos en la ruta de recepción y transmisión externa a este módulo.

Cuadro 3.21: Parámetros del módulo **iso14443a**.

Los parámetros del UID son igual a los definidos en el módulo **iso14443_3a**, y son pasados a ese módulo sin modificaciones. El parámetro *FDT_TIMING_ADJUST* es usado para ajustar el comportamiento del módulo **FDT** para que cumpla con el FDT especificado en la norma. El módulo **iso14443_2a** contiene tres ciclos de retardo en el envío de las respuestas. Por lo tanto, el módulo **iso14443_3a** es instanciado con su parámetro *FDT_TIMING_ADJUST* igual al valor del parámetro en este módulo más tres.

El banco de prueba para este módulo está basado en una mezcla de los bancos de pruebas de los módulos **iso14443_3a** y **iso14443_4a**. Este banco de prueba usa el modelo del AFE para generar el reloj y las pausas. El FDT es verificado, midiendo el tiempo entre el último flanco ascendente de la señal *pause_n_synchronised* y el primer flanco ascendente de la señal *lm_out*. Las pruebas ejecutadas son las pruebas de inicialización y las de la parte cuatro de la norma. Esas pruebas son repetidas diez veces con la parte variable del UID diferente cada vez.

Nombre	Dirección	Descripción
clk	Entrada	Reloj.
rst_n	Entrada	Reset, activo bajo. El flanco ascendente debe ser sincronizado al reloj.
uid_variable	Entrada	Bits menos significativos del UID.
power	Entrada	Señal desde el AFE que indica si la potencia recibida es apta.
pause_n_synchronised	Entrada	Pausa, activa baja, sincronizada.
lm_out	Salida	Salida al modulador de carga para enviar respuestas.
app_rx_iface	Salida	Fuente por la *rx_interface*, conectada a la aplicación.
app_tx_iface	Entrada	Sumidero por la *tx_interface*, conectada a la aplicación.
app_resend_last	Salida	Un indicador para pedir la aplicación reenviar su última respuesta.
iso14443a_version	Salida	La versión de este núcleo IP, debe ser constante.

Cuadro 3.22: Entradas y Salidas del módulo **iso14443a**.

Como por el banco de prueba del módulo **initialisation**, este banco de prueba es ejecutado tres veces, una vez para cada tamaño del UID. El informe de cobertura por estas tres simulaciones dan resultados por el DUT de: 92 %, 93 % y 93 % por UIDs simples, dobles y triples respectivamente. Esos resultados son por el DUT y sus submódulos. El DUT solo, tiene 89 %, 93 % y 95 % cobertura en cada caso. Las partes que faltan cobertura son porque la salida *iso14443a_version* y la señal interna *part4_tx_append_crc* son constantes. Aunque algunos de los submódulos no poseen alta cobertura en esta prueba, todos tienen un alto porcentaje de cobertura en sus propios bancos de pruebas.

Otros

Hay algunos módulos extras que no son usados en este núcleo IP, son incluidos para uso en la aplicación si son necesarios.

synchroniser

Este módulo sincroniza una señal al dominio del reloj para prevenir metaestabilidad [16]. Este módulo es parametrizado y sus parámetros se describen en el Cuadro 3.23. Las

salidas y entradas son descritas en el Cuadro 3.24. La implementación es dos registros conectados como se muestra la Figura 3.21.

Nombre	Tipo	Descripción
WIDTH	int	Ancho de la señal a sincronizar.
RESET_VAL	logic [WIDTH-1:0]	Valor de la salida cuándo el módulo está en un estado de reset.

Cuadro 3.23: Parámetros del módulo **synchroniser**.

Nombre	Dirección	Descripción
clk	Entrada	Reloj.
rst_n	Entrada	Reset activo bajo.
D	Entrada	Señal a sincronizar.
Q	Salida	Señal sincronizada.

Cuadro 3.24: Entradas y Salidas del módulo **synchroniser**.

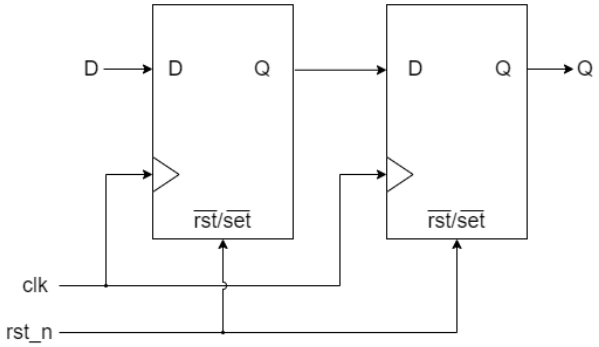

Figura 3.21: El circuito implementado en el módulo **synchroniser**.

Debido a la simplicidad de este módulo no hay un banco de prueba, el comportamiento debería ser verificado en los bancos de prueba de los módulos que lo usan.

active_low_reset_synchroniser

Este módulo es un sincronizador de reset para sincronizar el flanco ascendente de resets asincrónicos, activos bajos, para prevenir metaestabilidad cuándo el diseño sale del estado de reset. Las salidas y entradas se describe en el Cuadro 3.25. La implementación simplemente es dos registros como se muestra la Figura 3.22. Como en el caso del módulo synchroniser, no hay banco de prueba para este módulo.

Nombre	Dirección	Descripción
clk	Entrada	Reloj.
rst_n_in	Entrada	Reset a sincronizar, activo bajo.
rst_n_out	Salida	Reset con su flanco ascendente sincronizado, activo bajo.

Cuadro 3.25: Entradas y Salidas del módulo **active_low_reset_synchroniser**.

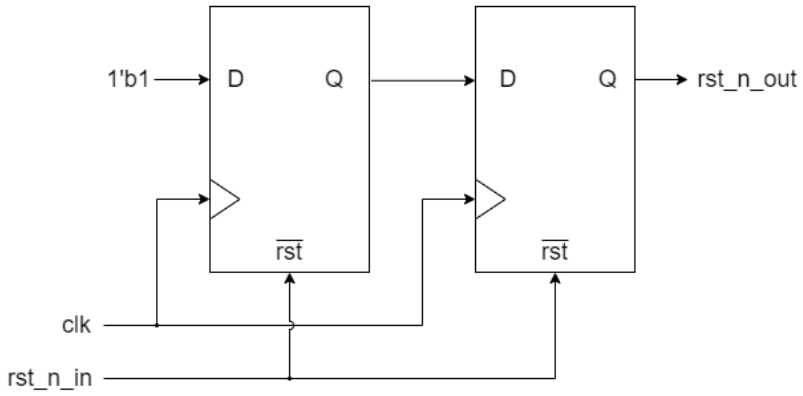

Figura 3.22: El circuito implementado en el módulo **active_low_reset_synchroniser**.

pause_n_latch_and_synchroniser

En el caso de un AFE que produce un reloj que se detiene durante las pausas, si no se puede garantizar que haría al menos un flanco ascendente del reloj mientras la pausa está activa, sería necesario usar un circuito asincrónico para retener las pausas hasta que el reloj comience de nuevo. El circuito usado es un FFD (Flip Flop D) con un reset y un set asincrónicos, conectado como se muestra en la Figura 3.23. La entrada *pause_n_async* es conectada a la reset asincrónica, así durante la pausa, la salida *pause_n_latched* es '0'. La entrada D del FFD es un '1' constante, así en el primer flanco ascendente del reloj después de la pausa, la salida *pause_n_latched* vuelve a '1'. La entrada *rst_n* está conectada a la set asincrónica, así durante un reset la salida es '1'. Debido a que se usa un reset asincrónico el flanco descendente de la *pause_n_latched* es asincrónica al reloj, por lo tanto es necesario usar un sincronizador para prevenir metaestabilidad. La Figura 3.24 muestra el diseño final.

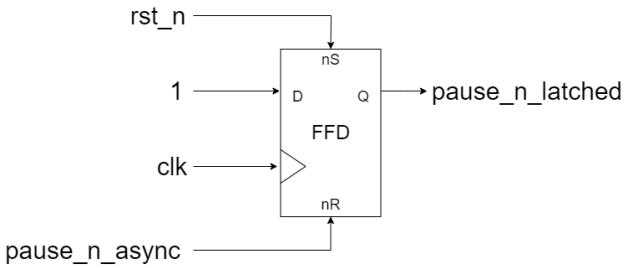

Figura 3.23: Circuito asincrónico por la retención de pausas.

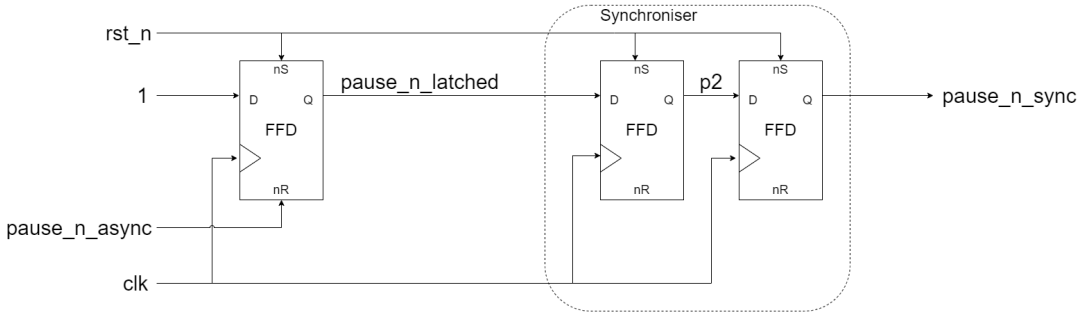

Figura 3.24: El circuito implementado en el módulo **pause_n_latch_and_synchroniser**.

Las salidas y entradas son descritas en el Cuadro 3.26. La Figura 3.25 muestra un diagrama de timing de este diseño cuándo el reloj está detenido durante la pausa entera.

Nombre	**Dirección**	**Descripción**
clk	Entrada	Reloj.
rst_n	Entrada	Reset activo bajo.
pause_n_async	Entrada	Pausa, asincrónica, desde el AFE.
pause_n_sync	Salida	Pausa retenida y sincronizada.

Cuadro 3.26: Entradas y Salidas del módulo **pause_n_synch_and_latch**.

Para cumplir con el FDT definida en la parte tres de la norma es necesario compensar por el retardo entre los flancos ascendentes de las señales *pause_n_async* y *pause_n_sync*. Este retardo es dos ciclos del reloj más t_{p2c}. En este trabajo el mecanismo para compensar por este retardo es con el parámetro *FDT_TIMING_ADJUST* del módulo más alto del núcleo IP. Cualquier diseño que instancia este núcleo IP y usa este módulo para retener y sincronizar las pausas debe incluir estos retardos en la calculación del parámetro *FDT_TIMING_ADJUST*.

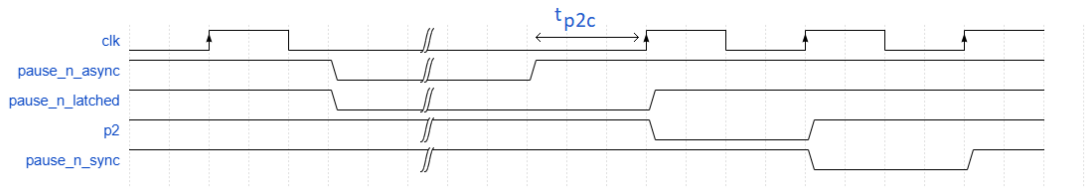

Figura 3.25: Diagrama de timing de una pausa sin reloj.

Para verificar este módulo el modelo analógico: **analogue_sim** es usado para enviar miles de pausas con la relación entre la señal *pause_n_async* y el reloj elegido aleatoriamente por cada pausa. Unas aserciones son usadas para verificar los retardos entre la entrada y la salida. El informe de cobertura da un resultado de 100 % por el DUT.

Aplicación - Interfaz con el sensor y ADC

La aplicación es el bloque principal de este trabajo, instancia el núcleo IP ISO/IEC 14443A, recibe, actúa sobre, y responde a los mensajes de nivel aplicación contenidos en bloques estándares de tipo I. El propósito de este bloque es proveer un método para manipular las salidas al sensor y al ADC, y para devolver la muestra al PCD. El Cuadro 3.27 muestra las señales conectadas al sensor y al ADC:

Nombre	Dirección	Descripción
sens_config[2:0]	Salida	Configuración del sensor.
sens_enable	Salida	Activa el sensor.
sens_read	Salida	Lee el sensor.
sens_version[3:0]	Entrada	Versión del sensor.
adc_enable	Salida	Activa el ADC.
adc_read	Salida	Lee el ADC.
adc_conversion_complete	Entrada	Indica que el ADC ha tomado la muestra.
adc_value[15:0]	Entrada	Dato muestreado.
adc_version[3:0]	Entrada	Versión del ADC.

Cuadro 3.27: Señales en la interfaz con el sensor y el ADC.

La Figura 3.26 muestra el proceso para tomar una muestra del sensor. Primero se fija *sens_config* con el valor deseado para elegir la configuración del sensor. Después las señales *sens_enable* y *adc_enable* son asignadas el valor '1' para activar los dos componentes. Después de un tiempo (t_1) la señal *sens_read* debe ser asignado el valor '1' para pedir que el sensor comienza una lectura. Siguiente, después de un tiempo (t_2) la señal

adc_read debe ser asignado el valor '1' para pedir que el ADC comience a tomar una muestra. Cuándo el ADC termina la lectura debe pulsar la señal *adc_conversion_complete* por un ciclo del reloj para indicar que la señal *adc_value* es válida y estable. La interpretación de estas señales está a cargo del diseñador del sensor y del ADC, siendo ésta fuera del alcance de este trabajo. El valor de *sens_config* y los valores de t_1 y t_2 pueden ser controlados por el PCD al tiempo de la lectura.

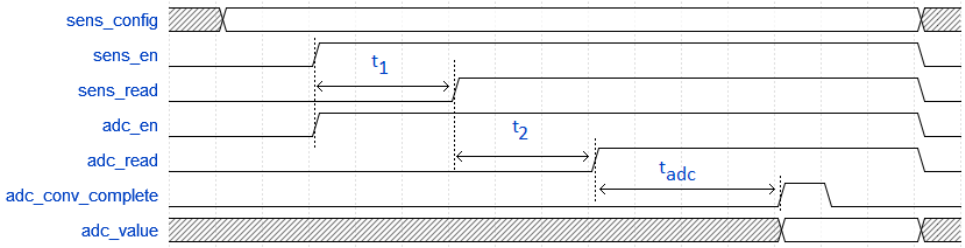

Figura 3.26: Diagrama de timing de una lectura del sensor.

Sincronización del muestreo

Para cumplir con las metas del proyecto marco es requerido poder tomar muestras de múltiples de TAGs de forma sincronizada. Un método para sincronizar las muestras sería usar una trama broadcast para iniciar el proceso en cada PICC a la vez. Desafortunadamente la norma ISO/IEC 14443-4 no permite al PCD envíar bloques estándares de forma broadcast. Sería posible modificar el diseño del núcleo IP para permitir esto, por ejemplo con el CID 15 que es reservado para uso futuro en la norma. La desventaja de esto es que no conforme con la norma y significaría que el núcleo IP no sería una implementación genérica.

El método usado para sincronizar las muestras en este trabajo es parecido a como el FDT funciona para sincronizar las respuestas de las tramas de ANTICOLLISION. Todas las PICCs reciben las pausas mediante el mismo circuito, por lo tanto el flanco ascendente de la pausa debe ser recibido en todas las PICCs a aproximadamente el mismo momento. Además las PICCs recuperan sus relojes de la portadora, así todas las PICCs en el campo electromagnético de un PCD tienen relojes con exactamente la misma frecuencia. Estos hechos significan que si las PICCs tienen un contador que es reseteado a cero en cada flanco ascendente de la pausa, y en los otros ciclos del reloj incrementa su valor, los contadores serían sincronizados dentro de un margen de error de unos ciclos del reloj. El margen de error exacto depende en la implementación del AFE.

Unos de los mensajes en el protocolo propietario diseñado por este trabajo tienen un campo de sincronización. Una PICC que recibe uno de esos mensajes no realiza la acción especificada hasta que su contador de sincronización ha llegado al valor especificado en ese campo. El PCD puede enviar el mismo mensaje a cada PICC activa en turno. Con un tiempo de sincronización suficientemente grande el PCD puede enviar la siguiente trama antes de que la acción sea realizada, reseteando los contadores en todas las PICCs. De esta manera se logra que todas las PICCs realizan la misma acción de forma sincronizada.

El tiempo de sincronización tiene que ser más grande que el tiempo entre el flanco ascendente de la última pausa de una trama enviada por el PCD y el flanco ascendente de la primera pausa en la próxima trama: $t_{synch} > t_{fdt1} + t_{respuesta} + t_{fdt2} + t_p$ como se muestra la Figura 3.27. Los valores en ciclos del reloj son:

$t_{fdt1} = 1236$ ciclos:
: El FDT máximo entre el último flanco ascendente de una trama desde el PCD y el primer flanco de modulación de la respuesta, como definida en la norma [12].

$t_{respuesta} = 128\,(B + 1)$ ciclos:
: El tiempo necesario para enviar la respuesta, dónde B es el máximo número de bits en la respuesta incluyendo la cabecera, el CRC y los bits de paridad. En el protocolo la respuesta más grande a un comando con sincronización es diez bytes, por lo tanto $B = 90$ bits, y $t_{respuesta} = 11\,648$ ciclos.

$t_{fdt2} = 1272$ ciclos:
: El FDT entre el último flanco de modulación en una respuesta y el primer flanco de la pausa de la próxima trama del PCD, como definida en la norma [12]. Este valor es el mínimo requerido por la norma, el valor actual depende en la implementación del PCD.

$t_p \approx 32$ ciclos:
: La duración de una pausa.

Estos valores dan un resultado de $t_{sync} > 14\,188$ ciclos ($t_{sync} > 1{,}0$ ms).

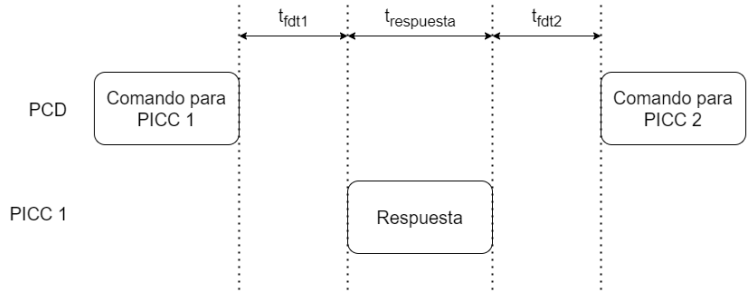

Figura 3.27: Timings para sincronizar acciones en más de una PICC.

Protocolo Propietario

El protocolo de nivel aplicación es un protocolo propietario diseñado para este trabajo, el Apéndice B contiene un header del lenguaje de programación C con definiciones y estructuras por este protocolo. Cada mensaje y respuesta comienzan con una cabecera de cinco bytes que consiste en un valor mágico de cuatro bytes, y un byte para indicar el tipo de comando. El valor mágico está presente porque la norma ISO/IEC 14443 no define un método para determinar qué protocolo de aplicación usa una PICC. Al comenzar cada mensaje y respuesta con un valor mágico provee un método para verificar que un PCD y una PICC son compatibles. Los comandos definidos son:

IDENTIFY:

Solicitud por las versiones de los bloques. El PCD puede usar la respuesta para determinar qué revisión de silicio usa una PICC. No lleva argumentos.

SET_SIGNAL:

Solicitud para cambiar las señales al sensor y al ADC. Esta acción puede ser sincronizada con otras PICCs. Lleva tres argumentos:

- *sync[15:0]*: Tiempo de sincronización.
- *mask[7:0]*: Máscara de bits para especificar qué señales cambiar. Los bits son:
  - 7-5: *sens_config[2:0]*
  - 4: *sens_enable*
  - 3: *sens_read*
  - 2: *adc_enable*
  - 1: *adc_read*
  - 0: reservado
- *value*: Valores nuevos de esas señales.

AUTO_READ:

Solicitud para tomar una muestra del sensor automáticamente. Esta acción puede ser sincronizada con otras PICCs. Lleva tres argumentos, los argumentos *timing1* y *timing2* corresponden con el t_1 y t_2 mostrados en la Figura 3.26:

- *sync[15:0]*: Tiempo de sincronización.
- *timing1[31:0]*: Tiempo entre los flancos ascendentes de las señales *sens_enable* y *adc_enable*, y el flanco ascendente de la señal *sens_read*. El protocolo usa 32 bits para este argumento, pero solo los 25 bits menos significativos son usados.
- *timing2[31:0]*: Tiempo entre el flanco ascendente de la señal *sens_read* y el flanco ascendente de la señal *adc_read*. El protocolo usa 32 bits para este argumento, pero solo los 25 bits menos significativos son usados.

GET_RESULT:

Solicitud por el valor de la última muestra. No lleva argumentos.

ABORT:

Solicitud para abandonar la acción concurrente, sea una AUTO_READ o una SET_SIGNAL. No lleva argumentos.

Las respuestas son:

IDENTIFY_REPLY:

Respuesta al IDENTIFY, contiene:

- Versión de este protocolo.
- Versión de esta aplicación.
- Versión del núcleo IP ISO/IEC 14443A.
- Versión del sensor MOSFET de radiación.
- Versión del ADC.

GET_RESULT_REPLY:

Respuesta al GET_RESULT, contiene el valor de la última muestra del sensor, y unas flags de estado:

- *conv_complete*: Indica que el ADC ha terminado una muestra después de la última vez que este flag fue enviado.
- *already_busy*: Indica que la PICC sigue ocupada, y no puede procesar esta solicitud.
- *unexpected_pause*: Indica que una pausa fue recibida durante los periodos de t_1, t_2, o t_{adc} del proceso AUTO_READ. Durante esos periodos el PCD debe quedarse quieto para que la PICC pueda tener un reloj continuo para medir

los timings de forma precisa.

- *error*: Indica que el último mensaje contuvo un error del protocolo, por ejemplo el comando no fue conocido, o el mágico fue inválido.

STATUS_REPLY:

Respuesta a SET_SIGNAL, AUTO_READ y ABORT, contiene los mismos flags de estado de la respuesta GET_RESULT_REPLY.

Ejemplos

Un ejemplo de la lectura de un único sensor mediante el comando SET_SIGNAL. En este proceso los timings entre las etapas son controlados completamente por cuándo el PCD envía las tramas. El argumento *sync* en todos casos debería ser cero.

1. El PCD activa la PICC.
2. El PCD envía el comando IDENTIFY y verifica la respuesta.
3. El PCD envía el comando SET_SIGNAL para fijar la señal *sens_config* como desea y las otras señales en '0'.
4. El PCD envía el comando SET_SIGNAL para fijar las señales *sens_enable* y *adc_enable* en '1'.
5. El PCD espera por el t_1 deseado.
6. El PCD envía el comando SET_SIGNAL para fijar la señal *sens_read* en '1'.
7. El PCD espera por el t_2 deseado.
8. El PCD envía el comando SET_SIGNAL para fijar la señal *adc_read* en '1'.
9. El PCD espera por el t_{adc}, que depende en la implementación del ADC.
10. El PCD envía el comando GET_RESULT, verifica que la respuesta no indica un error, y guarda la muestra.

Un ejemplo de la lectura de un único sensor mediante el comando AUTO_READ. El argumento *sync* en todos casos debería ser cero.

1. El PCD activa la PICC.
2. El PCD envía el comando IDENTIFY y verifica la respuesta.
3. El PCD envía el comando SET_SIGNAL para fijar la señal *sens_config* como desea y las otras señales en '0'.
4. El PCD envía el comando AUTO_READ, con los tiempos t_1 y t_2 deseados, y verifica la respuesta.
5. El PCD espera por más que $t_1 + t_2 + t_{adc}$ para que el proceso se termine.

6. El PCD envía el comando GET_RESULT, verifica que la respuesta no indica un error, y guarda la muestra.

Un ejemplo de la lectura de dos sensores de forma sincronizada mediante el comando AUTO_READ. La Figura 3.28 muestra el mismo flujo de forma gráfica.

1. El PCD activa las dos PICCs.
2. El PCD envía el comando IDENTIFY a PICC 1 y PICC 2 en turno, y verifica las respuestas.
3. El PCD envía el comando SET_SIGNAL a PICC 1 y PICC 2 en turno, para fijar las señales *sens_config* como desea, y las otras señales en '0'. El argumento *sync* debería ser cero, siendo que este comando no necesita ser sincronizado.
4. El PCD envía el comando AUTO_READ a PICC 1 y PICC 2 en turno, especificando los mismos t_{sync}, t_1 y t_2, en los dos mensajes.
5. El PCD espera más que $t_{sync} + t_1 + t_2 + t_{adc}$ para que los procesos terminen.
6. El PCD envía el comando GET_RESULT a PICC 1 y PICC 2 en turno, verifica las respuestas no indicas errores, y guarda las muestras.

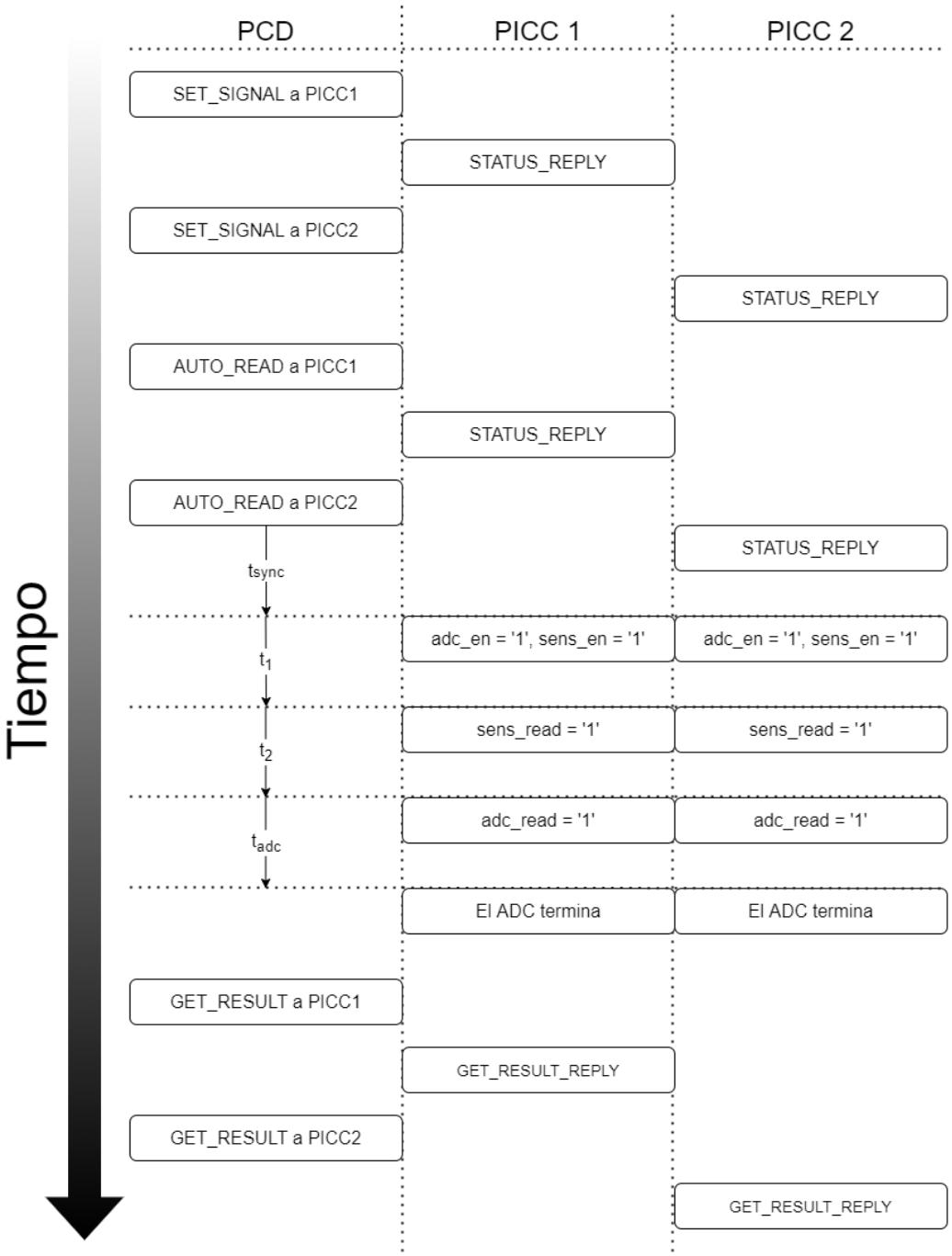

Figura 3.28: Ejemplo de la lectura de dos TAGs de forma sincronizada.

Marco de Verificación Extendido

El Marco de Verificación usado en el núcleo IP ISO/IEC 14443A es extendido para permitir la verificación de esta aplicación. Tres componentes extras son agregados.

- Un paquete de SystemVerilog **ProtocolGenerator** que tienen funciones para generar colas de bytes por los mensajes de nivel aplicación.
- Una secuencia **AppCommsTestSequence** que extiende **CommsTestSequence**. Esta secuencia contiene unas pruebas adicionales para verificar el comportamiento de la aplicación. Estas pruebas son descritas dónde son usadas primero.
- Un modelo del ADC. Este modelo monitoriza la señal *adc_read* por flancos ascendentes. Un número de ciclos después de la activación de esa señal, fija la señal *adc_value* en un valor aleatorio, y pulsa la señal *adc_conversion_complete* por un ciclo. El retardo es elegido aleatoriamente entre un mínimo y un máximo que son parámetros del modelo.

Implementación

Esta parte del diseño consiste en tres módulos:

- **signal_control**: Control de la interfaz con el sensor y el ADC.
- **adapter**: Procesa el protocolo propietario, realiza la acción pedido mediante el módulo **signal_control**, y devuelve la respuesta.
- **radiation_sensor_digital_top**: El módulo más alto de este trabajo.

signal_control

Este módulo es responsable por la interfaz con el sensor y el ADC. Actúa sobre los comandos SET_SIGNAL y AUTO_READ para realizar el muestreo del sensor. Cuadro 3.28 muestra las entradas y salidas del módulo.

La implementación consiste en un contador y una máquina de estados con cinco estados: IDLE, SYNC, TIMING1, TIMING2 y WAIT_FOR_ADC. En los estados IDLE y SYNC, el contador es reseteado a cero cuándo se detecta un flanco ascendente en la entrada *pause_n_synchronised*, e incrementa su valor en los otros ciclos. Si una pausa es detectada en cualquier otro estado la salida *unexpected_pause* es asignada '1', para avisar el PCD que una pausa fue recibida durante el proceso del muestreo. Debido a que el reloj se detiene

durante las pausas, una pausa en estos estados tiene dos implicaciones no deseadas: por un lado que los timings no son precisos; y por otro que puede existir variaciones en la tensión VDD que impacten negativamente en la precisión del muestreo.

Nombre	**Dirección**	**Descripción**
clk	Entrada	Reloj.
rst_n	Entrada	Reset activo bajo.
pause_n_synchronised	Entrada	Pausa, activa baja, sincronizada.
sync_timing	Entrada	Número de ciclos de sincronización.
auto_read_timing1	Entrada	Número de ciclos por t_1.
auto_read_timing2	Entrada	Número de ciclos por t_2.
cmd	Entrada	Comando recibido
set_signals_mask	Entrada	Máscara de bits en el comando SET_SIGNAL
set_signals_value	Entrada	Valor nuevo de las señales en el comando SET_SIGNAL
start	Entrada	Señal de activación.
abort	Entrada	Abandona el proceso concurrente.
result_read	Entrada	Indica que la última muestra fue enviada al PCD.
signals	Salida	Estructura conteniendo las salidas que conectan al sensor y al ADC.
adc_conversion_complete	Entrada	Indica que la entrada *adc_value* contiene una muestra válida, desde el ADC
adc_value	Entrada	Dato muestreado.
busy	Salida	Indica este módulo está ocupado.
unexpected_pause	Salida	Indica que una pausa fue recibida durante t_1, t_2 o t_{adc}.
cached_adc_conversion_complete	Salida	Indica que la salida *cached_adc_value* es válida.
cached_adc_value	Salida	Último dato muestreado.

Cuadro 3.28: Entradas y Salidas del módulo **signal_control**.

En el estado IDLE, el módulo se transiciona al estado SYNC cuando *start* está '1'. En el estado SYNC el módulo espera por el contador a tener un valor mayor o igual que la entrada *sync_timing*. En este momento la acción realizada depende en el tipo del comando:

SET_SIGNAL:

Las salidas al sensor y al ADC son asignado al resultado de:

$$(signals \ \& \sim set\_signals\_mask) \mid (set\_signals\_value \ \& \ set\_signals\_mask)$$

Eso es decir que las señales especificadas en la máscara de bits se cambian a los valores nuevos, y las otras señales se quedan en su estado original. Si esta operación causa un flanco ascendente en la señal *adc_read*, se transiciona al estado WAIT_FOR_ADC, y si no, se vuelve al estado IDLE.

AUTO_READ:

Las señales *sens_enable* y *adc_enable* son asignados el valor '1', el contador es reseteado a cero, y se transiciona al estado TIMING1.

En el estado TIMING1 el módulo espera por el contador a tener un valor igual que la entrada *auto_read_timing1*, y entonces fija la señal *sens_read* a '1', el contador es reseteado a cero, y se transiciona al estado TIMING2. En el estado TIMING2 el módulo espera por el contador a tener un valor igual que la entrada *auto_read_timing2*, y entonces fija la señal *adc_read* a '1', y se transiciona al estado WAIT_FOR_ADC. Finalmente, en el estado WAIT_FOR_ADC el módulo espera por la entrada *adc_conversion_complete* estar '1'. Cuando esto ocurre, fija la salida *cached_adc_conversion_complete* en '1' y asigna la entrada *adc_value* a la salida *cached_adc_value*. Después se vuelve al estado IDLE.

En cualquier momento si la entrada *abort* está '1', la máquina de estados vuelve al estado IDLE y las salidas son reseteadas a '0'. Cuándo la entrada *result_read* es '1' para indicar que la última muestra fue enviada al PCD, la salida *cached_adc_conversion_complete* es asignada el valor '0'.

Para verificar este módulo, el banco de prueba contiene un bloque que monitorea las señales: *pause_n_synchronised*, *sens_enable*, *sens_read*, *adc_enable* y *adc_read*. Los tiempos entre flancos de esas señales son medidos y comparados con los tiempos esperados calculados desde las entradas *sync_timing*, *auto_read_timing1* y *auto_read_timing2*. Además verifica que las señales cambian a los valores esperados. El modelo del ADC es usado para controlar las entradas al DUT *adc_value* y *adc_conversion_complete*. Cada prueba es repetida mil veces con valores aleatorios. Las pruebas ejecutadas son:

- Simula el comando SET_SIGNAL, sin tiempo de sincronización.
- Simula el comando SET_SIGNAL, con tiempo de sincronización, sin pausas.
- Simula el comando SET_SIGNAL, con tiempo de sincronización, con pausas.

- Las últimas tres pruebas son repetidas por el comando AUTO_READ.
- Simula el comando AUTO_READ, con pausas durante los periodos t_1, t_2 y t_{adc}. Esta prueba verifica la salida: *unexpected_pause*.
- Simula los comandos SET_SIGNAL y AUTO_READ, pulsando la entrada *abort* en el medio del proceso para abandonar la muestra.

Finalmente hay algunas aserciones:

- Verifica las salidas cuándo el DUT está en reset.
- Verifica que la salida *busy* está en '1' el ciclo después de un pulso en la entrada *start*, cuando la entrada *cmd* es SET_SIGNAL o AUTO_READ.
- Verifica que las salidas son correctas después de un pulso en la entrada *abort*.
- Verifica que un pulso en la entrada *adc_conversion_complete* produce que la salida *cached_adc_conversion_complete* suba a '1', y que la salida *cached_adc_value* sea igual al resultado devuelto del modelo del ADC.

El informe de cobertura da un resultado de 96 % por el DUT. La mayoría de casos que faltan cobertura son situaciones imposibles de ocurrir. La única parte del DUT que falta cobertura es las entradas *auto_read_timing1* y *auto_read_timing2* son limitados a 19 bits en el banco de prueba en vez de los 25 bits en el DUT, por lo tanto esas entradas y los bits más significativos del contador faltan cobertura de toggle. Esto es para reducir la duración de la simulación.

adapter

Este módulo es un adaptador entre el núcleo IP y el módulo **signal_control**. Recibe los mensajes del protocolo de nivel aplicación desde el núcleo IP, instancia el módulo **signal_control**, manipula sus entradas, y envía las respuestas al núcleo IP por transmisión al PCD. El Cuadro 3.29 muestra las entradas y salidas del módulo.

Cuando el núcleo IP recibe un bloque estándar de tipo I direccionada a esta PICC, envía el mensaje contenido en el campo de información a la aplicación. Este módulo recibe esas tramas en un buffer de quince bytes, lo que es el tamaño del mensaje más largo (AUTO_READ). Después de haber recibido una trama entera, el valor mágico de la cabecera es verificado. En el caso que no sea válido, este módulo no envía una respuesta. Si el mágico es correcto, se arma la respuesta y si es un comando AUTO_READ o SET_SIGNAL, comienza la operación mediante el módulo signal_control.

Nombre	Dirección	Descripción
clk	Entrada	Reloj.
rst_n	Entrada	Reset, activo bajo.
pause_n_synchronised	Entrada	Pausa, activa baja, sincronizada.
rx_iface	Entrada	Sumidero de la *rx_interface*
tx_iface	Salida	Fuente de la *tx_interface*.
app_resend_last	Entrada	Pedido para reenviar la última respuesta.
sens_config	Salida	Configuración del sensor.
sens_enable	Salida	Activa el sensor.
sens_read	Salida	Lee el sensor.
adc_enable	Salida	Activa el ADC.
adc_read	Salida	Lee el ADC.
adc_conversion_complete	Entrada	El ADC ha tomado la muestra.
adc_value	Entrada	Dato muestreado.
const_iso_iec_14443a_digital_version	Entrada	Versión del núcleo IP.
const_iso_iec_14443a_AFE_version	Entrada	Versión del AFE.
const_sensor_version	Entrada	Versión del sensor.
const_adc_version	Entrada	Versión del ADC.

Cuadro 3.29: Entradas y Salidas del módulo **adapter**.

La parte cuatro de la norma especifica que si la PICC recibe un bloque estándar R(ACK) o R(NAK), con número de bloque igual al de la PICC, la última respuesta debe ser reenviada [13]. En este caso el núcleo IP pulsa su salida *app_resend_last*, que está conectada a la entrada de este módulo. La respuesta es construida en un buffer de diez bytes que quedan constantes hasta que el próximo comando es recibido. Cuándo la entrada *app_resend_last* es '1', la respuesta en el buffer de transmisión es reenviado.

Para verificar este módulo el banco de prueba consiste en el controlador **RxByteIfaceDriver** por la *rx_iface*, y el monitor **TxByteIfaceMonitor** y el controlador **TxByteIfaceSinkDriver** por la *tx_iface*. Como en el banco de prueba del módulo: **signal_control**, el modelo del ADC es instanciado, y hay lógica para verificar las salidas al sensor y al ADC funcionan correctamente. Finalmente hay una clase **AdapterTbSequence** que extiende la secuencia **AppCommsTestsSequence**, para proveer habilidades específicas (customisations) a este banco de prueba. Unos ejemplos de estas habilidades son:

- Sustituya las tareas **go_to_state_*()**. La secuencia usa estas tareas para poner el DUT en el estado de inicialización apto por cada prueba, pero en este caso el DUT solo soporta el protocolo de nivel aplicación, por lo tanto estas transiciones

de estados tienen que ser simuladas.

- Sustituya la función **get_identify_reply_args()** para devolver las versiones simuladas de los bloques para verificar la respuesta al IDENTIFY.
- Sustituya las tareas **send_transaction()** y **verify_trans()** para quitar la cabecera de los bloques estándares. Esto es necesario porque la secuencia usa tramas enteras pero el DUT solo soporta el protocolo de nivel aplicación contenido en el campo de información de los bloques estándares tipo I.
- Sustituya la tarea **send_std_i_block()** para falsificar una pausa cuándo una transacción es enviada. Esto es necesario para verificar que el DUT detecta pausas durante los periodos t_{sync}, t_1, t_2, y t_{adc}.

Las pruebas ejecutadas son en la secuencia **AppCommsTestsSequence**. Unos ejemplos son:

- Envía el comando IDENTIFY y verifica la respuesta.
- Envía los comando SET_SIGNAL y AUTO_READ con pausas en los periodos t_{sync}, t_1, t_2, y t_{adc}.
- Envía los comando SET_SIGNAL y AUTO_READ, y después abandonar el proceso con el comando ABORT.
- Realiza una lectura manualmente mediante los comandos SET_SIGNAL y GET_RESULT.
- Envía un comando con un mágico inválido, o un error de CRC.
- Envía un comando y después un R(ACK) / R(NAK) con número de bloque igual a lo de la PICC para verificar que la PICC retransmite su última respuesta.

El informe de cobertura da un resultado de 96 % por el DUT. El módulo **adapter** solo, sin su submódulo **signal_control**, tiene un resultado de 97 %. Los casos que faltan cobertura son imposibles de ocurrir, por ejemplo el valor mágico de las respuestas es constante, así falta cobertura de toggle.

radiation_sensor_digital_top

Este módulo es el módulo más alto en el trabajo. Instancia el núcleo IP ISO/IEC 14443A, el módulo **adapter**, y unos sincronizadores. Cuadro 3.30 muestra las entradas y salidas del módulo.

Debido a que las entradas *rst_n_async* y *adc_conversion_complete_async* son asincrónicas, los módulo **active_low_reset_synchroniser** y **synchroniser** son instanciado para

sincronizar el flanco ascendente del reset, y ambos flancos de la señal *adc_conversion_complete_async* al dominio del reloj. Además debido a que la entrada *pause_n_async* también es asincrónica y que el reloj se detiene durante la pausa, el módulo **pause_latch_and_synchrhoniser** es instanciado para retener la pausa, y sincronizarla al dominio del reloj.

Nombre	Dirección	Descripción
AFE		
clk	Entrada	Reloj.
rst_n_async	Entrada	Reset, activo bajo, asincrónica.
power	Entrada	Especifica si la potencia recibida del campo electromagnético es suficiente.
pause_n_async	Entrada	Pausa, activa baja, asincrónica.
lm_out	Salida	Salida al modulador de carga.
afe_version	Entrada	Versión del AFE.
UID Control		
uid_variable	Entrada	Bits menos significativos del UID.
Sensor MOSFET de Radiación		
sens_config	Salida	Configuración del sensor.
sens_enable	Salida	Activa el sensor.
sens_read	Salida	Lee el sensor.
sens_version	Entrada	Versión del sensor.
ADC		
adc_enable	Salida	Activa el ADC.
adc_read	Salida	Lee el ADC.
adc_conversion_complete_async	Entrada	Indicador asincrónico que el ADC ha terminado la muestra.
adc_value	Entrada	Dato muestreado.
adc_version	Entrada	Versión del ADC.

Cuadro 3.30: Entradas y Salidas del módulo **radiation_sensor_digital_top**.

El UID elegido por este proyecto es un NUID simple de: 0xEFFEC70x, dónde los tres bits menos significativos son únicos por cada PICC. Estos bits son especificados con la entrada *uid_variable*. El núcleo IP es instanciado con parámetros que especifican el tamaño del UID, el número de bits variables y la parte constante del UID. El parámetro: *FDT_TIMING_ADJUST* del DUT, es asignado el valor dos, para compensar por los dos circos de retardo de la pausa en el módulo **pause_n_latch_and_synchroniser**.

Para verificar este módulo el banco de prueba es una mezcla de los bancos de pruebas por los módulos **iso14443a** y **adapter**. Los modelos del AFE y del ADC son instanciados

para generar el reloj, simular las pausas desde el PCD, y simular el comportamiento del ADC. Las salidas al sensor y al ADC son verificadas en la misma manera de en el banco de prueba del módulo **adapter**. El FDT es verificado en la misma manera de en el banco de prueba del módulo **iso14443a**. Finalmente la clase **RadSensDigTop_TbSequence** extiende la secuencia **AppCommsTestSequence**, para proveer habilidades específicas (customisations) a este banco de prueba:

- Sustituya la tarea: **send_transaction()** para configurar aleatoriamente los timings en el modelo del AFE antes de enviar cada trama.
- Sustituya las tareas de transmisión de los comandos SET_SIGNAL, AUTO_READ y ABORT para determinar el comportamiento esperado de las salidas al sensor y al ADC.

Todas las pruebas son ejecutadas cuatro veces con un UID distinto cada vez. Dentro de este lazo, cada prueba es ejecutada cien veces. Las pruebas ejecutadas son la combinación de las en los bancos de pruebas de los módulos **iso14443a** y **adapter**. Estas son las pruebas de inicialización, las de parte cuatro de la norma, y las del protocolo propietario de nivel aplicación.

El informe de cobertura da un resultado de 92 % por el DUT. El DUT mismo, sin considerar sus submódulos también tiene 92 % cobertura debido a que la salida *iso14443a_version* del núcleo IP ISO14443A es constante.

Síntesis y Place & Route

Síntesis y Place & Route (PnR) es el proceso de convertir el diseño digital implementado en HDL a un conjunto de máscaras (layout) que el fabricante puede usar para fabricar el ASIC. Síntesis es el proceso de convertir el HDL a una netlist de nivel compuertas, la herramienta usada para síntesis en este trabajo es Synopsys Design Compiler. PnR es el proceso de convertir esa netlist a un conjunto de máscaras, la herramienta usada para PnR es Synopsys IC Compiler 2. Estas herramientas son controladas con scripts TCL diseñados para este proyecto.

El proceso de fabricación utilizado en este trabajo es XFAB XH018, que es una tecnología CMOS de 180 nm. El fabricante (XFAB) provee un PDK (Process Design Kit) que contiene bibliotecas de celdas estándares compuestas por compuertas, registros y otros componentes necesarios para implementar un circuito integrado digital. Existen diferentes bibliotecas optimizadas para usos específicos. Las usadas en este proyecto son: D_CELLS_HD y D_CELLS_HDLL, las dos son celdas estándares de alta densidad de ruteo (HD). La biblioteca D_CELLS_HDLL es optimizada para bajo corriente de fuga (Low Leakage). Las herramientas eligen una celda HD o HDLL dependiendo en las optimizaciones elegidas y los constraints del diseño.

Minimizar el consumo de potencia estática es muy importante en este proyecto, porque la potencia es recibida inalámbricamente mediante el campo electromagnético, y durante las pausas el campo es reducido a menos de 5 % de su valor original. Por eso es necesario tener un capacitor en el AFE para mantener la tensión en el rail VDD, y el tamaño de ese capacitor depende en el consumo del ASIC durante las pausas. Debido a que el reloj es recuperado de la portadora del campo electromagnético, no hay reloj durante las pausas, lo que significa que no hay consumo de potencia dinámica tampoco. Así a minimizar el consumo de potencia estática implica que el capacitor en el AFE pueda ser más pequeño, reduciendo el área total del proyecto.

Estas celdas estándares HD y HDLL pueden funcionar con un rail de alimentación de 1,2 V o 1,8 V [28], este proyecto usa 1,8 V. Las celdas estándares incluyen el layout de cada celda y caracterizaciones de timing y potencia. El PDK también contiene las reglas del diseño para el proceso, por ejemplo el ancho y pitch mínimo por cada capa. El proceso XFAB XH018 se permite elegir la cantidad de capas de metal a usar, en este trabajo la

variación usada se llama “43” y tiene 6 capas de metal: MET1, MET2, MET3, MET4, METTP, METTPL.

Síntesis

El primer paso en síntesis con Design Compiler es construir una biblioteca milkyway, esto es un base de datos que Design Compiler usa para guardar el proyecto incluyendo información física de las celdas estándares, información de la tecnología, e información del diseño [24]. Además Design Compiler toma datos de otras bibliotecas que no están guardadas en la biblioteca milkyway: información de timing y potencia para cada celda, e información de capacitancia y resistencia para la extracción RC de las rutas. Después los archivos RTL del diseño son leídos, analizados y convertidos a un formato intermedio independiente de la tecnología.

Los constraints (restricciones) de timing son una parte muy importante del diseño digital, especifican detalles del timing de los relojes, las entradas y las salidas en el diseño. Esta información es necesaria para que las herramientas puedan realizar análisis de timing sobre todas las rutas y asegurar que el diseño va a funcionar con el reloj especificado. Debido a que no hay una implementación del AFE ni del ADC todavía para este proyecto, es necesario asumir algunas características sobre sus comportamientos, cuándo la implementación del proyecto entero esté finalizada estos constraints deberían ser revisados. Los constraints son:

- El reloj tiene una frecuencia de 13,5 MHz, con incertidumbre de 20 % para el análisis de Setup y 50 ps por análisis de Hold. Estos valores son basados en las recomendaciones de XFAB [27]. La norma ISO/IEC 14443-2 define la frecuencia de la portadora [11]: $f_c = 13{,}56\,\text{MHz} \pm 7\,\text{kHz}$. Una incertidumbre de 20 % es más que suficiente para cobrar la 7 kHz (0,05 %) de variación.
- Las entradas: *afe_version*, *adc_version*, y *sens_version* son constantes, por lo tanto no es necesario realizar análisis de timing sobre estas rutas. Para especificar esto las rutas son cortadas mediante el constraint set_false_path.
- La entrada: *uid_variable* también es constante, y las rutas son cortadas. El nombre indica que es la parte del UID que es única para cada PICC. Esta señal debería ser fija mientras el diseño está fuera de reset.
- La entrada: *rst_n_async* es asincrónica, el diseño contiene un sincronizador de reset para sincronizar el flanco ascendente, por lo tanto no es necesario realizar análisis de timing sobre esta ruta.

- Las salidas al sensor y al ADC: *sens_config*, *sens_enable*, *sens_read*, *adc_enable* y *adc_read*, son asincrónicas también. Si el sensor o el ADC necesita recibir esas señales de forma sincrónica estos bloques deben sincronizar esas señales al dominio del reloj internamente.
- Las entradas desde el ADC: *adc_value* y *adc_conversion_complete*, son asíncronas y no es necesario realizar análisis de timing sobre estas rutas. En el RTL *adc_conversion_complete* pasa por un sincronizador para prevenir metaestabilidad. Además un requisito del ADC es que *adc_value* es estable antes del pulso en *adc_conversion_complete*. Esto significa que cuándo el pulso ocurre en la versión sincronizada de *adc_conversion_complete*, es seguro usar *adc_value* sin arriesgar metaestabilidad. En realidad estas señales van a ser sincrónicas, pero sin conocer como son ruteados y la relación de latencia entre el reloj llegando a este bloque y al ADC, es imposible escribir constraints más precisos.
- La entrada *power* desde el AFE es asincrónica. Un requisito del diseño del AFE es que esa señal debe ser constante durante el envío de una respuesta al PCD, el único momento en que esa entrada es usada, así que no es necesario sincronizar esta señal.
- La entrada *pause_n_async* y la salida *lm_out* son asincrónicas también, *pause_n_async* pasa por un sincronizador en el RTL. A diferencia de las otras señales asincrónicas, estas rutas no son cortadas. Al cortar una ruta significa que las herramientas no intentan optimizar el tiempo de propagación interno de la ruta. Debido a que el requisito del FDT es estricto, es deseado minimizar el retardo de propagación en estas rutas. Para cumplir esto el constraint set_max_delay es usado, especificando un retardo de propagación máximo de 5 ns.
- Debe ser asumido que todas las entradas son manejadas por un inversor de tamaño mínimo, y que todas las salidas tienen carga de 0,4 pF lo que es equivalente de aproximadamente 2 mm de ruteo en la tecnología XH018. Estos valores son especificados en la documentación del PDK [27].

En un circuito digital el tiempo de propagación de una compuerta depende principalmente en tres factores conocido como el PVT [2]:

- Proceso: Cada chip CMOS fabricado posee diferencias debido a la variabilidad del proceso de fabricación.
- Voltaje: La tensión de alimentación V_{DD}.
- Temperatura: La temperatura del chip.

En análisis de timing, un corner se refiere a ciertas condiciones PVT de operación. El PDK provee bibliotecas con caracterizaciones de timing para varios corners del PVT, que pueden ser usados para verificar el timing de un diseño. En este trabajo se utiliza el análisis BCWC (best case, worst case). El worst case corner genera las condiciones que causan el retardo máximo, lo cual ocurre con el proceso más lento, la tensión más baja y la temperatura más alta. El best case corner genera las condiciones que causan el retardo mínimo, lo cual ocurre con el proceso más rápido, la tensión más alta y la temperatura más baja. Las herramientas usan el worst case corner por análisis de Setup, y el best case corner por análisis de Hold. Los corners usados son:

Best Case: Rápido, 1,98V, -40°C.
Worst Case: Lento, 1,62V, 125°C.

El diseñador del AFE debe asegurarse que su diseño siempre puede proveer un V_{DD} entre esos extremos. La temperatura del chip debería mantenerse siempre dentro de los límites especificados, ya que este ASIC está diseñado para uso en contacto directo con humanos.

El consumo de potencia de un circuito digital puede ser dividido en tres categorías: estático, interno y dinámico. La potencia interna es consumida dentro de la celda cada vez que una entrada de la celda cambia, y la potencia dinámica es consumida dentro de la celda cada vez que la salida cambia. Para realizar análisis y optimización de potencia es necesario conocer a qué frecuencia cambian las señales. Esta información puede ser especificada mediante un archivo SAIF, lo que es producido mediante una simulación del diseño. El archivo SAIF contiene la duración de la simulación y tres parámetros principales por cada señal:

T0: Duración de la señal en '0'
T1: Duración de la señal en '1'
TC: Cantidad de veces que la señal cambió su estado.

Debido a que el archivo SAIF no contiene información sobre cuándo ocurren los cambios, las herramientas solo pueden obtener una frecuencia promedio por cada señal, lo que implica que la anàlisis de potencia produce una estimación de potencia promedio, y no incluye información sobre el consumo pico. En este trabajo el archivo SAIF fue generado con una simulación de nivel RTL, lo que significa que solo contiene información sobre las señales nombradas en el RTL, y no señales intermedias ni las señales dentro de las celdas. El otro problema con archivos SAIF es que su precisión depende del banco de prueba. Si el banco de prueba no es representativo del uso real la estimación de

potencia no sería precisa. El banco de prueba que genera el SAIF para este proyecto es: **generate_top_saif_tb**, es una copia modificada del banco de prueba del módulo **radiation_sensor_digital_top**. Este banco de prueba simplemente realiza el proceso de inicialización definida en la norma ISO/IEC 14443 y después toma cinco muestras del sensor mediante los comandos AUTO_READ y GET_RESULT, lo que debería ser representativa del uso real. Este archivo SAIF es leído en Design Compiler antes de la compilación del diseño.

Compilación es el proceso que convierte el diseño a un netlist de nivel de compuertas. Design Compiler puede realizar este proceso controlado por requerimientos de área, timing y potencia, que significa que intenta producir un netlist con área mínima, potencia mínima y que cumple con timing adecuado. Una optimización que se usa para reducir el consumo de potencia es Clock Gating, una técnica dónde el reloj de un registro es desactivado cuándo el valor del registro no cambia de estado. Esta optimización reduce la potencia interna del registro porqué evita el cambio de estado del reloj dentro de la celda. La Figura 4.1 muestra esta técnica. Esta optimización puede impactar el timing de las rutas involucradas, y el área del diseño. Sin embargo puede reducir el área del diseño, porque puede reemplazar un multiplexor en la entrada de cada registro en un banco, con un solo Clock Gate compartido entre todo el banco. La estimación de consumo de potencia reportado al final de síntesis es reducido 63 % desde 454,0 µW a 169,5 µW, y el área total de las celdas es reducido 2 % desde 56 307 μm^2 a 55 348 μm^2 al aplicar el método de Clock Gating.

Al final de la síntesis se obtiene el netlist de nivel compuertas el cual es procesado con IC Compiler 2 para realizar el PnR. A partir de los reportes de síntesis se obtienen los siguientes resultados: el área total de las celdas es 55 348 μm^2, y la estimación de consumo de potencia es 169,5 µW.

Después del script de síntesis se usa la herramienta Formality de Synopsys para verificar formalmente la equivalencia entre la lógica representada en el RTL y en el netlist de post síntesis. Este proceso es realizado mediante métodos formales y no requiere vectores de simulación [22, traducción mía]. Design Compiler produce un archivo SVF que contiene información de orientación sobre los cambios realizados en el proceso de síntesis, por ejemplo: el renombramiento de registros, o el agregado de celdas de Clock Gating. En el proceso conocido como Matching, Formality usa este archivo para encontrar puntos de comparación comunes entre los dos diseños. Un punto de comparación puede ser una salida del diseño o un registro. Desde un punto de comparación Formality determina un

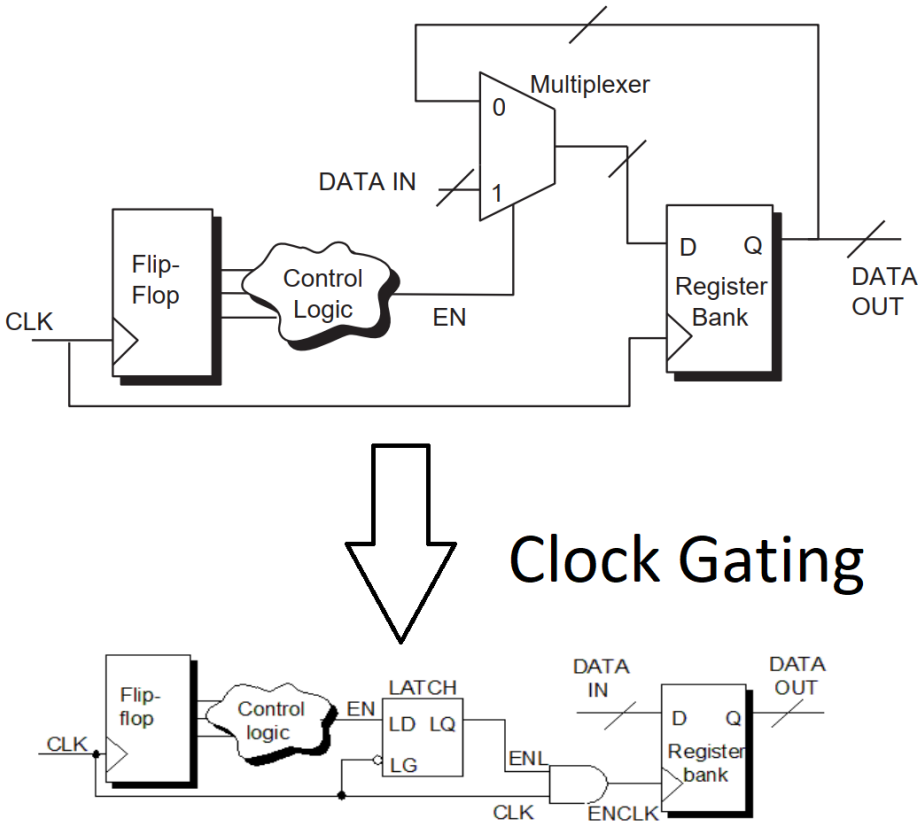

Figura 4.1: Optimización de potencia interna mediante la técnica de Clock Gating [25, modificaciones mías].

cono lógico moviendo atrás que incluye toda la lógica que conecta a ese punto, hasta llegar a otro punto de comparación o una entrada del diseño. La Figura 4.2 muestra un cono lógico. Después en el proceso de verificación Formality compara formalmente los conos de los dos diseños por cada punto de comparación.

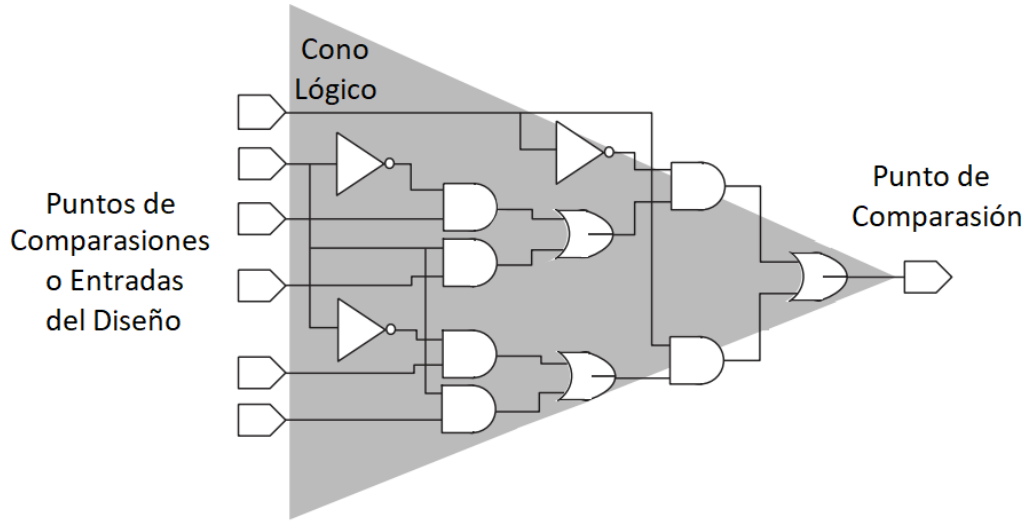

Figura 4.2: Un cono lógico [22, modificaciones mías].

Preparación de librerías

IC Compiler 2 requiere las CLIBs (Cell Libraries) en el formato NDM (New Data Model). Al tiempo de realizar este trabajo XFAB no tenía disponible bibliotecas en el formato NDM por el proceso XH018. Por lo tanto es necesario convertir las bibliotecas provistas al formato NDM. Este proceso es realizado mediante la herramienta Library Manager de Synopsys. Cada celda en un NDM contiene información de timing, potencia, lógica, el layout físico y las reglas de la tecnología. En las CLIBs provistas por XFAB, esa información es contenido en bibliotecas de varios formatos:

- TF: Las reglas de la tecnología.
- TLU: Información sobre parásitos.
- GDS: El layout de las celdas.
- LEF: Información sobre el layout de las pines y obstrucciones, e información sobre los propiedades antena de las celdas.
- DB: Información sobre timing, potencia y la lógica de las celdas.

Un script TCL fue diseñado para cargar todas esas bibliotecas y combinar la información a tres CLIBs en el formato NDM para uso con IC Compiler 2. Las tres CLIBs son:

- tech_only.ndm: Contiene información sobre la tecnología, esta incluye: las reglas

de la tecnología, los parasíticos y las direcciones del ruteo en cada capa de metal (Horizontal o Vertical).

- d_cells_hd.ndm: Contiene información sobre las celdas estándares de D_CELLS_HD.
- d_cells_hdll.ndm: Contiene información sobre las celdas estándares de D_CELLS_HDLL.

El proceso para producir las CLIBs por las celdas estándares es: leer los archivos: TF, GDS, LEF y DB, y arreglar las discrepancias entre las descripciones de las celdas en cada formato. Por ejemplo: Los archivos GDS usan los nombres: vdd! y gnd! para los rails de alimentación, pero los archivos LEF usan los nombres: vdd y gnd.

Design Planning

Design Planning (DP) es un paso inicial de la implementación de un IC. En este trabajo el script de DP es principalmente usado para construir un floorplan (la forma y las dimensiones físicas del bloque), elegir ubicaciones por los pines que conectan a las otros bloques (el AFE, el Sensor MOSFET de radiación, y el ADC), y armar los anillos, mallas y rails de alimentación.

El floorplan posee las dimensiones físicas del diseño, en este caso del circuito digital. Luego los diferentes bloques del proyecto serán instanciados en un top cell el cual requerirá el ruteo manual entre los mismos, los pines de entrada/salida y los pads de prueba. El área del floorplan por el circuito digital debe ser al menos suficiente para contener las celdas, y los anillos de potencia, pero frecuentemente es necesario dejar más espacio en el núcleo para evitar congestión de ruteo. El área de las celdas es calculada desde el netlist producido en síntesis, para este proyecto esto es: 55 347 μm^2. El núcleo del floorplan es el área que contiene las celdas, y la tasa de utilización del núcleo es la proporción del núcleo que contiene celdas útiles. Usando un proceso iterativo se determinó que la tasa de utilización del núcleo máxima para que el diseño cumple con timing es 75 %. Así el núcleo debería tener un área de 73 796 μm^2, lo que se corresponde con una región cuadrada de lado 272 µm. La documentación del PDK especifica que los anillos de alimentación deben tener un ancho de 5 µm, con separación de 2,5 µm [27]. Hay un anillo de VDD y uno de GND, así se necesita 12,5 µm alrededor del núcleo. Esto significa que el floorplan tiene dimensiones de 297 µm por 297 µm, y un área total de 88 209 μm^2. Debido a la forma de las celdas, la herramienta eligió las dimensiones: 295,68 µm por 294,0 µm dando un área de 86 930 μm^2.

Este bloque está diseñado para ser conectado con los otros bloques del diseño internamente al IC, por eso es necesario agregar un pin por cada entrada y salida alrededor del núcleo. Esas pines son agregadas en las capas de metal: MET3 y MET4. Se ha considerado los siguientes criterios por los pines de interfaz con los bloques circuitales periféricos:

- Señales desde / hacia el AFE están en el borde superior, en MET4.
- Señales hacia el Sensor MOSFET de Radiación están en la mitad inferior del borde derecho, en MET3.
- Señales desde / hacia el ADC están en la mitad superior del borde derecho, en MET3.
- Las entradas del UID están juntas pero no son constrained a un borde en particular. La herramienta eligió el borde izquierdo en MET3.

Las celdas estándares provistas en el PDK son diseñadas para tener la misma altura, con un rail de alimentación en el borde superior y un rail de tierra en el borde inferior, los dos en la primera capa de metal: MET1. La Figura 4.3 muestra un ejemplo de una celda XOR con los rails de alimentación nombrados. Las celdas son fijadas en filas cruzando el núcleo del diseño, cada fila adicional se instancia de forma espejada para que compartan los rails. Es necesario conectar estos rails a los anillos y mallas de VDD y GND, mientras minimizando la caída de tensión debido a la resistencia parásita de las metalizaciones (caída IR). Los anillos rodean el núcleo, las mallas consisten en stripes (tiras) que cruzan el núcleo y los rails conectan las celdas con las mallas y anillos. La documentación del PDK [27] específica que los anillos y mallas deben colocarse en las capas de metal superiores: METTP y METTPL, con ancho mínimo de 5 um, y separación máximo entre mallas de 350 um. METTP debe ser usado por las rutas horizontales y METTPL por las rutas verticales. Debido a que este bloque tiene dimensiones menores que 350 um por 350 um, las mallas de VDD y GND solo consisten en un stripe vertical y uno horizontal cruzando el centro del núcleo por las dos mallas. La Figura 4.4 muestra el floorplan generado con las rutas de potencia y los pines de las entradas y salidas.

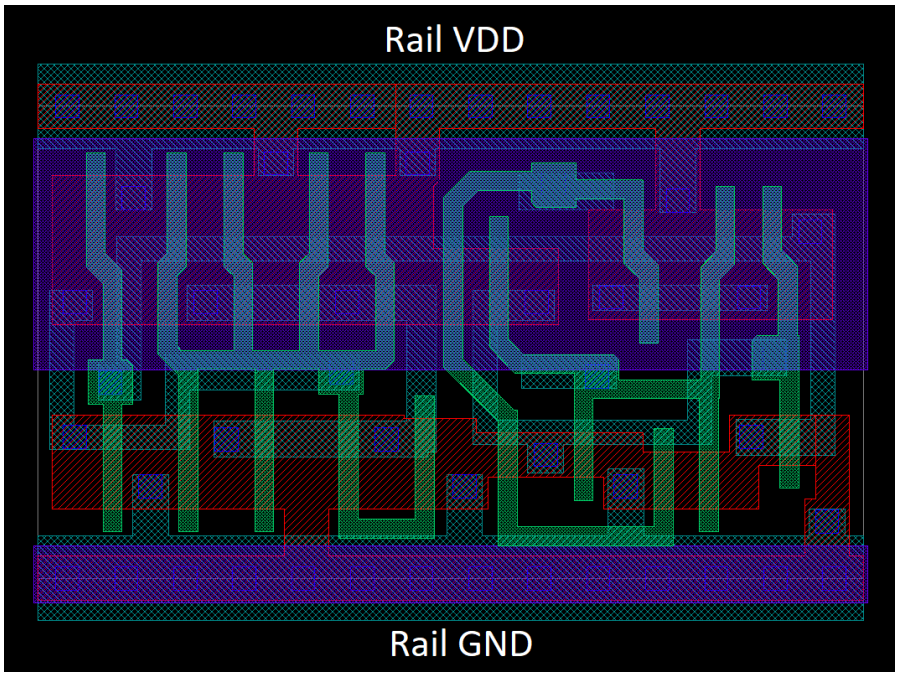

Figura 4.3: Una celda XOR de la biblioteca D_CELLS_HD.

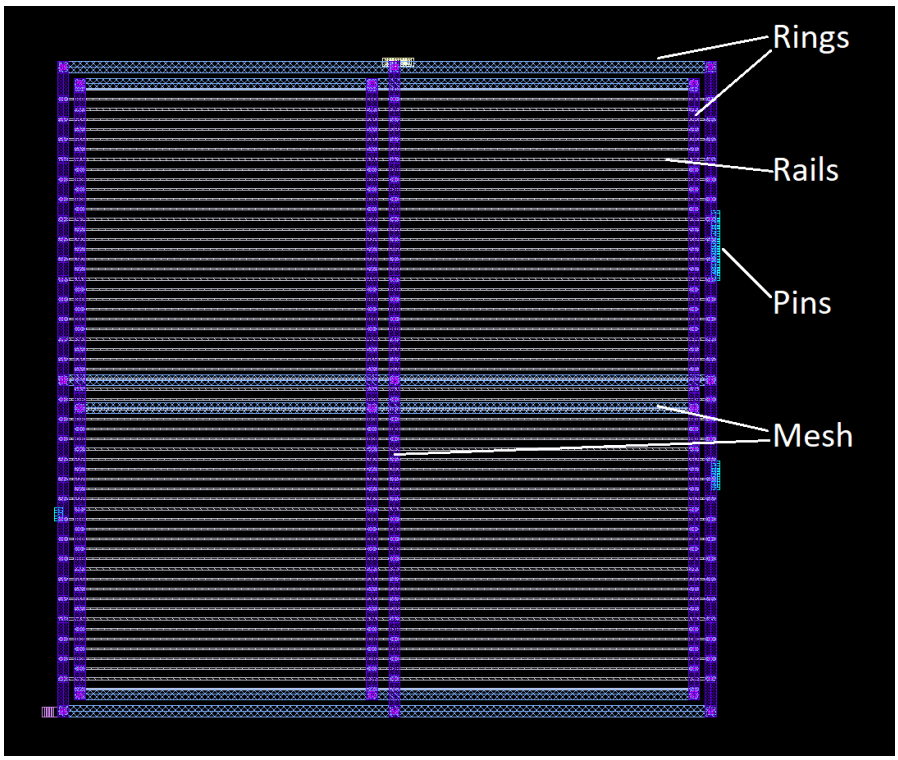

Figura 4.4: El floorplan del diseño con las rutas de potencia y los pines IO.

Para poder conectar VDD y GND a la fuente de potencia en el AFE, es necesario agregar terminales. Debido a que los otros pines que conectan al AFE están en el borde superior, los terminales son colocados en el medio del borde superior de los anillos. La documentación del PDK [27] específica que la caída de tensión máxima entre los anillos y los stripes debe ser menor que 100 mV. Esto es verificado con las herramientas realizando un análisis de caída IR. Asumiendo un consumo de potencia máxima de 400 µW lo que es casi doble la estimación de potencia final, los resultados de este análisis son: por VDD la caída IR máximo es 34 µV, y por GND es 38 µV. Estos resultados son aproximadamente 0,036 % de la máxima especificada en el PDK. La Figura 4.5 muestra un mapa de calor por el análisis de VDD.

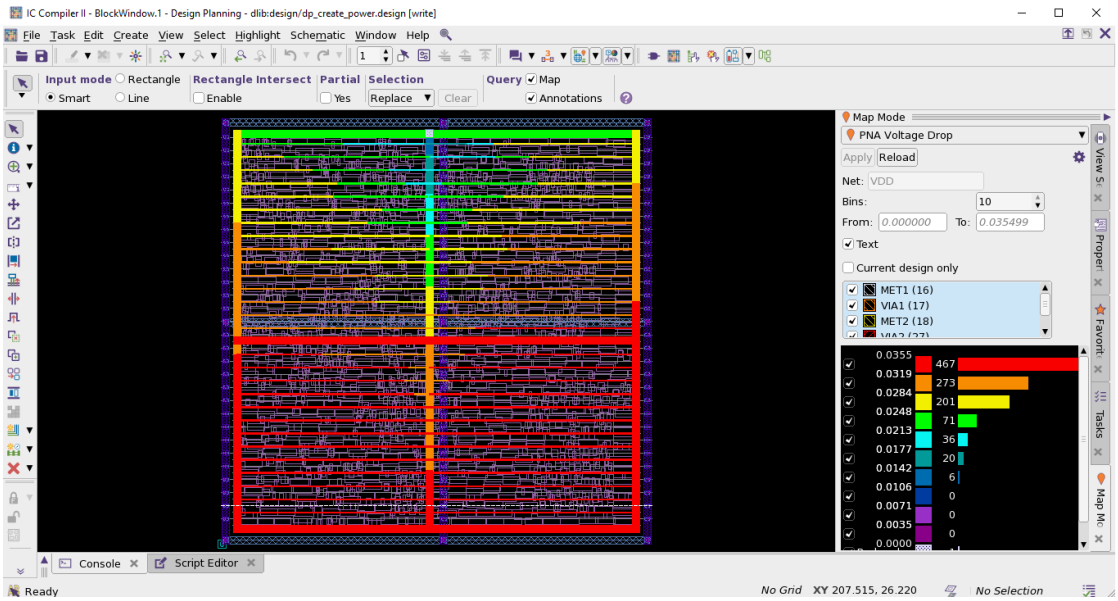

Figura 4.5: Mapa de calor del análisis de caída IR sobre la ruta de alimentación VDD.

Place & Route

El script de PnR comienza con la importación de los resultados de los script de Design Planning y de síntesis. Después las celdas son colocadas en el núcleo, este proceso es iterativo con múltiples de pasos para optimizar la ubicación de las celdas. Primero las celdas son colocadas de forma aproximada. Después las ubicaciones son ajustadas para optimizar por timing, área, consumo de potencia, y congestión. Después de cada paso los errores DRC son arreglados.

La próxima operación es CTS (Clock Tree Synthesis), el proceso de construir y rutear árboles de relojes balanceados mediante buffers. Es necesario usar un árbol porqué el fanout del net del reloj es grande, ya que tiene que conectar a cada registro en el dominio del reloj. Tiene que ser balanceado para minimizar el skew en el reloj entre cada registro. Los reportes especifican que el skew máximo después de CTS es 1,59 ns en el análisis de Setup, y 1,77 ns en el análisis de Hold, los que son respectivamente 2,1 % y 2,4 % del periodo del reloj.

Durante el ruteo de los nets en el diseño es necesario tener cuidado sobre el largo de cada segmento de metal en la ruta, para evitar efectos de antenas. Synopsys define el problema de antenas como:

> En la fabricación de circuitos integrados, el óxido de gate puede ser dañado fácilmente por descargas electrostáticas. La carga estática que es acumulada en las rutas de metal durante el proceso de metalización puede dañar el dispositivo o terminar con un fallo del chip entero. [23, traducción mía]

El PDK contiene reglas de antenas que especifican los ratios máximos entre las áreas de los segmentos de metal y de los gates de los transistores de que se conectan. Si un diseño cumple con esas reglas, no debería tener problemas de antena. Una forma de arreglar estos problemas es cortar segmentos de metal grandes con un segmento en una capa de metal superior cercana al gate. Otra forma es colocar diodos cercana a los gates. Cuándo las reglas de antena son agregadas, IC Compiler 2 intenta rutear el diseño evitando problemas de antenas, si siguen violaciones de antenas las celdas de diodos presentes en el PDK pueden ser agregadas.

La operación de ruteo también es un proceso iterativo, con cada paso intentando arreglar violaciones DRC o de timing del paso anterior. Al final de este proceso el slack de Setup es 2,6 ns, y el slack de Hold es 0,0 ns. Los reportes indican que no hay violaciones DRC, ni problemas de antena, tampoco hay rutas abiertas o cortocircuitos.

Después del ruteo vienen dos acciones conocidas juntas como Finishing. La primera acción es agregar vías redundantes dónde hay espacio, para mejorar las conexiones entre las capas, esto se hace con un script del PDK. Después se agregan celdas de relleno (filler cells) en los espacios vacíos, esto es para asegurar que los rails de alimentación están conectados a lo largo de la fila. Hay dos tipos de celdas de relleno: con capacitores de desacople, y sin capacitores de desacople. Las celdas con capacitores son usadas para ayudar a mantener las rails de alimentación de forma local durante periodos de alto

consumo de corriente. Esas celdas son agregadas primero, y las que generan errores DRC son quitadas. Después, los sitios que se quedan vacíos son llenados con las celdas sin capacitores. Finalmente el proceso de ruteo es ejecutado de nuevo en caso que haya que arreglar violaciones DRC. La Figura 4.6 muestra el layout después del finishing.

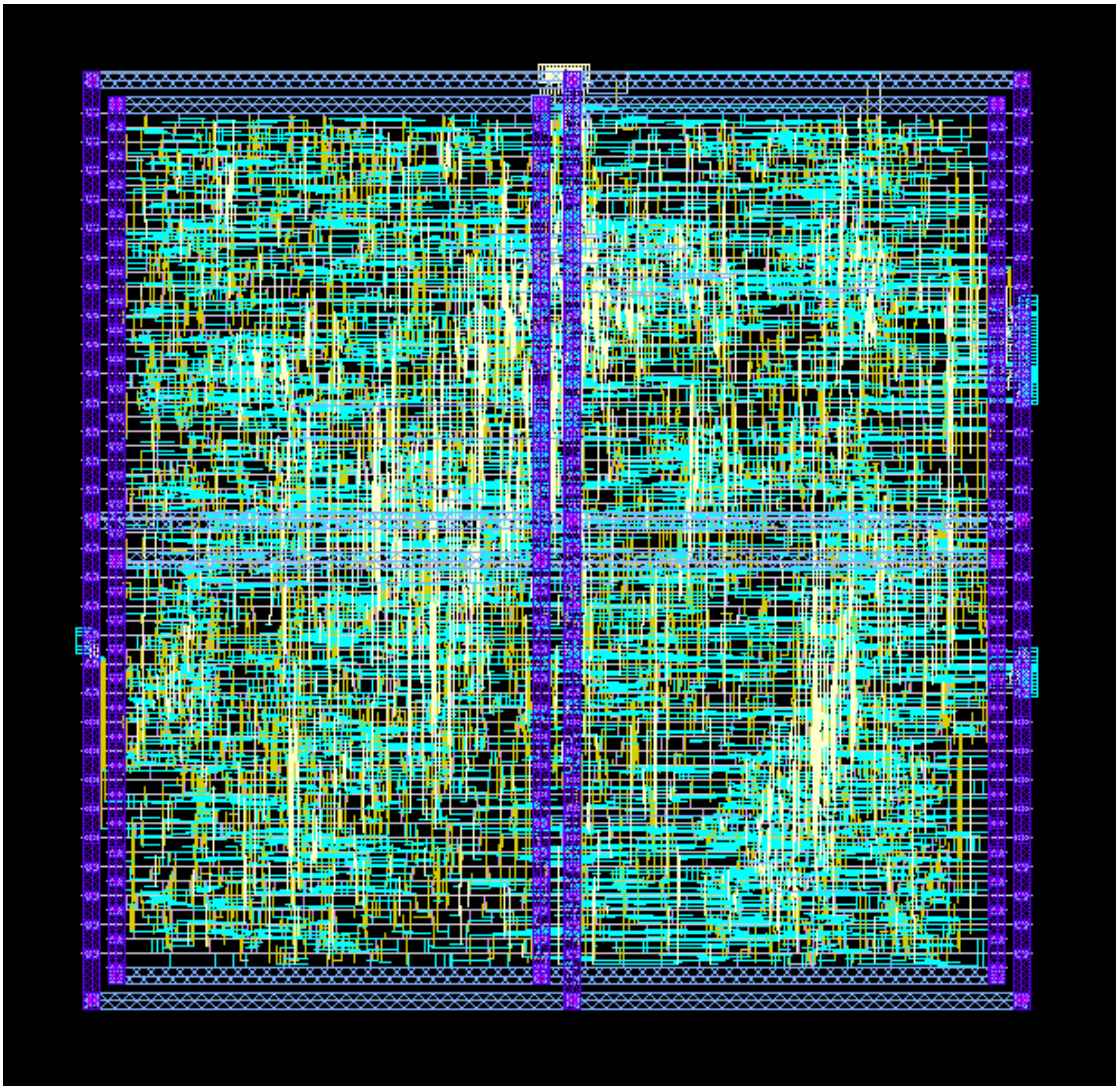

Figura 4.6: El layout del diseño después del finishing.

La próxima etapa es ICV In Design, esa etapa ejecuta unas acciones mediante la herramienta Synopsys IC Validator. La primera acción es verificar que no hay ninguna violación DRC en el diseño mediante un script provisto en el PDK, y si hay violaciones, intenta corregirlas. El próximo paso es el proceso de Metal Fill. Para reducir errores en

el proceso de fabricación es necesario asegurar que la densidad de metal de cada capa está dentro de unos límites definidos en las reglas de diseño. Si la densidad de metal en una capa es demasiado baja, IC Validator puede agregar metal extra en los espacios libres. Después del metal fill, las pruebas DRC son ejecutadas de nuevo, pero esta vez activando una prueba opcional para verificar la densidad de metal.

El siguiente paso es escribir los resultados del PnR. El netlist final es exportado en dos formatos, uno para uso en LVS (Layout Vs Schematic) y uno para uso en el script de verificación de equivalencia con Formality. El layout final es guardado en el formato GDS II.

La estimación de consumo de potencia final es 256,0 µW, compuesta de: 163,5 µW (64,0 %) potencia interna, 91,6 µW (35,8 %) potencia dinámica, y 623,8 nW (0,2 %) potencia estática. Análisis de timing indica que el mínimo Setup slack es 1,59 ns y el mínimo Hold slack es 0,02 ns. La Figura 4.7 muestra la distribución de slack de Setup y slack de Hold. Los dos rutas con menor Setup slack son 1) La ruta entre el registro de *lm_out* en el módulo **tx**, y la salida *lm_out* de este diseño. 2) La ruta entre la entrada *pause_n_async* de este diseño y el primer registro en el módulo **pause_n_latch_and_synchroniser**. Estas rutas tienen slack bajo por los constraints: "set_max_delay 5.0", los cuales están presentes para minimizar los retardos en la ruta del FDT.

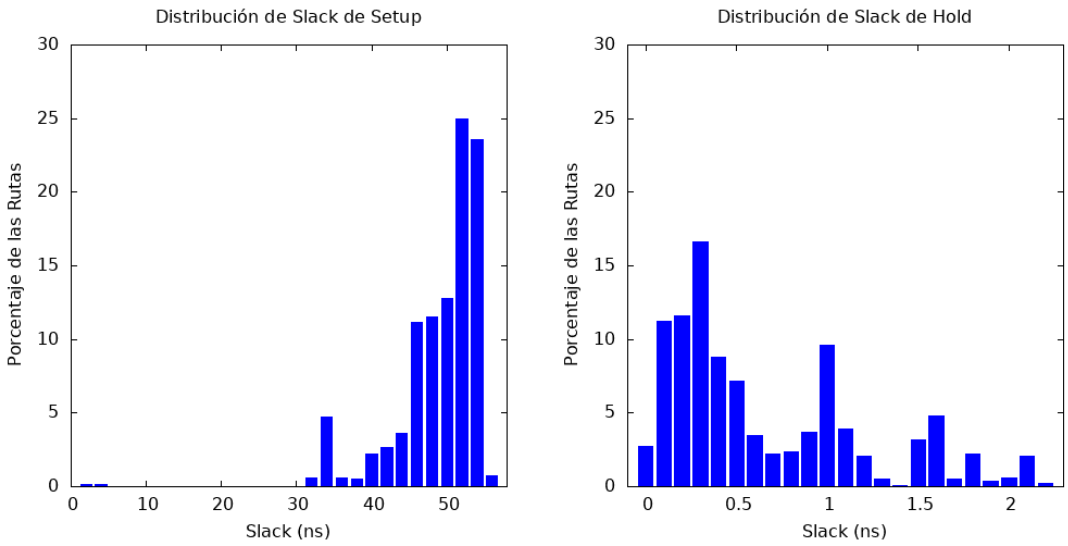

Figura 4.7: Distribución de slack de Setup y de Hold en el diseño final.

Finalmente la herramienta Formality es usada de nuevo para verificar formalmente la equivalencia de la lógica entre el RTL y el netlist de post implementación. Este proceso

es equivalente del proceso hecho después de síntesis, salvo por que se usa los archivos SVF de coordinación generados en Design Planning y en Place & Route, en adición a lo de síntesis.

LVS / DRC

En el script de Place & Route el diseño fue analizado para verificar violaciones DRC, este paso es repetido en este punto pero usando el archivo GDS II e instanciando en IC Validator directamente. La ventaja de esto es que verifica que no haya errores DRC introducidos en el proceso de escribir el GDS II.

LVS (Layout Vs Schematic) es un análisis que compara la netlist final del diseño con el GDS II. La primera parte de esta operación es convertir el netlist de nivel compuertas escrito al final del script de PnR a un netlist de nivel SPICE mediante la herramienta nettran. Después IC Validator es ejecutado con un script provisto en el PDK para extraer los dispositivos y nets del layout en el archivo GDS II. Ese resultado es comparado con el netlist de nivel SPICE para confirmar que se corresponden.

Conclusión

En este trabajo un circuito digital fue implementado, verificado mediante simulaciones y preparado para fabricación con el proceso XFAB XH018. El circuito implementado permite la lectura de un sensor de forma inalámbrica mediante un protocolo propietario implementado encima del estándar ISO/IEC 14443A. En este caso el sensor es un dosímetro MOSFET de radiación ionizante para aplicación al control en tratamientos de radioterapia. Sin embargo el circuito digital diseñado no está limitado a esta aplicación y puede trabajar con diferentes sensores para otras aplicaciones sin necesidad de modificación alguna. Los bloques externos necesarios para este uso son: un AFE (Analogue Front End) conforme con ISO/IEC 14443-2A, un ADC de 16 bits, y el sensor propiamente dicho. La implementación del estándar ISO/IEC 14443A fue diseñado como un núcleo IP genérico, cuyo código RTL y los scripts de síntesis y PnR están disponibles en GitHUB de forma abierta bajo la licencia GNU v3.0. Esto significa que puede ser usado en otros proyectos académicos o comerciales, siempre que cumplan con las condiciones de la licencia.

En este proyecto, fue clave que el núcleo IP ISO/IEC 14443A sea genérico, para que funcione con AFEs distintas. Para ello las propiedades más importantes son el comportamiento del reloj y la pausa detectada. Por lo tanto:

- El núcleo IP incluye un módulo asincrónico para la retención de pausas en el caso que el AFE no pueda garantizar que haya un flanco ascendente del reloj durante la pausa.
- El módulo **sequence_decode** fue diseñado para decodificar las tramas correctamente aún cuando el reloj está detenido, hasta un máximo de 59 ciclos, durante las pausas, incluyendo el caso de un reloj continuo.
- El núcleo IP lleva un parámetro *FDT_TIMING_ADJUST* que permite el ajuste del temporizador en el módulo **FDT** para compensar por los retrasos externos en la ruta del FDT.

Uno de los objetivos de esta tesis fue la posibilidad de tomar muestras de múltiples de PICCs de forma sincronizada. Para cumplir con este objetivo se propone una solución mediante el protocolo a nivel de aplicación. Este protocolo utiliza dos características del sistema: 1) Todas las PICCs extraen el reloj de la misma portadora y por lo tanto

sus relojes son sincrónicos, 2) todas las PICC reciben las pausas en el mismo instante. Además se implementa una lectura del ADC y del sensor de manera automatizada y parametrizable. Aprovechando estas características es posible realizar la lectura sincrónica de todos los sensores que estén en el rango del PCD.

Todo el RTL implementado es verificado con una serie de bancos de pruebas exhaustivas, con más de cien horas de simulaciones. Los bancos de pruebas contienen 187 aserciones que son ejecutados más de cien mil millones de veces en total, sin fallar siquiera una vez. Cada simulación genera un informe de cobertura y en todos los casos se observan resultados favorables. Además el modelo del AFE implementado permite la configuración de varias de sus propiedades para simular el comportamiento de diferentes AFEs. Finalmente el RTL se verifica formalmente, mediante una cadena de equivalencias: el RTL es equivalente al netlist post implementación, y a su vez este netlist es equivalente al layout del diseño.

Los reportes finales indican que:

- El diseño contiene 595 registros (568 de esos son gated), 2017 compuertas combinacionales y 2706 nets.
- El consumo de potencia promedio estimado es 256 µW, compuesto de: 163 µW potencia interna, 92 µW potencia dinámica, y 624 µW potencia estática. Esta estimación está basada en una simulación de la inicialización de la PICC y cinco muestras del sensor.
- El área total del diseño es 86 930 μm^2 = 0,087 mm^2 (295,68 µm, 294,00 µm).
- No hay slack negativo de Setup. El menor slack de Setup es 1,59 ns y ocurre en la ruta entre el registro de *lm_out* en el módulo **tx**, y la salida *lm_out* del diseño entero. Esa ruta tiene un slack bajo por el constraint: "set_max_delay 5.0 -to [get_ports lm_out]", el cual está presente para minimizar los retardos en la ruta del FDT. El mínimo slack de setup corresponde a una ruta sin un constraint de set_max_delay tiene un valor de 32,26 ns.
- No hay slack negativo de hold. El slack mínimo es 0,02 ns.
- El diseño pasa sin errores las verificaciones de DRC y LVS.
- La lógica en el netlist que se obtiene post-implementación es equivalente a la del RTL.

Todos estos resultados dan una excelente confianza en el diseño. Debido a las suposiciones asumidas sobre otros bloques aún no diseñados, es altamente recomendable revisar el correcto funcionamiento del sistema completo antes de su fabricación, para verificar la

correcta interoperabilidad entre todos los bloques.

Requisitos para los otros bloques

Para asegurar el funcionamiento correcto de un ASIC que incluya el diseño elaborado en esta tesis, los otros bloques presentes deben cumplir con algunos requisitos. El sensor y el ADC deben operar de forma que se puede tomar una muestra mediante la operación mostrado en la Figura 3.26. Los valores de t_1 y t_2 requeridos para tomar una muestra correcta deben ser menor que 33 554 431 ciclos $\approx$ 2,47 s. El AFE debe cumplir con los siguientes requisitos:

- Debe ser conforme con la norma ISO/IEC 14443-2A.
- Debe recuperar el reloj desde la portadora transmitida por el PCD. El reloj recuperado debe ser monótono y no tener glitches. Si se detiene durante las pausas debe hacerlo por un tiempo menor a 59 ciclos, aunque se recomienda que el tiempo sea menor a 56 ciclos, de esta manera el sistema tolera hasta ± 3 ciclos de jitter en la detección de la pausa.
- Para minimizar el retardo en la ruta del FDT, el reloj debe comenzar de nuevo lo antes posible y el fin de la pausa también debe ser detectado lo antes posible. Además debe minimizarse el retardo de propagación entre la salida *lm_out* de este diseño y el modulador de carga.
- Cuándo la PICC entra en un campo electromagnético, la señal de reset *rst_n_async* debe estar '0' hasta que el rail de alimentación VDD y la entrada *uid_variable* de este diseño están estables.
- La salida *afe_version[3:0]* debe ser constante.

El diseñador del AFE debe considerar las restricciones impuestas en la forma de la pausa especificadas en la norma ISO/IEC 14443-2A, y asegurarse que su diseño cumple con esos requisitos por cualquiera forma de la pausa válida.

Recomendaciones para Trabajos Futuros

A continuación se listan trabajos que quedan fuera del alcance de esta tesis pero serían necesarios para la fabricación del chip considerado en el proyecto marco:

- Realizar una mejor estimación del consumo de potencia, incluyendo consumo pico. Esto es importante al momento de diseñar el AFE.

- Realizar simulaciones de nivel compuerta para verificar el diseño post síntesis y post PnR.
- Implementar el AFE, el ADC y el sensor en la tecnología XH018 de X-Fab.
- Agregar puntos de prueba al diseño para permitir realizar mediciones e inyectar señales de forma directa.
- Fabricar y medir el IC.

Además existen algunas características extras que podrían ser implementadas en el núcleo IP ISO/IEC 14443 para que sea adecuado para uso en un rango más amplio de proyectos:

- Agregar soporte para tags de tipo B.
- Implementar las partes opcionales de la norma: soporte para las otras tasas de bits, NADs, bloques estándares encadenados, los bloques estándares S(PARAMETERS) y S(WTX), y bloques aumentados [11][13].

Apéndice A:

Repositorio de Código Fuente

Todo el código RTL y los scripts usados en este trabajo están disponibles en un repositorio de GitHUB: https://github.com/andrewparlane/fiuba_thesis. Todo el código y los scripts son disponible de forma abierto bajo la licencia GNU v3.0. Se puede clonar este repositorio con los comandos:

```
git clone https://github.com/andrewparlane/fiuba_thesis.git
git submodule init
git submodule update
```

La estructura del repositorio es:

```
fiuba_thesis
├── hdl
│   ├── components
│   │   └── iso_iec_14443A ..... Git Submodule del núcleo IP ISO/IEC 14443A
│   ├── rtl ..... RTL de la aplicación
│   ├── synth_pnr ..... Scripts de síntesis y PnR
│   └── verification
│       ├── bfms ..... Clases y paquetes de verificación de la aplicación
│       ├── tb ..... Los bancos de pruebas por los módulos de la aplicación
│       └── vcs ..... Contiene un Makefile para ejecutar las simulaciones
└── thesis ..... Fuente de este documento
```

La carpeta *fiuba_thesis/hdl/components* contiene un Git Submodule por el repositorio del núcleo IP ISO/IEC 14443A: https://github.com/andrewparlane/iso_iec_14443A. También todo el código y los scripts son disponible de forma abierto bajo la licencia GNU v3.0. La estructura de este repositorio es:

```
iso_iec_14443A
├── rtl ..... RTL del núcleo IP
└── verification
    ├── bfms ..... Clases y paquetes de verificación del núcleo IP
    ├── tb ..... Los bancos de pruebas por los módulos del núcleo IP
    └── vcs ..... Contiene un Makefile para ejecutar las simulaciones
```

Apéndice B:

Definiciones del protocolo propietario

Este apéndice contiene un header del lenguaje de programación C con definiciones y estructuras por el protocolo propietario de la aplicación.

```
// La mágica y la versión del protocolo
#define PROTOCOL_MAGIC                  0xF100BA00
#define PROTOCOL_VERSION                1

// Las mascaras de bits de las salidas al sensor / ADC
#define SIGNALS_MASK_SENS_CONFIG        0xE0
#define SIGNALS_MASK_SENS_ENABLE        0x10
#define SIGNALS_MASK_SENS_READ          0x08
#define SIGNALS_MASK_ADC_ENABLE         0x04
#define SIGNALS_MASK_ADC_READ           0x02

// Las mascaras de bits de los flags de estado
#define FLAGS_MASK_CONV_COMPLETE        0x80
#define FLAGS_MASK_ALREADY_BUSY         0x40
#define FLAGS_MASK_UNEXPECTED_PAUSE     0x20
#define FLAGS_MASK_ERROR                0x10

// Los mensajes definidos
enum Command
{
    Command_IDENTIFY      = 0,
    Command_SET_SIGNAL    = 1,
    Command_AUTO_READ     = 2,
    Command_GET_RESULT    = 3,
    Command_ABORT         = 4
};

// La cabecera
struct __attribute__((packed)) Header
{
    uint32_t      magic;    // PROTOCOL_MAGIC
    uint8_t       cmd;      // enum Command, respuestas deben usar el mismo que el solicitud
};
```

```
34 // ---------------------------------
35 // Solicitudes
36 // ---------------------------------
37
38 // Comand_IDENTIFY
39 struct __attribute__((packed)) IdentifyRequest
40 {
41     struct Header    hdr;      // La cabecera
42 };
43
44 // Command_SET_SIGNAL
45 // Cambia las salidas al sensor / ADC:
46 // nuevos = (anteriores &  mask) — (value & mask);
47 // dónde nuevos, anteriores, mask, y value son mascaras de bits (SIGNALS_MASK_...)
48 struct __attribute__((packed)) SetSignalRequest
49 {
50     struct Header    hdr;      // La cabecera
51     uint16_t         sync;     // Tiempo de sincronización (t_sync)
52     uint8_t          mask;     // Mascara de bits
53     uint8_t          value;    // Señales a cambiar
54 };
55
56 // Command_AUTO_READ
57 struct __attribute__((packed)) AutoReadRequest
58 {
59     struct Header    hdr;          // La cabecera
60     uint16_t         sync;         // Tiempo de sincronización (t_sync)
61     uint32_t         timing1;      // [24:0] Tiempo entre sens_en/adc_en y sens_read
62     uint32_t         timing2;      // [24:0] Tiempo entre sens_read y adc_read
63 };
64
65 // Comand_GET_RESULT
66 struct __attribute__((packed)) GetResultRequest
67 {
68     struct Header    hdr;      // La cabecera
69 };
70
71 // Comand_ABORT
72 struct __attribute__((packed)) AbortRequest
73 {
74     struct Header    hdr;      // La cabecera
75 };
76
77
```

```
78 // ------------------------------
79 // Respuestas
80 // ------------------------------
81
82 // Command_IDENTIFY
83 struct __attribute__((packed)) IdentifyReply
84 {
85     struct Header    hdr;                       // La cabecera
86     uint8_t          protocol_version;          // La versión de este protocolo
87                                                 // (PROTOCOL_VERSION)
88     uint8_t          adapter_version;           // La versión del adaptador (adapter.sv)
89     uint8_t          iso_iec_14443a_version;    // [7:4] versión del núcleo IP digital
90                                                 // [3:0] versión del AFE
91     uint8_t          sensor_adc_version;        // [7:4] versión del sensor
92                                                 // [3:0] versión del ADC
93 };
94
95 // Command_GET_RESULT
96 struct __attribute__((packed)) GetResultReply
97 {
98     struct Header    hdr;          // La cabecera
99     uint8_t          flags;        // Flags de estado (FLAGS_MASK_...)
100    uint16_t         adc_value;    // El último dato muestreado
101 };
102
103 // Respuesta a cualquier otro comando incluyendo los comandos inválidos.
104 struct __attribute__((packed)) StatusReply
105 {
106     struct Header    hdr;      // La cabecera
107     uint8_t          flags;    // Flags de estado (FLAGS_MASK_...)
108 };
```

Bibliografía

[1] Fabricio Alcalde, Octavio Alpago y José Lipovetsky. «CMOS design of an RFID interface compatible with ISO/IEC-14443 type a protocol». En: (2014). DOI: 10.1109/EAMTA.2014.6906077.

[2] Rakesh ChadhaJ. Bhasker. *Static Timing Analysis for Nanometer Designs*. Springer, 2009. ISBN: 978-0-387-93819-6. DOI: 10.1007/978-0-387-93820-2.

[3] M. Garcia-Inza y col. «6MV LINAC characterization of a MOSFET dosimeter fabricated in a CMOS process». En: *Radiation Measurements* 117 (2018), págs. 63-69. ISSN: 1350-4487. DOI: 10.1016/j.radmeas.2018.07.009. URL: https://www.sciencedirect.com/science/article/pii/S1350448718301586.

[4] Mariano Garcia-Inza y col. «Radiation Sensor Based on MOSFETs Mismatch Amplification for Radiotherapy Applications». En: *IEEE Transactions on Nuclear Science* 63.3 (2016), págs. 1784-1789. DOI: 10.1109/TNS.2016.2560172.

[5] Wilson Research Group. «2020 Wilson Research Group functional verification study – IC/ASIC functional verification trend report». En: (2020). URL: https://verificationacademy.com/seminars/2020-functional-verification-study/rte/trend-reports.

[6] ICRP. *ICRP publication 86. Prevention of Accidents to Patients Undergoing Radiation Therapy*. Annals of the ICRP. London, England: Elsevier Health Sciences, 2000.

[7] *IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language*. en. Standard IEEE 1800-2009. 2009. URL: https://standards.ieee.org/ieee/1800/3989/.

[8] *IEEE Standard Verilog Hardware Description Language*. en. Standard IEEE 1364-2001. 2001. URL: https://standards.ieee.org/ieee/1364/2052/.

[9] Dr. Ing. Mariano Garcia Inza. «Diseño de sensores CMOS de radiación ionizante». 2019.

[10] ISO Central Secretary. *Identification Cards – Contactless integrated circuit card(s) – Proximity cards – Part 1: Physical characteristics*. en. Standard ISO/IEC FDIS 14443-1:1999(E). International Organization for Standardization, International

Electrotechnical Commission, 1999. URL: https://www.iso.org/standard/28728.html.

[11] ISO Central Secretary. *Identification cards – Contactless integrated circuit cards – Proximity cards – Part 2: Radio frequency power and signal interface*. en. Standard ISO/IEC 14443-1:2016. International Organization for Standardization, International Electrotechnical Commission, 2016. URL: https://www.iso.org/standard/66288.html.

[12] ISO Central Secretary. *Identification cards – Contactless integrated circuit cards – Proximity cards – Part 3: Initialization and anticollision*. en. Standard ISO/IEC 14443-3:2016. International Organization for Standardization, International Electrotechnical Commission, 2016. URL: https://www.iso.org/standard/70171.html.

[13] ISO Central Secretary. *Identification cards – Contactless integrated circuit cards – Proximity cards – Part 4: Transmission protocol*. en. Standard ISO/IEC 14443-4:2016. International Organization for Standardization, International Electrotechnical Commission, 2016. URL: https://www.iso.org/standard/70172.html.

[14] ISO Central Secretary. *Information technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures*. en. Standard ISO/IEC 13239:1997. 1997. URL: https://www.iso.org/standard/21474.html.

[15] Arana Leandro Javier. «Interfaz RFID para dosimetría MOS de aplicación médica». Universidad de Buenos Aires, 2018.

[16] David J. Kinniment. *Synchronization and Arbitration in Digital Systems*. Wiley, ene. de 2008. ISBN: 9780470510827.

[17] Gerald J. Kutcher y col. *Comprehensive QA for Radiation Oncology*. Inf. téc. 1994. DOI: 10.37206/45.

[18] OutputLogic.com. *CRC Generator*. 2009. URL: http://outputlogic.com/?page_id=321.

[19] *Quality Assurance in Radiotherapy*. Geneva, Switzerland: World Health Organization, dic. de 1988. ISBN: 9241542241.

[20] CampusIDNews Staff. *IS THE DEBATE STILL RELEVANT? An in-depth look at ISO 14443 and its competing interface types*. 2003. URL: https://www.campusidnews.com/is-the-debate-still-relevant-an-in-depth-look-at-iso-14443-and-its-competing-interface-types/.

[21] Stuart Sutherland y Don Mills. «Synthesizing SystemVerilog – Busting the Myth that SystemVerilog is only for Verification». En: (2013). URL: https://sutherland-hdl.com/papers/2013-SNUG-SV_Synthesizable-SystemVerilog_paper.pdf.

[22] *Formality® User Guide*. Versión T-2022.03. Mar. de 2022.

[23] *IC Compiler™ II User Guide*. Versión R-2020.09-SP1. Oct. de 2020.

[24] *Milkyway™ Database Application Note*. Versión J-2014.09. Sep. de 2014.

[25] *Power Compiler™ User Guide*. Versión R-2020.09. Sep. de 2020.

[26] E.G. Villani y col. «A monolithic 180 nm CMOS dosimeter for wireless In Vivo Dosimetry». En: *Radiation Measurements* 84 (2016), págs. 55-64. ISSN: 1350-4487. DOI: 10.1016/j.radmeas.2015.11.004. URL: https://www.sciencedirect.com/science/article/pii/S1350448715300755.

[27] *Digital Implementation Guidelines for Multibit Registers in 180nm Technologies*. Versión V1.2.1. Mayo de 2020.

[28] *X-FAB XH018 Digital Standard Cell Libraries Overview*. Versión 3.7.0. Abr. de 2018.