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.

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 [8]. 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. [15, 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 [14][5], 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 [8]. 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 [3][2]. 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 [13], 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 [18], 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 [16].

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

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

El PCD emite un campo electromagnético con frecuencia $f_c = 13{,}56\,\text{MHz}$. 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.

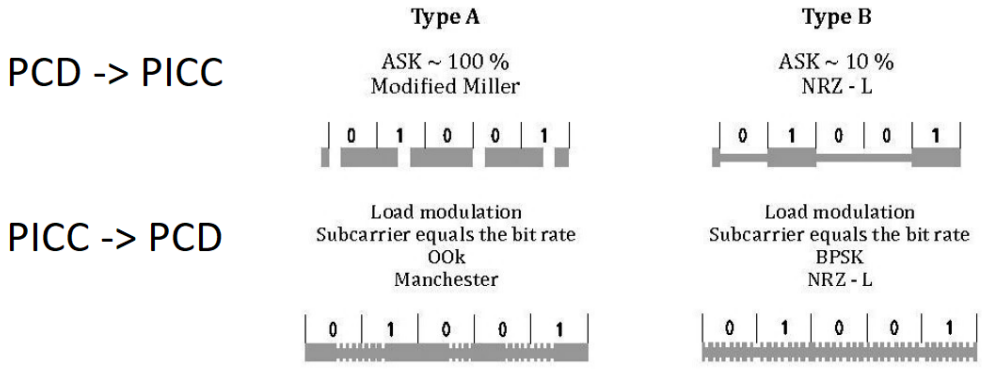

Figura 1.1: Comunicaciones con una tasa de bits de $f_c/128$ [9, Modificaciones Mías]

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 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.

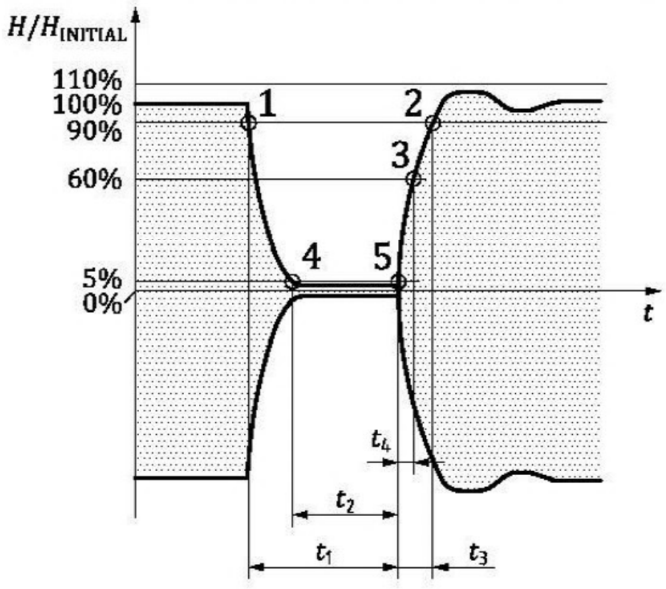

Figura 1.2: El formato de una pausa tipo A [9]

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 [9]

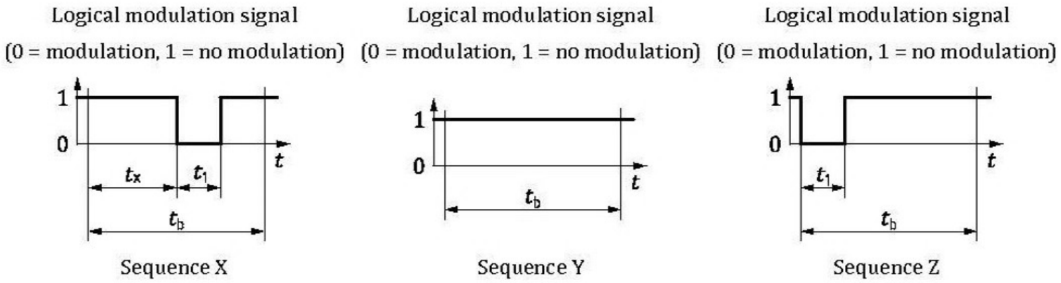

Figura 1.3: Codificación Modified Miller [9]

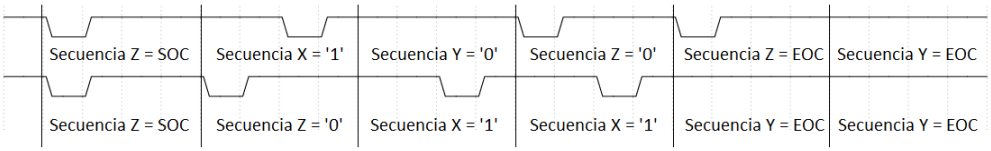

Figura 1.4: Ejemplos de tramas PCD → PICC tipo A

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 es $f_c/16 \approx 848\,\text{kHz}$. 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.

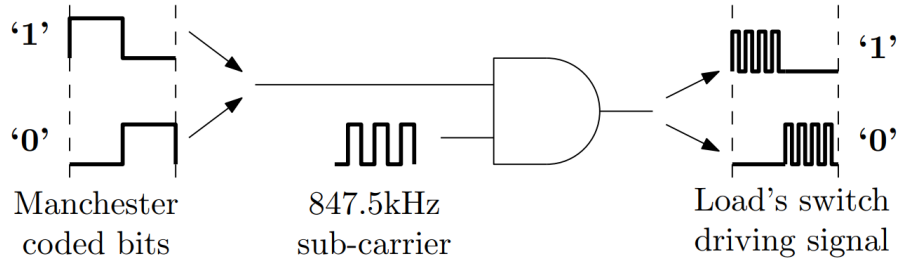

Figura 1.5: Codificación de bits y modulación de la portadora con la subportadora para comunicaciones de la PICC a PCD [1]

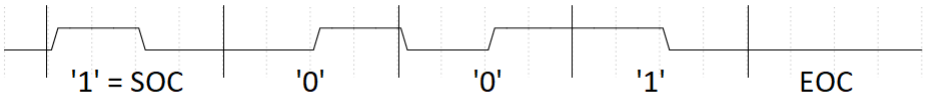

Figura 1.6: Ejemplo de una trama por comunicaciones PICC tipo A → PCD.

ISO/IEC 14443-3: Inicialización y Anticolisión

La información presentada en esta sección es válida considerando una tasa de bit de fc/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 [10][12].

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

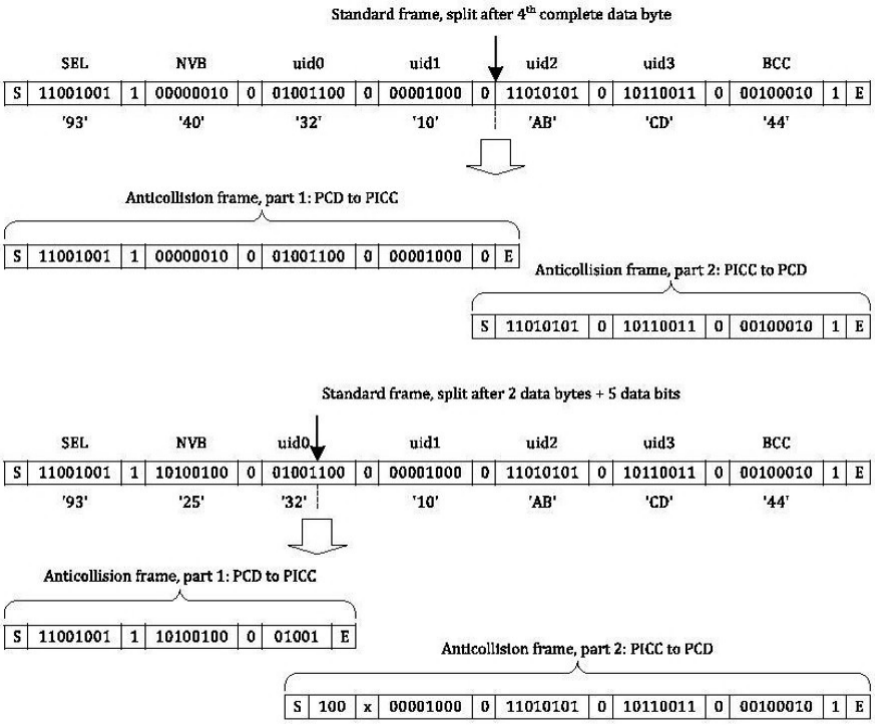

Figura 1.7: Trama Anticolisión orientada a bits [10, Modificaciones Mías]

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.

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

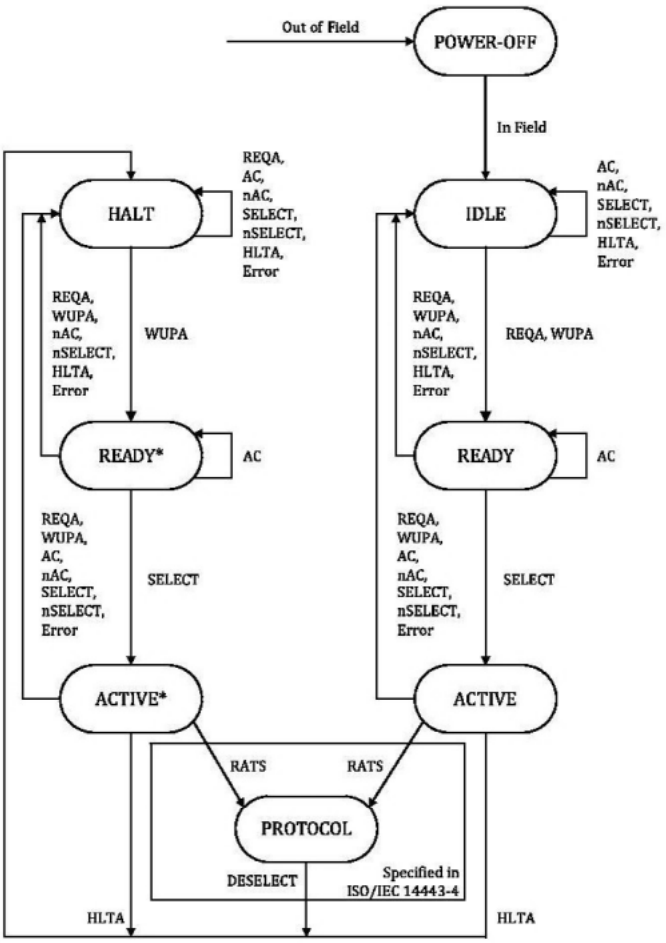

Figura 1.8: Diagrama de transiciones de estados [10]

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.

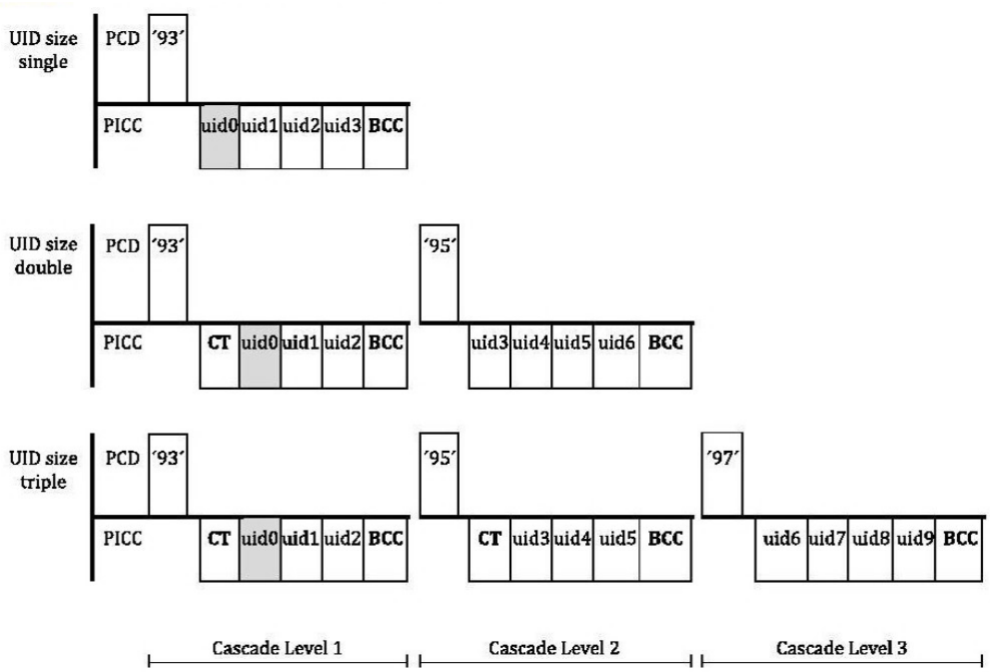

Figura 1.9: Niveles cascadas para UIDs simples, dobles y triples [10]

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) [11]

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 General 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:

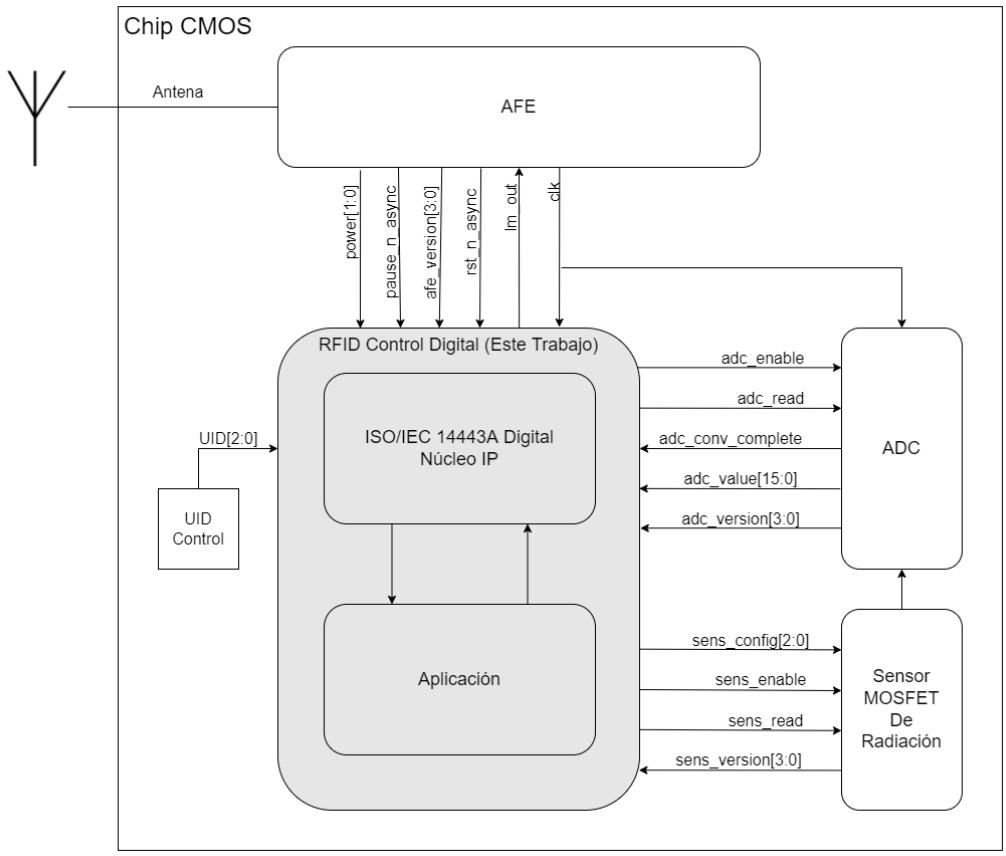

Figura 2.1: Diagrama general del proyecto marco y su relación con el bloque a diseñar en esta Tesis.

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 variable 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 más bajos 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 [6]. SystemVerilog es basado en Verilog, formalmente definido en IEEE 1364 [7]. 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 [17], 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.

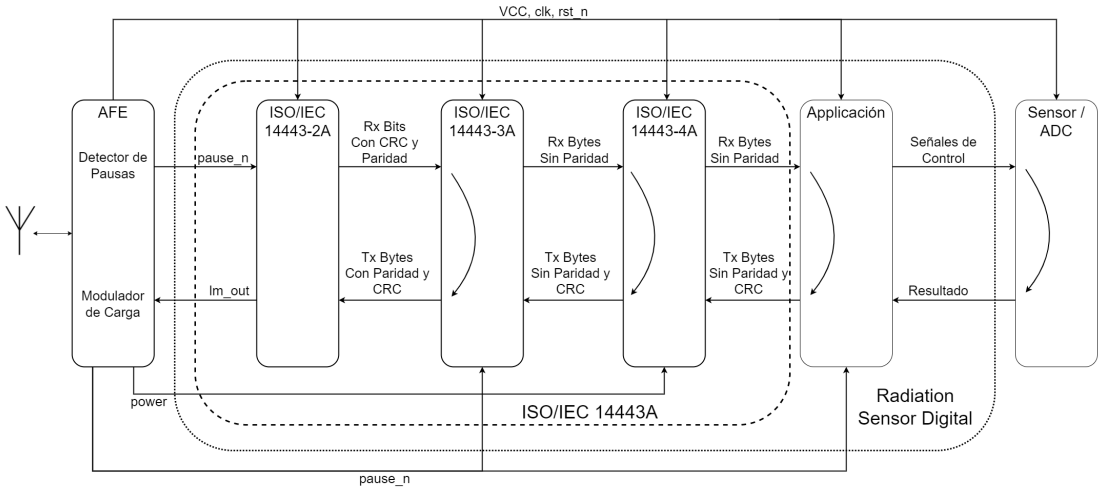

Figura 3.1: Los bloques principales de la implementación.

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. [6, 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 una o más modports, los que especifican las direcciones de las señales. Esos modports pueden ser usados en la lista de puertas 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.

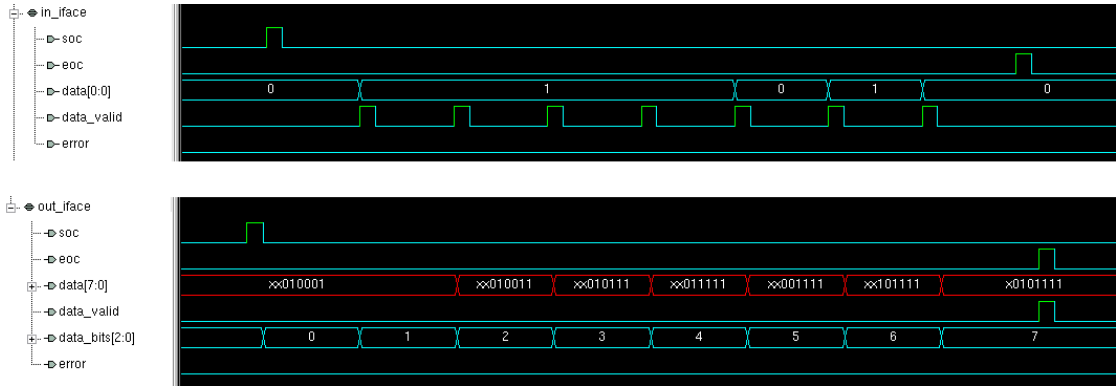

Figura 3.2: Una simulación de la recepción de una trama corta representada con una *rx_interface* de bits (arriba) y su conversión a una *rx_interface* de bytes (abajo).

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.

Cuándo 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.

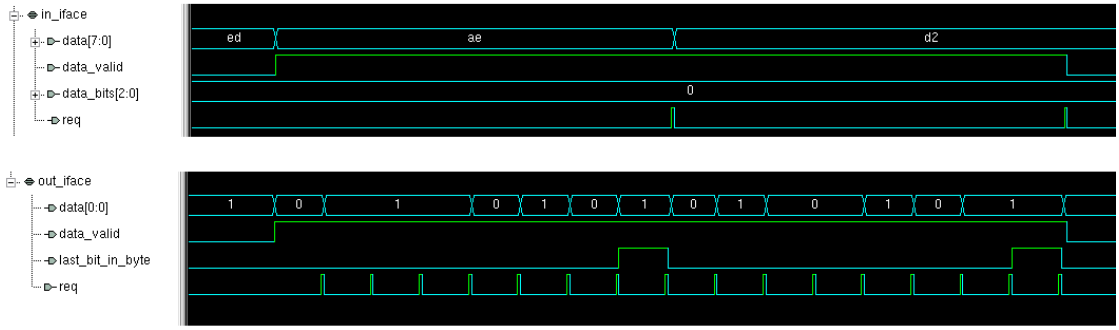

Figura 3.3: Una simulación de la transmisión de una trama de 16 bits representada con una *tx_interface* de bytes (arriba) y su conversión a una *tx_interface* de bits (abajo).

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 [4], 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 [4]. 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 el dato 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 ? (14443-2A). **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.

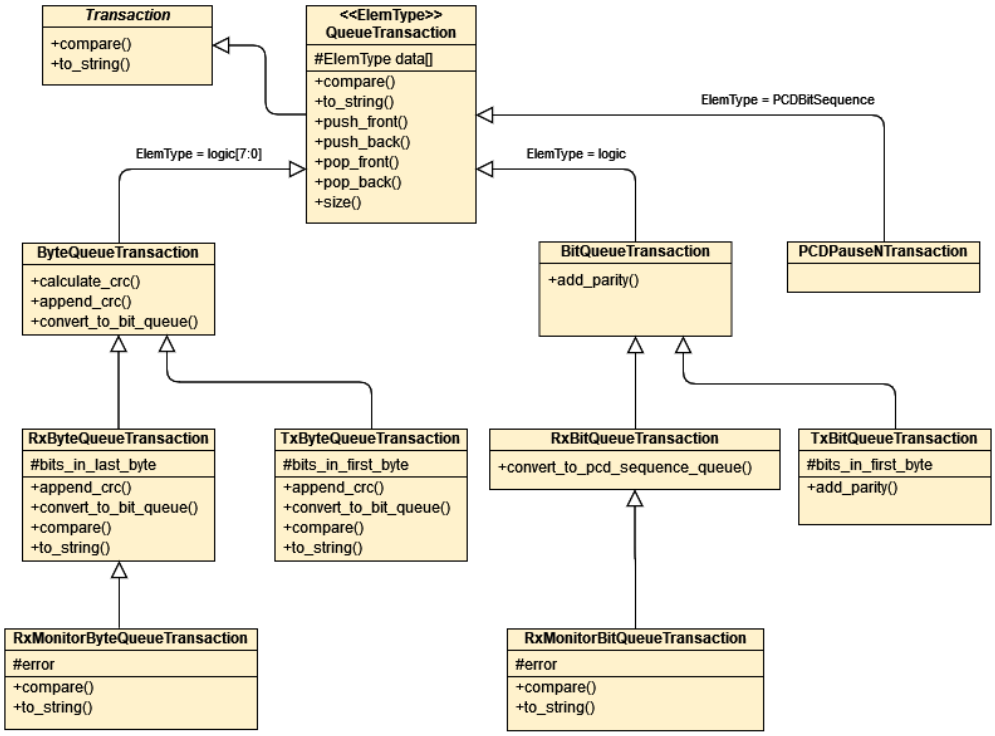

Figura 3.4: Diagrama UML de clases de transacciones.

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'.

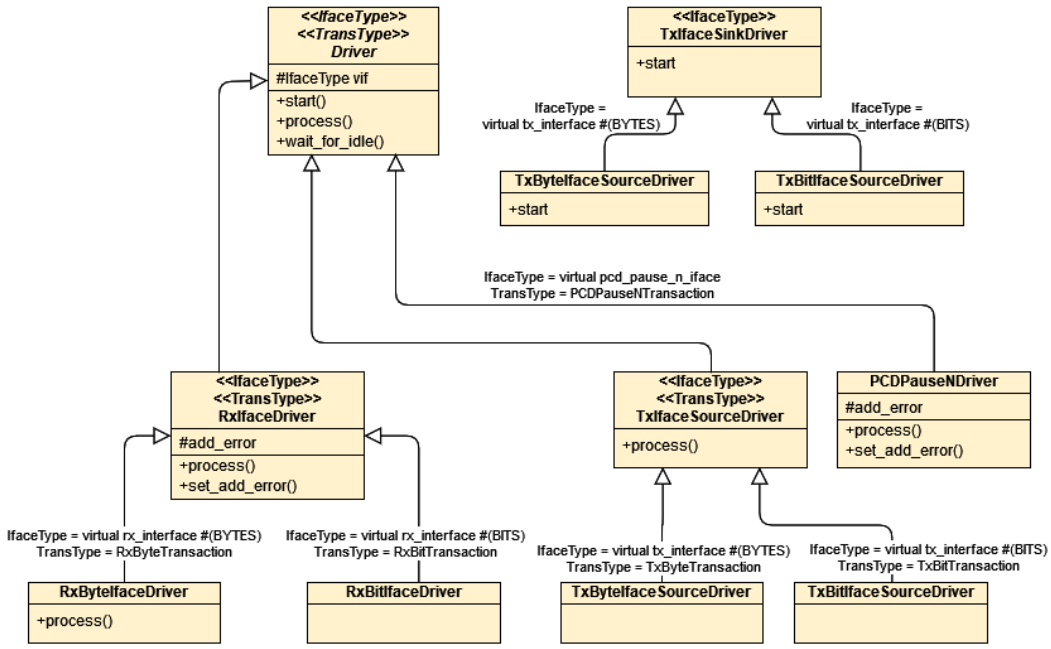

Figura 3.5: Diagrama UML de clases para los controladores.

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.

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.

Para la *rx_interface*, las clases son **RxIfaceMonitor** y sus hijas **RxByteIfaceMonitor**

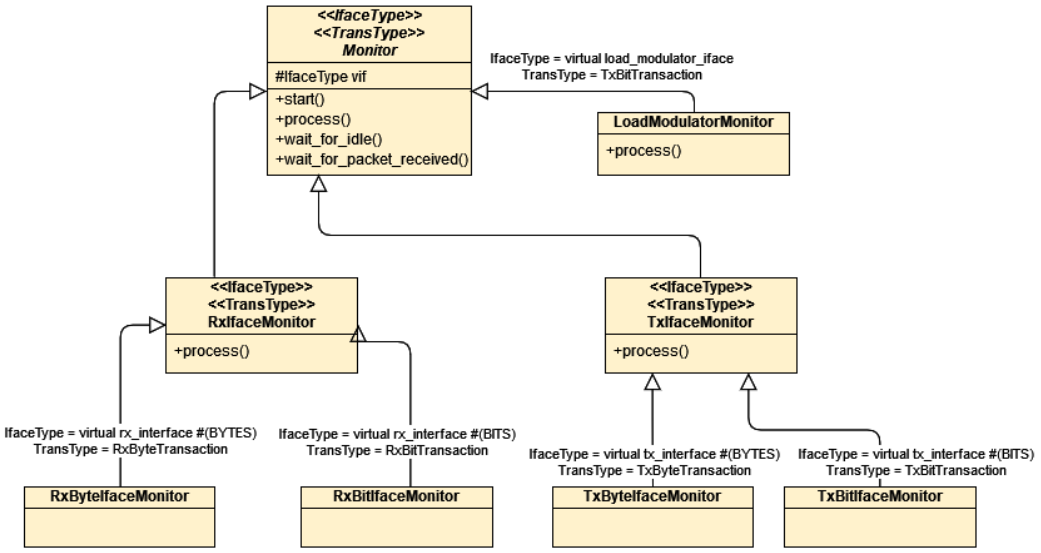

Figura 3.6: Diagrama UML de clases para los monitores.

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: 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 de la norma 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.

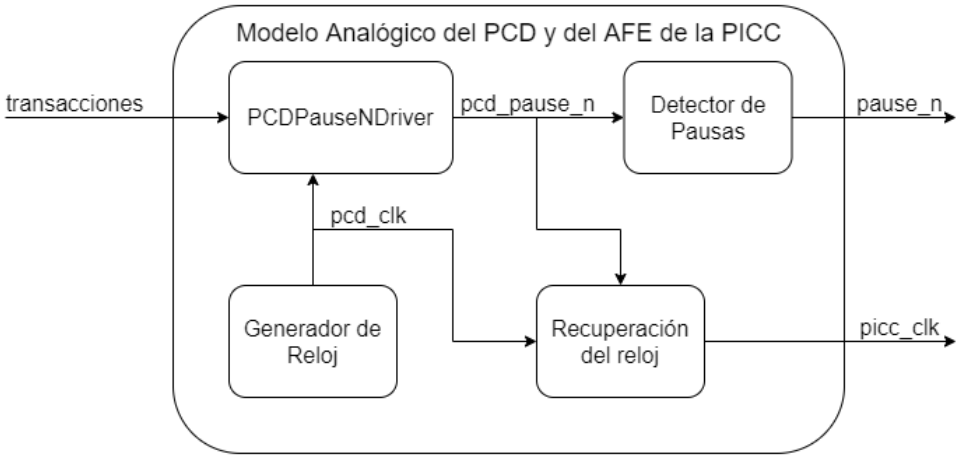

Figura 3.7: Modelo del PCD y del AFE de la PICC para uso en simulaciones.

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.

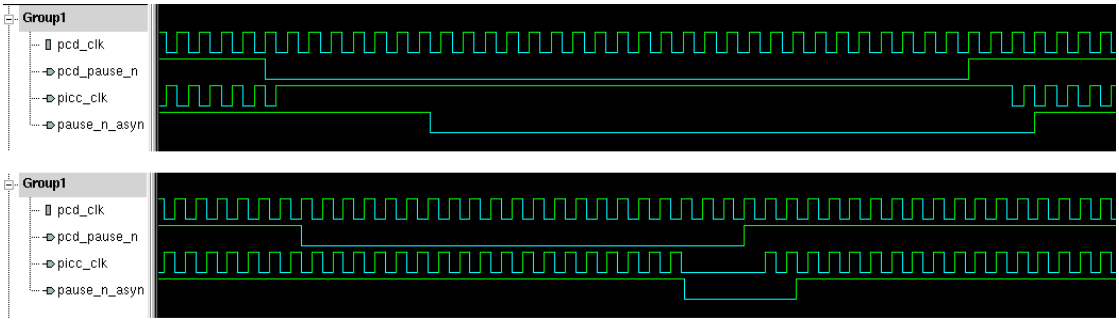

Figura 3.8: Dos ejemplos con timings diferentes del modelo analógico del PCD y el AFE de la PICC.

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

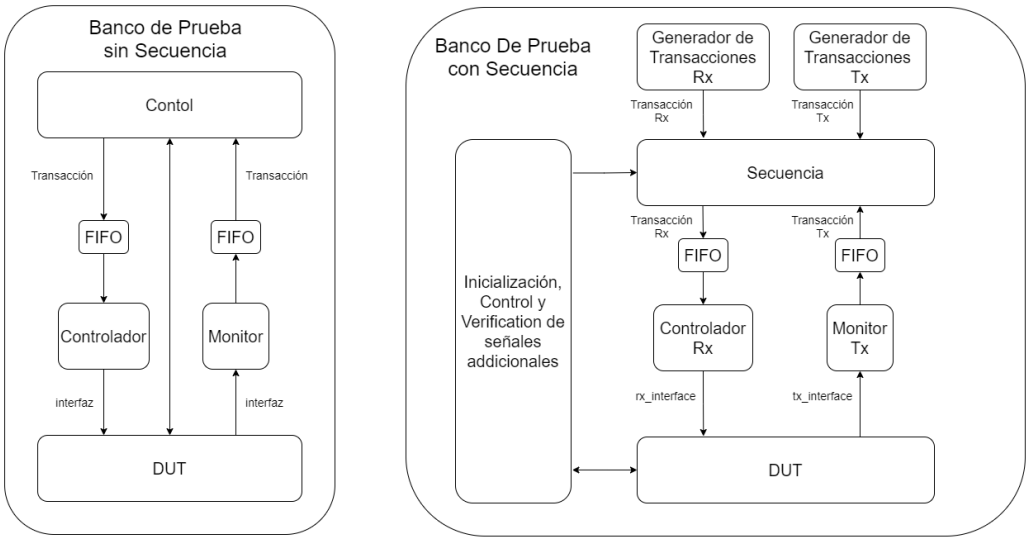

Figura 3.9: Dos estructuras comunes de los bancos de pruebas.

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.

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 fc/16, y que el periodo de bit debería comenzar con el estado cargado de la portadora [ISO/IEC 14443A]. 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 [iso/iec 14443-2] 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.

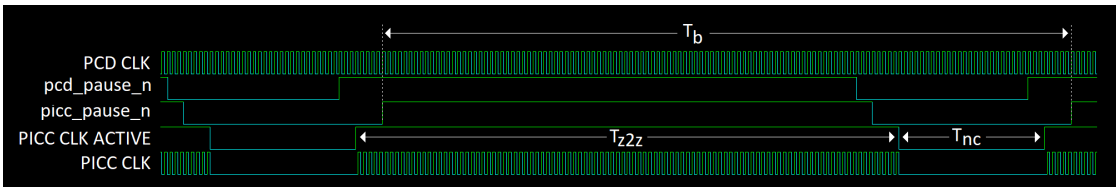

Figura 3.10: Diagrama de timing de dos pausas y el comportamiento del reloj de la PICC.

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 $T_{nc} = 0$.

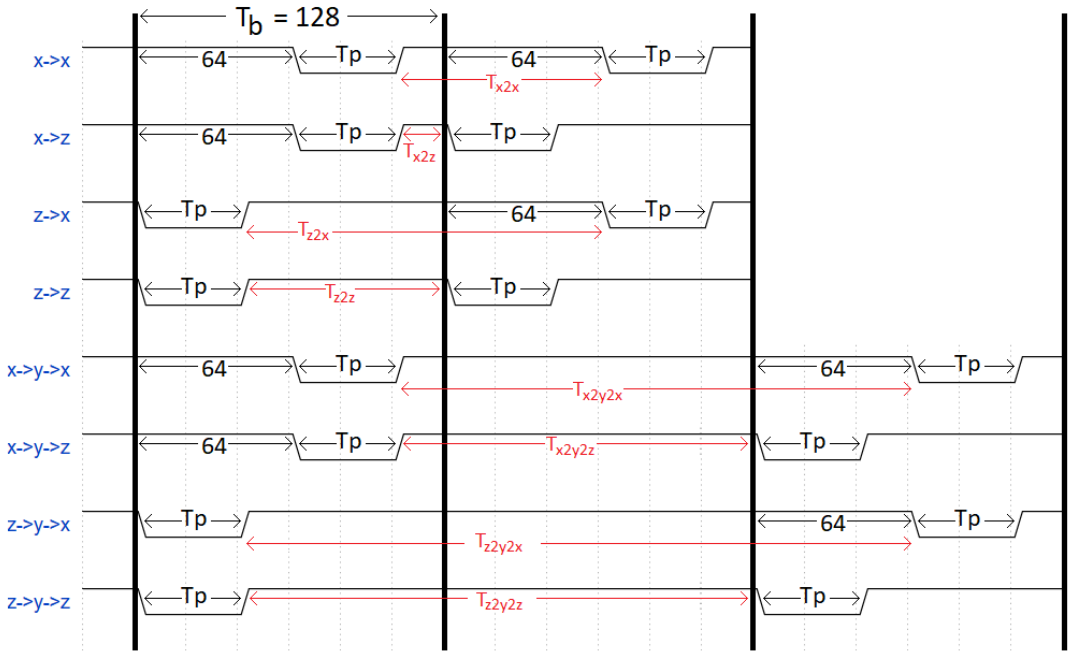

Figura 3.11: Números de ciclos posibles entre dos pausas.

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 tres 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 laCuadro 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.

La 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. El diseño comienza en un estado inactivo, cuándo detecta la primera pausa emite una secuencia Z debido a que el SOC de la trama es una secuencia Z. 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.

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

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**.

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 enviada por el PCD y el reloj deteniéndose.
- El retardo entre el final de una pausa enviada por el PCD y el reloj activándose de nuevo.
- El retardo entre el principio de una pausa enviada por el PCD y el AFE detectándolo.
- El retardo entre el final de una pausa enviada por el PCD y el AFE detectándolo.

Para asegurar que un amplio rango de las condiciones posibles 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 de la PICC. 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 este diseño funciona si el reloj se detiene cómo mínimo 0 ciclos, y cómo máximo 59 ciclos. Esos valores corresponden con un *missing_edges* entre 0 y 118. Las pruebas son repetidos cien veces para cada uno de *missing_edge* 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 en los que falta cobertura son condiciones imposibles de ocurrir, por ejemplo emitiendo una secuencia Z después de una secuencia X. La norma define una Z → X 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**.

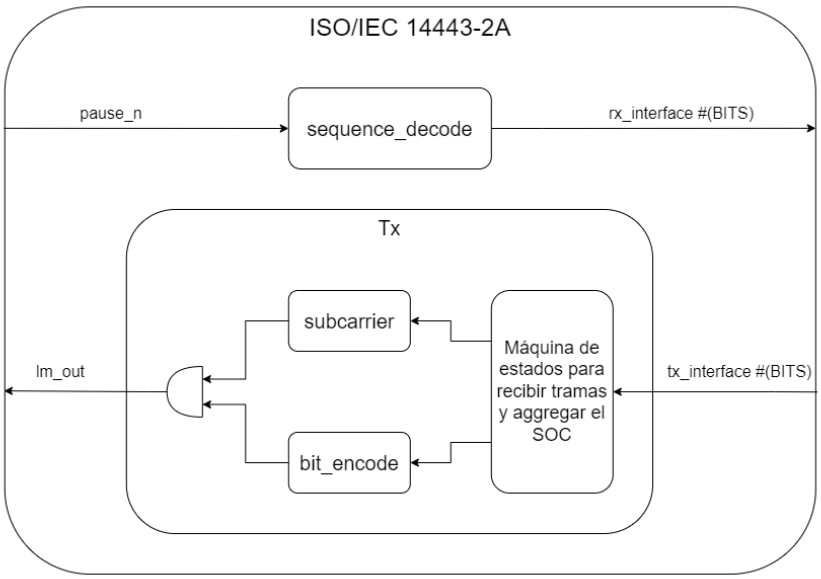

Figura 3.12: Diagrama en bloque del módulo **iso14443_2a**.

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.

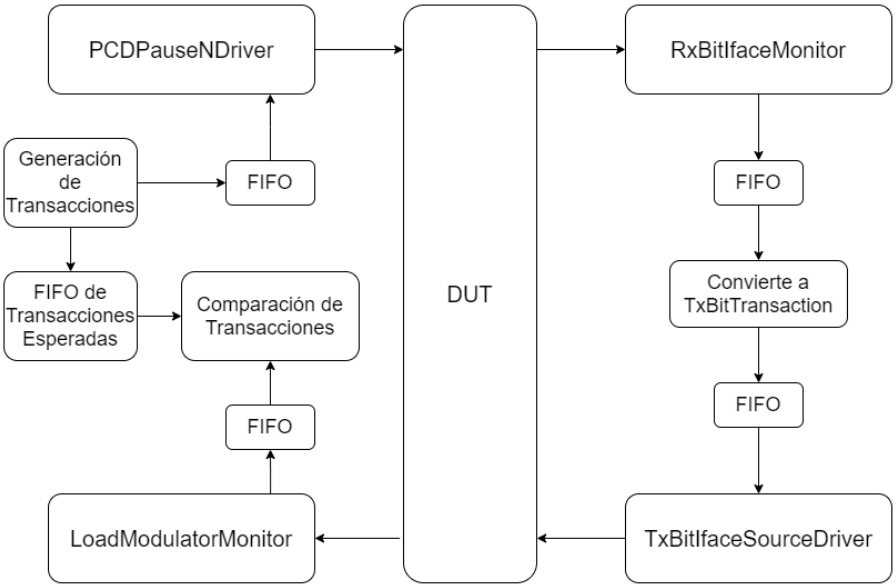

Figura 3.13: Diagrama en bloques del banco de prueba para el módulo **iso14443_2a**

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 compuertas (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 la respuesta adecuada.
- **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.

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

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**.

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 último bit recibido, incluyendo bits de paridad. Esta señal es usada en el módulo **FDT** para determinar cuándo comenzar a enviar la respuesta.

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, 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 la última bit enviada. La segunda verifica que esa señal no cambia entre tramas.

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

deserialiser

FDT

CRC_A

crc_control

serialiser

frame_encode

framing

routing

initialisation

ISO/IEC 14443-3A

ISO/IEC 14443-4A

ISO/IEC 14443A

Otros

synchroniser

active_low_reset_synchroniser

pause_n_latch_and_synchroniser

Aplicación - Interfaz con el sensor MOSFET de radiación y el ADC

Sincronización

Protocolo

Ejemplos

Síntesis y Place & Route

Síntesis

Preparación de librerías

Design Planning

Place & Route

LVS / DRC

Resultados y Conclusiones

Recomendaciones para Trabajos Futuros

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] 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.

[3] 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.

[4] Wilson Research Group. «2020 Wilson Research Group functional verification study – IC/ASIC functional verification trend report». En: (2020). URL: https://uobdv.github.io/Design-Verification/WilsonResearchGroupFunctionalVerificationStudy/2020-WRGFV-Study/ic-asic-trend-report_2020-wilson-research-verification-study_hfoster.pdf.

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

[6] *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/.

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

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

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

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

[16] 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/.

[17] 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.

[18] 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.