UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Ingeniería Electrónica

VERIFICACIÓN BASADA EN MÉTRICAS DEL DISEÑO DE UN ROIC

Trabajo Fin de Máster

Máster Universitario en Ingeniería de Sistemas Electrónicos

AUTOR/A: Melgar Beltrán, Tomás

Tutor/a: Martínez Peiró, Marcos Antonio

Cotutor/a: Torres Curado, Rubén

CURSO ACADÉMICO: 2023/2024

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

VERIFICACIÓN BASADA EN MÉTRICAS DEL DISEÑO DE UN ROIC

Autor: Tomás Melgar Beltrán

Tutor: Marcos Antonio Martínez Peiró

Cotutor: Rubén Torres Curado

Tutor de Empresa: Alberto Arias Drake

Trabajo Fin de Máster presentado en el Departamento de Ingeniería Electrónica de la Universitat Politècnica de València para la obtención del Título de Máster Universitario en Ingeniería de Sistemas Electrónicos

Curso 2023-24

Valencia, Julio de 2024

Resumen

Este trabajo trata sobre la verificación digital de un circuito integrado encargado de proporcionar digitalmente las lecturas que le llegan de un sensor de píxeles activos. Este dispositivo está siendo desarrollado por ams-OSRAM y será utilizado en equipos de radiografía digital. Además de un convertidor ADC por cada canal de entrada, el chip cuenta con varios bloques digitales encargados de diversas funciones: filtrar las lecturas, controlar la comunicación SPI y los registros de configuración, enviar los datos por hasta cuatro salidas LVDS, medir la temperatura interna y suministrar las señales de referencia al sensor. Se desea comprobar el correcto funcionamiento y cumplimiento de todas las especificaciones, tanto de cada parte aislada como del conjunto completo. Para esto, se desarrolla un plan de verificación que se enlaza con una serie de test. Se va midiendo si los test cubren todas las necesidades funcionales del diseño y todo su código. Se sigue por tanto la metodología conocida como verificación basada en métricas (MDV).

Resum

Este treball tracta sobre la verificació digital d'un circuit integrat encarregat de proporcionar digitalment les lectures que li arriben d'un sensor de píxels actius. Este dispositiu està sent desenvolupat per ams-OSRAM i serà utilitzat en equips de radiografia digital. A més d'un convertidor ADC per cada canal d'entrada, el xip compta amb diversos blocs digitals encarregats de diverses funcions: filtrar les lectures, controlar la comunicació SPI i els registres de configuració, enviar les dades per fins a quatre eixides LVDS, mesurar la temperatura interna i subministrar els senyals de referència al sensor. Es desitja comprovar el correcte funcionament i compliment de totes les especificacions, tant de cada part aïllada com del conjunt complet. Per a això, es desenvolupa un pla de verificació que s'enllaça amb una sèrie de tests. Es va mesurant si els tests cobrixen totes les necessitats funcionals del disseny i tot el seu codi. Se seguix per tant la metodologia coneguda com a verificació basada en mètriques (MDV).

Abstract

This work deals with the digital verification of an integrated circuit in charge of providing digitally the readings coming from an active pixel sensor. This device is being developed by ams-OSRAM and will be used in digital radiography equipment. In addition to an ADC converter for each input channel, the chip has several digital blocks responsible for various functions: filtering the readings, controlling the SPI communication and configuration registers, sending the data through up to four LVDS outputs, measuring the internal temperature, and supplying the reference signals to the sensor. It is desired to check the correct operation and compliance with all specifications, both of each isolated part and of the complete assembly. For this, a verification plan is developed and linked to a series of tests. It is measured whether the tests cover all the functional needs of the design and all its code. The methodology known as metric-driven verification (MDV) is therefore followed.

Palabras clave: verificación, Metric Driven Verification, vManager, SystemVerilog, aserciones

Dedicado a mis familiares y amigos, y también a todos los demás.

Índice general

Índice de figuras

Índice de tablas

Listado de siglas empleadas

ADC Conversor analógico-digital (Analog to Digital Converter).

API Interfaz activa de píxeles (Active Pixel Interface).

APS Sensor de píxeles activos (Active Pixel Sensor).

CCD Dispositivo de carga acoplada (Charge-Coupled Device).

CMOS Semiconductor complementario de óxido metálico (Complementary Metal-Oxide-Semiconductor).

CRC Verificación por redundancia cíclica (Cyclic Redundancy Check).

CS Chip Select.

DMS Señal digital y mixta (Digital and Mixed-Signal).

DUT Dispositivo bajo prueba (Device Under Test).

LSB Bit de menor peso (Least Significant Bit).

LVDS Señal diferencial de bajo voltaje (Low-Voltage Differential Signaling).

MDV Verificación basada en métricas (Metric Driven Verification).

MISO Entrada maestro - Salida esclavo (Master Input - Slave Output).

MOSI Salida maestro - Entrada esclavo (Master Output - Slave Input).

OSR Ratio de sobremuestreo (Oversampling Ratio).

OTP Programable una sola vez (One Time Programmable).

PL Posterior a la disposición del circuito integrado (Post-Layout).

RAM Memoria de acceso aleatorio (Random Access Memory).

ROIC Circuito integrado de lectura (Readout Integrated Circuit).

RTL Lógica de resistencia-transistor (Register Transfer Level).

SCLK Reloj SPI (SPI Clock).

SDF Formato estándar de retrasos (Standard Delay Format).

SnH Muestreo y retención (Sample and Hold).

SPI Interfaz periférica en serie (Serial Peripheral Interface).

TnH Seguimiento y retención (Track and Hold).

UVM Metodología universal de verificación (Universal Verification Methodology).

Parte I

Memoria

Capítulo 1

Introducción

1.1. Contexto

En el ámbito de la imagen médica, la radiología digital es un método para crear imágenes de secciones de un objeto con un equipo que utiliza rayos X. Existen tres formas principales de digitalizar una radiografía:

- Escaneando una película (analógica) una vez esta ha sido revelada.
- Escaneando una placa de fósforo fotoestimulable que se graba con la imagen de la prueba radiológica (radiografía), y se puede posteriormente reutilizar.
- Mediante detectores sensibles, estos, o bien detectan directamente los rayos X, o bien están indirectamente expuestos, y detectan típicamente la luz que emite un centellador situado delante suyo al recibir la radiación.

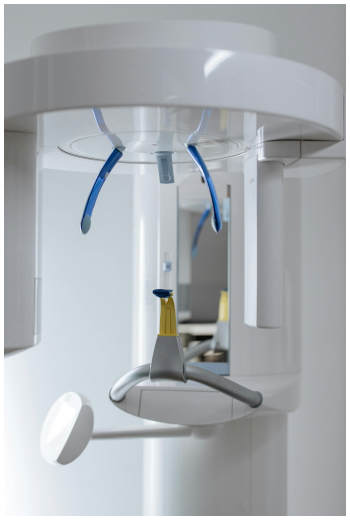


Figura 1.1: Ejemplo de imagen médica usando radiología digital: equipo de tomografía computarizada dental.

El sistema para el que está pensado el chip a verificar en este proyecto usa el último método, que es el único de los tres que permite digitalizar las imágenes en tiempo real. Algunas de las ventajas que caracterizan a estas máquinas son:

- No requieren gasto material ni el uso de productos químicos
- Consiguen imágenes de mayor calidad gracias al filtrado digital
- Dan más información: su mayor resolución provee a las imágenes de muchos más niveles de gris con los que trabajar que los que podría juzgar solamente el ojo humano
- Permiten el sencillo guardado o traspaso de esta información por medios digitales

Y algunas de sus desventajas son que:

- En ocasiones requieren la inyección de un medio de contraste intravenoso
- Proporcionan dosis de radiación ionizante al paciente (menores en las radiologías normales que en las de tomografía)

Una forma de percibir los rayos X, en lugar de medirlos directamente, es usar un centellador, que es un material que brilla con luz visible cuando recibe radiación ionizante. Así se puede medir esta luz con un sensor. Los sensores de píxeles activos tienen un detector por cada píxel, formando una matriz en la que se van seleccionando ciertas filas y columnas, y se van leyendo los píxeles en grupos por un número determinado de canales. El chip a verificar es el encargado de leer y digitalizar las salidas de este sensor.

1.2. Justificación

Es necesario un dispositivo que sea capaz de leer cada imagen y proporcionar su lectura digitalmente antes de que esta cambie. Es decir, entre el sensor, con una salida analógica para cada píxel, y el procesador digital, que guarda las imágenes y aplica los algoritmos, hace falta un circuito integrado que sea capaz de digitalizar el valor de todos los píxeles igual o más rápido de lo que estos cambian.

Puesto que la imagen tiene millones de píxeles, lo más barato y práctico es que solo tenga un submúltiplo de convertidores, y cada uno haga varias conversiones para cada imagen. Por ejemplo, se puede diseñar el sistema para convertir todos los píxeles de una fila a la vez (tener tantos convertidores como columnas), y necesitar tantas conversiones como filas por cada imagen.

El circuito integrado de lectura que se verifica en este trabajo tiene 210 entradas analógicas que convierte a digital y transmite en serie, todo esto a elevada frecuencia para poder hacerlo el número de veces necesario para tener una imagen completa en muy poco tiempo. Un sistema completo puede usar uno o varios de estos chips, pero en este trabajo se comprueba el funcionamiento del dispositivo en sí.

En las máquinas de tomografía computarizada, el sensor se va moviendo a velocidad considerable, rotando en torno al objeto, y obteniendo imágenes desde cada ángulo. En estos casos el chip de lectura debe tener una entrada por píxel, no hay tiempo suficiente para ir seleccionando

filas, pero en la radiología digital esta selección sí es posible, lo que permite sensores (y por tanto imágenes) más grandes o con más detectores (y por tanto píxeles).

Por ende, es fundamental que el chip cumpla las especificaciones que se han valorado necesarias, el dispositivo diseñado debe cumplir unos requerimientos de velocidad, fiabilidad y exactitud, además de otras funcionalidades que necesite el sistema al que apoya.

Una vez justificado el chip, es congruente la necesidad de una verificación funcional de su diseño. La especificación define qué debería hacer el aparato (para que sea útil en el contexto explicado), y la verificación comprueba que el diseño lógico que se ha creado concuerda con este documento. Es evidente que resulta mucho más barato y competitivo encontrar los defectos en un diseño electrónico antes de comenzar su fabricación.

La denominada metodología de verificación universal (*Universal Verification Methodology*, UVM) es muy popular hoy en día, pero, puesto que no el proyecto no se encuentra en una serie que la haya estado usando, hacerlo no valía la pena, pues el esfuerzo inicial hubiera sido demasiado elevado y lento. Por tanto, se ha usado la también común metodología de verificación basada en métricas, que se explicará en detalle en este documento.

1.3. Estructura

La memoria de este trabajo está dividida en varios capítulos. El primero es de introducción, en él se expone el entorno de este trabajo, su necesidad, y por último, en este apartado, la forma en que se va a describir el proyecto en este documento y el contenido de cada parte. Los siguientes dos capítulos comprenden el marco teórico.

El capítulo segundo trata de los fundamentos del diseño que se ha verificado. Antes de nada se da una introducción a los sensores de píxeles activos, puesto que el chip que nos concierne sirve para leer las salidas de uno. Tras esto se explica este tipo de circuitos de lectura, y luego los bloques funcionales más importantes que el de este trabajo tiene: el convertidor analógico-digital (y el tipo que usa este caso), el transmisor por señales diferenciales de bajo voltaje y el transmisor de interfaz periférica en serie. Después de esta contextualización, se expone un muy breve resumen de las especificaciones de esta solución concreta que se debe verificar. Hay una sección dedicada a las formas en que puede operar este chip, según su configuración, pues este es el aspecto más importante y complejo de su funcionamiento.

El tercer capítulo completa la parte teórica hablando de la verificación en sí. Comienza con el lenguaje que se utiliza, que es una combinación de Verilog y SystemVerilog. El segundo apartado sigue con la forma de constatar el funcionamiento del diseño mediante grupos de pruebas y la necesidad de medir el nivel de esta comprobación. Los apartados tres y cuatro desarrollan dos herramientas muy útiles para este fin. La última sección es sobre el método que ha hecho posible enlazar las necesidades técnicas que se han de comprobar del diseño con los test y simulaciones que se han creado.

El cuarto capítulo engloba el desarrollo práctico de este trabajo. La base de una buena verificación es su plan de verificación, así que se comienza listando sus elementos. Luego se hace también inventario de los test desarrollados para hacer simulaciones del diseño funcionando en distintas circunstancias. Después, cómo se han desarrollado estos, tareas comunes que usan, grupos de cobertura y aserciones. Se acaba explicando como se ha usado el programa vManager para

automatizar, acelerar y depurar el proceso de verificación.

Como se podrá comprobar, en esta memoria se ha tratado de no detallar demasiado la arquitectura del diseño en sí, mientras que sí se analiza de forma más concreta la verificación de este. Además de porque es este el tema del TFM, y no viceversa, esto se debe a que en esta materia puede resultar beneficioso no mirar muy de cerca el código del diseño, es mejor centrarse en las especificaciones requeridas. De lo contrario, el ingeniero verificador puede cometer errores como imitar fallos del diseño en sus modelos de referencia, o inconscientemente crear código que los evita en lugar de sacarlos a la luz.

Finalizan la memoria el capítulo de resultados, que se miden con el número de errores y el porcentaje de cobertura alcanzado, y el de conclusiones, donde se analiza el trabajo realizado y la consecución de su objetivo principal. La bibliografía se halla debajo de la memoria, y como primer anexo se ha completado un breve informe de la relación de este TFM con los objetivos de desarrollo sostenible de la agenda 2030. EL segundo anexo es la encuesta a los alumnos del MUISE.

Capítulo 2

Base teórica y descripción del dispositivo a verificar

2.1. Base teórica

2.1.1. APS

Un sensor de píxeles activos (*Active Pixel Sensor*, APS) es un tipo de sensor de imagen en el que cada elemento (píxel) tiene un fotodetector y un amplificador activo. El término APS se usa también para referirse al propio sensor del píxel individual, en ese caso el sensor de imagen se suele denominar *active pixel sensor imager*, o *active-pixel image sensor* [1]. Este sensor se produce usando tecnología CMOS, surgió como alternativa a a los sensores de imagen de dispositivos de carga acoplada (*Charge-Coupled Device*, CCD).

En general tienen menor consumo y retraso de imagen, y requieren instalaciones de fabricación menos especializadas. Se usan en aplicaciones de consumo, especialmente en teléfonos con cámara, pero también en campos como la radiografía digital, adquisición de imágenes militares a ultra alta velocidad, cámaras de seguridad y ratones ópticos.

La arquitectura del píxel CMOS estándar actual consta de un fotodetector (fotodiodo anclado), una difusión flotante, una puerta de transferencia, una puerta de reinicio (*reset*), un transistor de lectura de seguidor de fuente y una puerta de selección: la llamada celda 4T. Sin embargo, en este caso se usa una variante de solo tres transistores (3T, fig. 2.1), en la que no hay puerta de transferencia, es el transistor de reinicio, M_{rst}, el que actúa como un interruptor para restablecer la difusión flotante a V_{RST}, que en este caso se representa como la puerta del transistor M_{sf}.

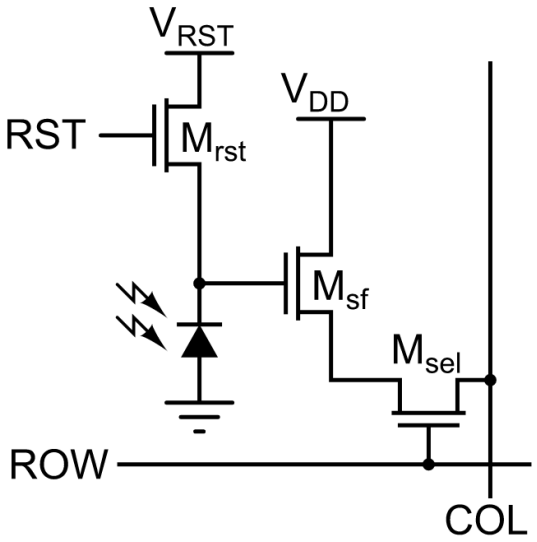


Figura 2.1: Circuito de un APS de tres transistores. Tomado de [1].

Cuando se enciende el transistor de reinicio, el fotodiodo se conecta efectivamente a la fuente de alimentación, V_{RST}, eliminando toda la carga integrada. Dado que el transistor de reinicio es de tipo n, el píxel opera en reinicio suave. El transistor de lectura, M_{sf}, actúa como un *buffer* (específicamente, un seguidor de fuente), un amplificador que permite observar el voltaje del píxel sin eliminar la carga acumulada. Su fuente de alimentación, V_{DD}, normalmente está vinculada a la fuente de alimentación del transistor de reinicio V_{RST}. El transistor de selección, M_{sel}, permite que la electrónica de lectura lea una sola fila de la matriz de píxeles.

2.1.2. ROIC

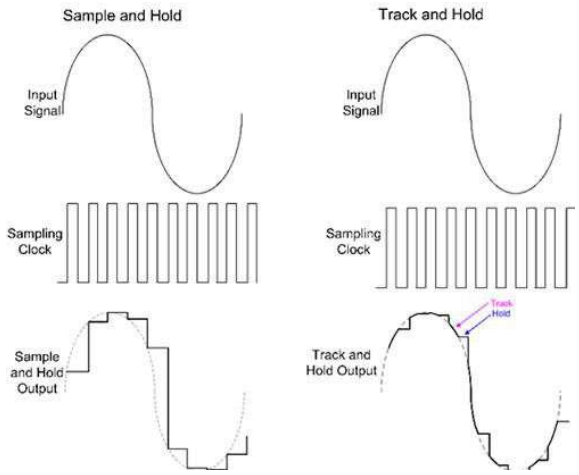
Un circuito integrado de lectura (ROIC) es el chip usado para leer sensores de un tipo concreto. Está compuesto de circuitos responsables de integrar, amplificar y multiplexar las débiles cargas del detector, actúa como la interfaz entre este y la unidad de procesamiento de señal.

El ROIC está conectado al sensor por un conjunto de canales, que sirven de interfaz eléctrica entre los detectores individuales (píxeles) del sensor y las celdas de entrada del ROIC. En cada uno se integra y muestrea la señal apropiadamente, para que después un circuito de control barra fila por fila los canales, y se conviertan en un solo flujo en serie de bits, que será el que se transmita a la unidad de procesamiento.

Hay dos tipos principales de ROIC: modo corriente y modo tensión. El modo corriente es menos común, pero puede tener ventajas como el menor uso de área de silicio [2]. El de nuestro caso es modo tensión. Cuando termina la integración de una muestra, o bien se muestrea el voltaje y retiene en un condensador (*sample and hold*, SnH), o bien se "rastrea" y se retiene en un condensador (*track and hold*, TnH).

La diferencia en estos dos modos es que en el primero no cambia la salida hasta que recibe un disparo en su puerta, durante este pulso de tensión la salida sí cambia al valor de la entrada (fig. 2.2(a)). Por otro lado, el TnH permite a la entrada del circuito pasar a la salida normalmente, hasta que recibe el disparo, mientras esté recibiéndolo la salida mantendrá el valor que tenía la entrada al

comenzar el pulso (fig. 2.2(b)). La diferencia clave entre TnH y SnH es la duración del intervalo de seguimiento: muy corto para SnH y significativamente más largo para TnH [3]. En nuestro caso el ROIC usa TnH.



(a) (b)

Figura 2.2: Gráfico ej. de la diferencia de la generación de la salida entre un circuito SnH (a) y uno TnH (b). Tomado de [3].

Después de este circuito, cada canal puede tener un ADC dedicado, cuyas salidas son las que deben ser transmitidas digitalmente a la unidad de procesamiento. Las etapas de integración y conversión son individuales, y la etapa de transmisión es común para todos los canales, se transmite en serie la información de cada uno. Estas tres etapas pueden ocurrir en paralelo: el ROIC puede estar convirtiendo en los ADC una línea de datos, y al mismo tiempo estar integrando la línea siguiente y transmitiendo en la salida la línea anterior.

2.1.3. ADC

Un convertidor de señal analógica a digital (ADC) es un dispositivo capaz de convertir una señal analógica continua en el tiempo en una digital discretizada y codificada en binario [4]. Se compone principalmente de tres elementos, cuya función específica es la siguiente para cada uno: (fig. 2.3)

- Muestreador: discretiza la señal en el tiempo, es decir, transforma esta señal continua en muestras discretas a base de capturar la señal en intervalos periódicos
- Cuantificador: convierte las amplitudes continuas de las muestras a uno de los valores concretos de un conjunto preestablecido
- Codificador: codifica los valores obtenidos en binario (muchas veces es el cuantificador mismo)

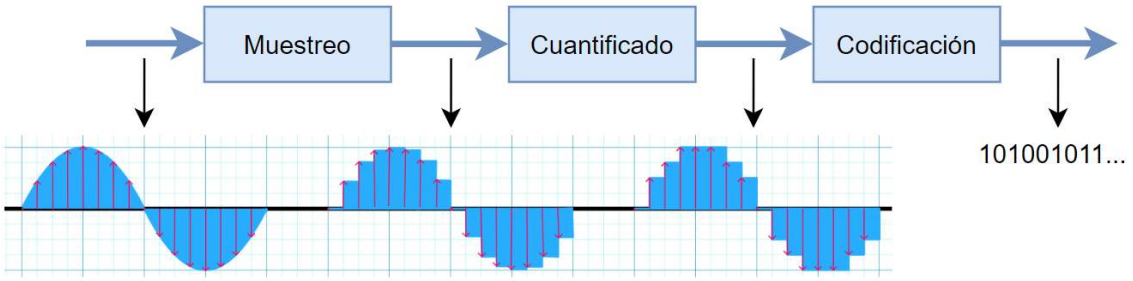


Figura 2.3: Diagrama de las etapas básicas de un ADC y evolución de una onda ejemplo.

El parámetro clave de la etapa de muestreo es el tiempo transcurrido entre cada muestra consecutiva de la señal. Se conoce este valor como el periodo de muestreo (T_s), o su inverso, la frecuencia de muestreo $(f_s = 1/T_s)$. Según el teorema de *Nyquist*, esta debe ser siempre mayor que dos veces la frecuencia máxima de la señal, para que sea posible recuperarla totalmente a partir de sus muestras.

$$f_s > 2 \cdot f_{max} \tag{2.1}$$

Al cuantificar la señal, se divide el rango dinámico de sus valores en un grupo discreto de intervalos. Cada intervalo de valores (de rango Q) es representado ahora por un solo valor, normalmente el valor intermedio de ese intervalo. Se puede definir el error de cuantificación máximo como la distancia máxima entre el valor real y el valor que se le asigna a la muestra tras esta etapa. Este error máximo se encuentra al principio y al final de cada intervalo, cuando están a medio intervalo de distancia del valor que se asignará. Por lo tanto, el error de cuantificación siempre oscila entre $-Q/2$ y $Q/2$, generando el llamado ruido de cuantificación.

Por último, en la codificación se asigna un código binario según cuál de los valores discretos sea. La diferencia mínima entre cada número (el bit de menor peso, llamado *Least Significant Bit*, LSB) es este valor Q de tensión, y se puede encontrar a partir del rango de voltaje del convertidor y el número de bits disponible (n):

$$V_{LSB} = \frac{V_{max} - V_{min}}{2^n} \tag{2.2}$$

Según la aplicación en la que se va a utilizar, se usan distintas arquitecturas. Cada una tiene ciertas ventajas y desventajas, como se puede ver en la tabla 2.1, y se escoge la más apropiada según los requerimientos del sistema.

Topología	Flash	Inter-leaving	Pipelined	SAR	Sigma-Delta	Rampa
Frec. de muestreo (muestr./s)	Alta (1G-10G)	Media-alta (100M-1G)	Media-alta (10M-100M)	Media (100k-10M)	Baja (10k-1M)	Baja (100-1k)
Resolución (bits)	Baja (6-8)	Media-alta (12-18)	Media-alta (12-18)	Media-alta (12-18)	Alta (16-24)	Media-alta (12-18)
Consumo	Alto	Alto	Alto	Muy bajo	Bajo	Bajo
Latencia	Baja	Baja	Alta	Baja	Alta	Baja-media
Área	Alta	Alta	Alta	Baja	Media	Baja
Precisión	Baja	Media	Media-alta	Media-alta	Alta	Variable
Coste	Alto	Alto	Alto	Bajo-alto	Bajo	Medio

Tabla 2.1: Comparativa del rendimiento de distintas arquitecturas ADC [5].

Dadas las ventajas de cada arquitectura, el chip que nos concierne usa convertidores de tipo Sigma-Delta, al ser el tipo que más encaja con los requerimientos de esta aplicación. Esta arquitectura consiste en usar la modulación Sigma-Delta para obtener los valores de salida, que es una derivación de la modulación Delta.

La modulación delta surgió de transmitir los cambios (delta) de valor entre muestras consecutivas, en lugar de las muestras reales en sí, logrando con ello una mayor eficiencia en la transmisión. La señal analógica se cuantifica mediante un ADC de un bit (un comparador) como se ve en la figura 2.4. La salida se vuelve a convertir a analógica con un DAC de un bit y se resta de la entrada después de pasar por un integrador. La forma de la señal analógica se transmite de esta forma: un ”1ïndica que se ha producido una desviación positiva desde la última muestra, un ”0ïndica que se ha producido una negativa.

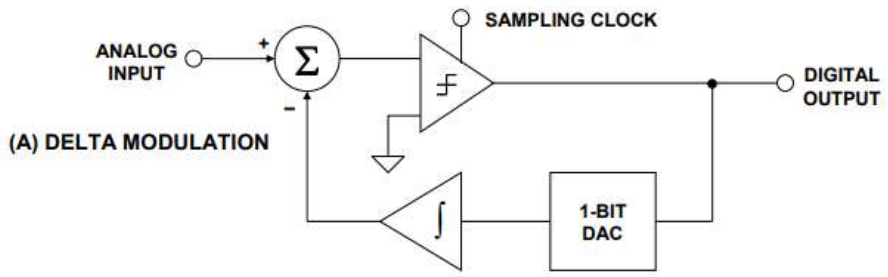


Figura 2.4: Circuito básico de la modulación delta. Tomado de [6].

El problema principal de esta modulación es que si la señal analógica cambia demasiado deprisa, el cuantificador es incapaz de seguirle el ritmo (la pendiente máxima del sistema es menor que la pendiente de la señal). Esto se reduce usando mayores frecuencias de muestreo, la modulación

delta requiere típicamente f_s 20 veces mayores que la frecuencia máxima, a diferencia de la f_s dos veces mayor que requiere un sistema normal solo afectado por Nyquist [6].

La modulación Sigma-Delta reorganiza los bloques (fig. 2.5) para proporcionar una codificación parecida, pero con características de señal mucho mejores. Consigue una implementación más simple que además aleja el ruido de cuantificación de las señales de interés (*noise shaping*).

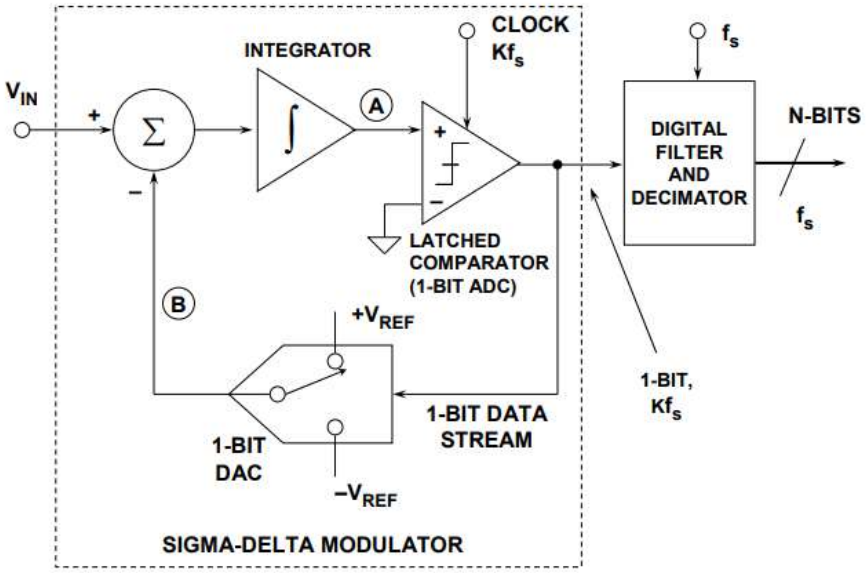


Figura 2.5: Circuito básico de un ADC Sigma-Delta de primer orden. Tomado de [6].

En el domino frecuencial, el integrador actúa como un filtro analógico. Este filtro tiene un efecto paso-bajo en la señal, y uno paso-alto en el ruido de cuantificación. Es decir: a baja frecuencia la salida es mayormente la señal, sin componente de ruido, y a alta frecuencia es principalmente ruido; es así como efectúa el *noise shaping*.

La razón de la frecuencia de (sobre-) muestreo respecto a la de la señal se conoce como *Oversampling Ratio* (OSR), y para un mismo OSR, si se usan más etapas de integrador y sumador en el Sigma-Delta se puede conseguir aún más atenuación del ruido a baja frecuencia (mayor *noise shaping*).

2.1.4. LVDS

La interfaz de señal diferencial de bajo voltaje (*Low-Voltage Differential Signal*, LVDS) es un estándar de transmisión de datos caracterizado por su alta velocidad, bajo consumo y resistencia al ruido mediante pares de señal complementarios. Sus especificaciones eléctricas: bajo voltaje, alta impedancia diferencial y excelente rechazo al modo común; contribuyen a su eficacia para mantener la integridad de la señal en largas distancias [7].

Esta interfaz se adhiere a dos estándares, ANSI TIA/EIA-644 y IEEE 1596.3–1996. Muchos controladores LVDS están construidos con corriente constante, por lo que el consumo de energía no aumenta con la frecuencia de transmisión [8].

2.1.5. SPI

El bus de interfaz de periféricos serie (*Serial Peripheral interface*, SPI) es uno de los protocolos más usados entre microcontroladores y otros circuitos integrados. Es una interfaz síncrona, full-dúplex y basada en un nodo principal y uno o varios subnodos. Los datos del nodo principal (denominado maestro) o secundario (denominado esclavo) se sincronizan en el flanco ascendente o descendente del reloj. Tanto el nodo principal como el subnodo pueden transmitir datos al mismo tiempo. La interfaz SPI puede ser de 3 o 4 cables, siendo la de 4 cables la interfaz más popular [9].

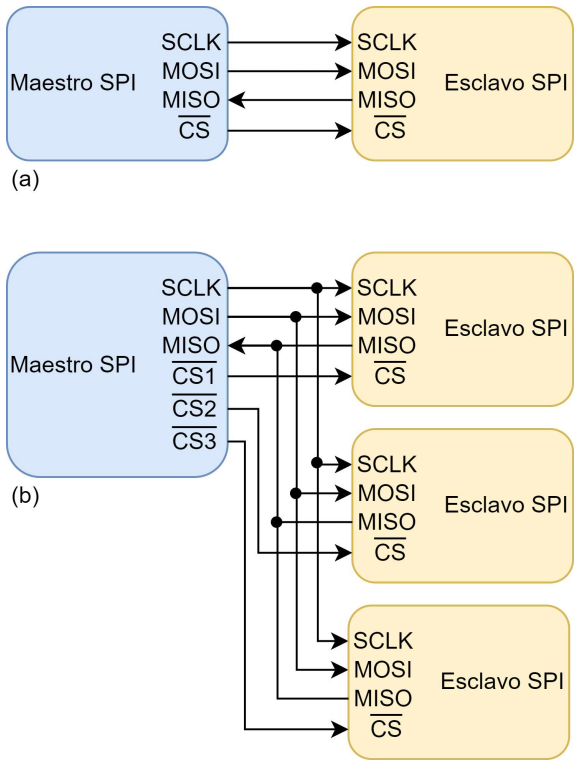


Figura 2.6: Dos topologías de bus SPI. (a) muestra un maestro SPI conectado a un solo esclavo (topología punto a punto). (b) muestra un maestro SPI conectado a múltiples esclavos.

Las cuatro líneas de señales (fig. 2.6) son las siguientes:

- La señal de reloj (SCLK) enviada por el maestro a todos los esclavos
- Una línea de transmisión de datos desde el maestro a los esclavos (MOSI)
- Una línea de transmisión de datos desde los esclavos al maestro (MISO)
- Una señal de selección de esclavo (CS) para cada esclavo, usada para seleccionar con cuál se comunica el maestro

El protocolo de comunicación SPI tiene siempre un solo maestro, que es el que inicia la comunicación con los esclavos. Cuando desea enviar datos a un esclavo y/o solicitar información de este, lo selecciona activando su línea de CS y acciona la señal del reloj a una frecuencia utilizable

por ambos. Envía datos por la línea MOSI a la vez que recibe otros enviados por el esclavo por la MISO. En general el protocolo SPI usa algo más de recursos (número de líneas, complejidad del circuito...) que otras alternativas como I2C, pero a cambio ofrece una velocidad de transmisión muy alta y una implementación sencilla [10].

2.2. Descripción del dispositivo específico

El elemento a verificar es un dispositivo de 210 canales compuestos por un ADC Sigma-Delta incremental de 16 bits y un espejo de corriente de polarización (*bias current*) para un APS. Este chip ha sido diseñado para ser conectado directamente a la lámina APS. Sus funciones principales son:

- Suministrar corriente de polarización a los seguidores de fuente del APS
- Convertir los voltajes de salida del APS de la anterior integración (n-1) en todos los canales (en paralelo)
- Proporcionar la lectura de los datos digitales (n-2) por una (modo simple), dos (modo doble) o cuatro (modo cuádruple) interfaces LVDS
- Generar la temporización en el chip para señales de control internas configurables mediante SPI
- Medir la temperatura del dado (*die*) y enviar la lectura por la interfaz LVDS
- Aislar totalmente las tensiones de alimentación digitales y analógicas mediante el uso de tecnología analógica aislada de alto rendimiento

Cada uno de los 210 canales paralelos admite la operación por voltaje de entrada. Este modo usa una interfaz de píxeles activos (API) para conectar el ROIC con el APS. La API consta de una fuente de corriente de bajo ruido y de dos voltajes de *reset* diferentes. Las tensiones de referencia y las corrientes de polarización de referencia se generan globalmente en el bloque de referencia (REF). El voltaje de referencia maestro y la corriente de polarización se ajustan durante la clasificación de obleas (*wafer sort*). Los diferentes voltajes de referencia se pueden seleccionar mediante registros configurables por la SPI.

El tiempo desde que se dispara la conversión analógico-digital mediante un pulso en la señal “trig”, hasta que esta termina, está determinado por los siguientes factores:

- La frecuencia del reloj del sistema
- La espera desde que se dispara el ADC hasta que comienza realmente la conversión, programada con los registros ADC_start_1/2/3 (usa uno de los tres dependiendo de la duración del pulso en trig).
- El OSR, que puede ser configurado a 32, 52, 62, 74 o 88.
- El tiempo de compensación, que activando el registro ADC_offset_comp_long puede aumentarse de 4 a 6 ciclos.

Los valores digitales que salen del filtro posterior al ADC se transfieren a través de una o varias salidas LVDS para reducir la frecuencia de comunicación. En el modo por defecto, con dos salidas, la mitad de los canales usa una salida LVDS y la otra mitad usa la segunda salida. Se pueden activar dos pares LVDS adicionales para aumentar la velocidad de comunicación. La interfaz también proporciona una salida de reloj LVDS de eco (reflejo de la señal de reloj recibida en el chip, DCLK, tras pasar por los mismos retardos que los datos, y sincronizada por tanto con estos).

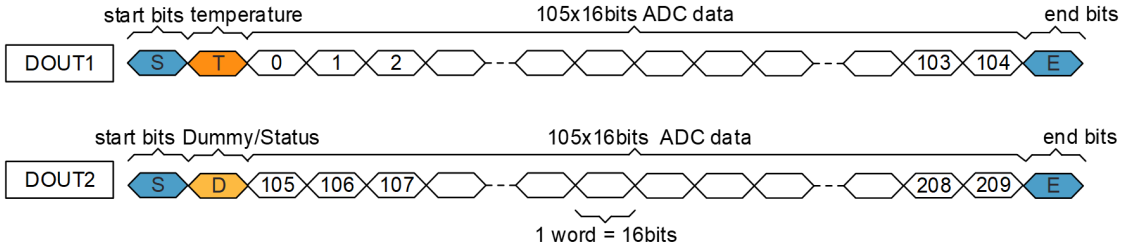


Figura 2.7: Trama de datos durante la transmisión de las lecturas por LVDS en modo doble.

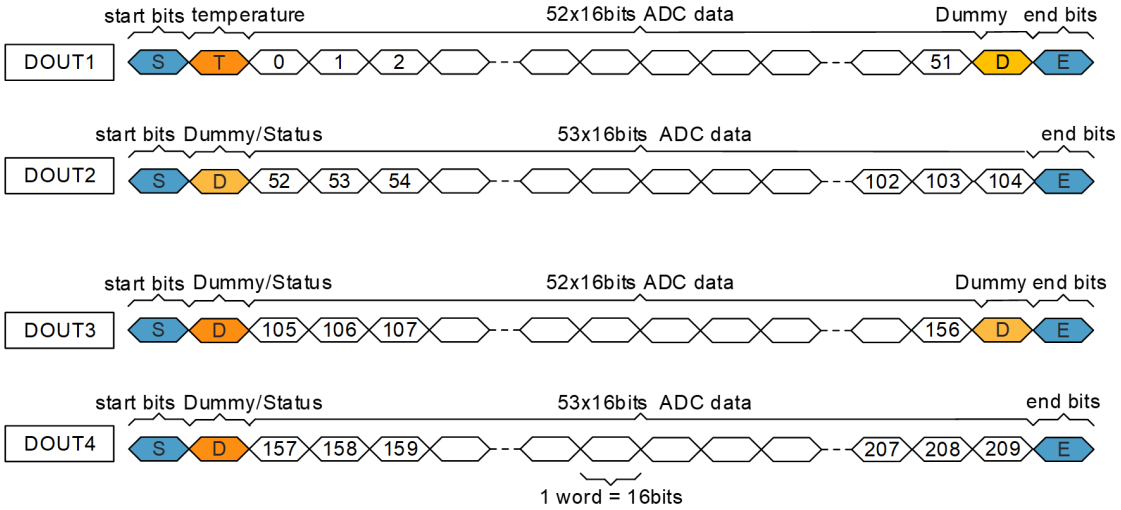


Figura 2.8: Trama de datos durante la transmisión de las lecturas por LVDS en modo cuádruple.

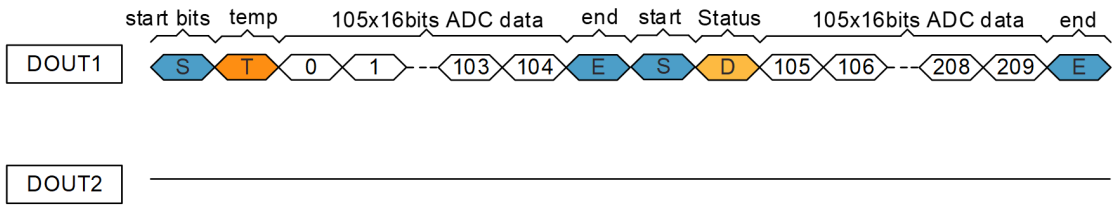


Figura 2.9: Trama de datos durante la transmisión de las lecturas por LVDS en modo simple.

La palabra S (*start*) es un registro (lvds_startword) para reconocer el inicio de la transmisión, mientras que la E (*end*) es el CRC (verificación por redundancia cíclica), un valor de verificación

basado en el residuo de una división de polinomios repetidamente con todos los datos. Esto permite saber si los datos recibidos están corruptos, al no concordar el CRC recibido con el que va calculando el receptor por su cuenta. Se utiliza un CRC-16 con el polinomio $x^{16} + x^{15} + x^2 + 1$.

Los bits finales se calculan a partir del valor de reinicio de CRC, que es 0xFFFF, y los bits de datos. Es decir, las palabras con datos van alterando el CRC hasta que termina esa línea (que puede estar constituida de 52 o 53, 105 o 210 palabras, dependiendo del modo y la línea que sea), y una vez termina se usa lo que haya quedado como palabra final.

Después de la palabra *start* siempre hay otra palabra antes de comenzar la transmisión de datos. En la primera línea es la que indica la temperatura en el ROIC, y en las líneas adicionales puede ser una cualquiera (*dummy*) o la de *status*. Esta última (lvds_status_word) es un registro que contiene en cada bit la paridad de cada registro de la memoria, lo que permite verificar que esta no está corrupta. Al ser un registro (forma parte de REG1), también se puede leer mediante comunicación SPI.

El sensor de temperatura permite monitorizar el cambio de temperatura en la unión. Además, el núcleo digital consta de una lógica de control, una memoria programable de una sola vez (OTP) y registros reprogramables (RAM) para definir las diferentes configuraciones. Tanto los OTP como los reprogramables se encuentran ordenados dentro de un mapa de registros. Mediante la SPI se puede leer o escribir cada uno de estos registros (tabla 2.2).

Nombre	**Tamaño (bits)**	**Tipo**	**Descripción**
REGID	40	OTP	Información del chip (ID)
REG1	40	RAM	Registro 1 (señales de configuración)
REG2	40	RAM	Registro 2 (señales de configuración)
REG3	40	OTP	Registro 3 (ajustes de las referencias)
REG4	40	RAM	Registro 4 (señales de configuración)
REG5	40	RAM	Registro 5 (señales de configuración)
REG6	40	RAM	Registro 6 (señales de configuración)
REGCS	210	RAM	Selección de canales individuales, permite aplicar modos de test solo a los seleccionados
REGTM	40	RAM	Registro de modos de test (para tests de validación y producción)
SFR	40	RAM	Registro de funciones especiales (control de los fusibles OTP)
ReadADC	210x16	R	Lecturas ADC de cada uno de los canales

Tabla 2.2: Tabla de registros del dispositivo a verificar.

En esta tabla los dos registros de tipo OTP funcionan con una P2RAM. Se pueden leer indefinidamente, pero para escribirlos es necesario habilitar esta memoria desde el registro de funciones especiales, y una vez esta se quema (se manda la orden de fundir los fusibles), estos ya no pueden

ser reprogramados, y al activar un bit de *load* (carga) se sobrescribe cualquier valor que se haya escrito después en estos registros con los valores "quemados".

El último registro, ReadADC, solo se puede leer. Los otros en general tienen acceso tanto de lectura como de escritura indefinido, pero hay algunos bits excepcionales que solo se pueden leer. Por ejemplo, si se lee un registro y este tiene bits no usados (sin función), se leerá un 0 en ellos, pero al escribir el registro estos bits no cambiarán. Tampoco se pueden escribir los bits de paridad de las líneas LVDS, aunque sí se pueden leer.

2.3. Patrones de operación

Desde que llegan al ROIC hasta que son leídos por el maestro, los datos pasan por tres fases: integración, conversión y transmisión LVDS. Estas tres fases pueden ocurrir completamente en paralelo, si a la vez que se está integrando un conjunto de datos (n), se está convirtiendo el anterior (n-1), y se está transmitiendo por las salidas LVDS el anterior al anterior (n-2). También puede ocurrir en serie: se integra, luego se convierte a digital el conjunto integrado, y hasta que no se ha terminado de transmitir no se empieza a integrar el siguiente. Por último, se puede usar una combinación personalizada (fig. 2.10).

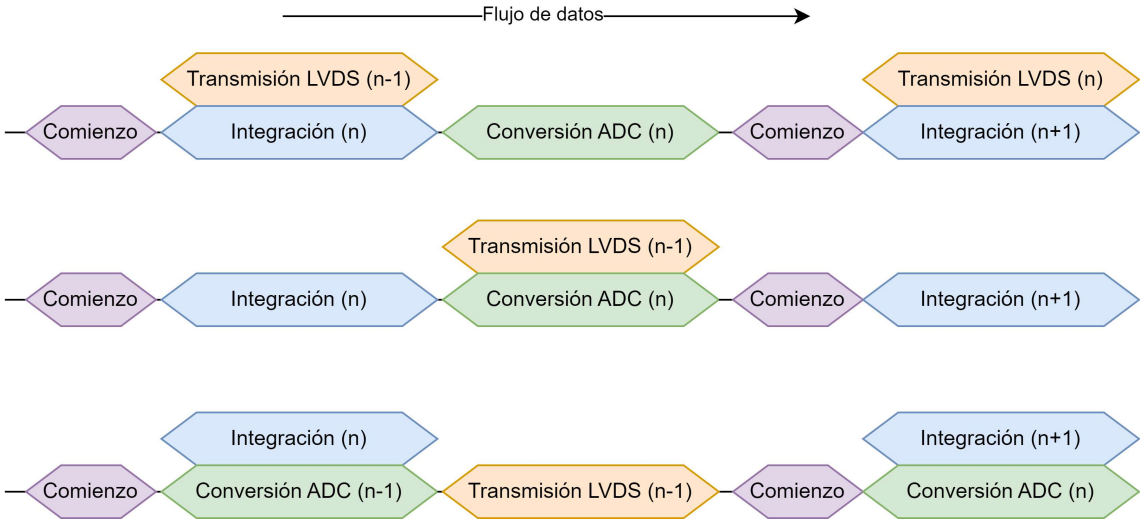


Figura 2.10: Tres tipos de secuencias parcialmente en serie y paralelo de integración, conversión ADC y transmisión de las lecturas por LVDS.

Lo más típico es totalmente paralelo, especialmente cuando se busca que cada operación dure poco (más velocidad). Usando estas combinaciones, la frecuencia de reloj, el OSR y el ajuste del convertidor, entre otras cosas, las especificaciones definen tres modos de operación: máxima velocidad (que conlleva además el máximo consumo), media potencia y baja potencia. Sin embargo, la verificación no debe observar demasiado estas categorías, sino que debe asegurar el funcionamiento para todas las posibilidades permitidas.

Capítulo 3

Metodología de verificación

3.1. Verilog y SystemVerilog

Verilog es el lenguaje de descripción de hardware más utilizado y respaldado por la industria, junto con VHDL. Destaca por su simplicidad, facilidad de uso, y compatibilidad con el lenguaje de programación C. Este lenguaje fue mejorando con los años hasta convertirse en un estándar.

La necesidad de ampliar las funcionalidades de este lenguaje, especialmente en temas de verificación, provocó que apareciera SystemVerilog, un "superconjunto" de Verilog que lo incluye a este y otras extensiones.

En el diseño de nivel de transferencia de registro (RTL), SystemVerilog es una extensión de Verilog-2005, todas las funciones de este están disponibles en SystemVerilog. En verificación, en cambio, SystemVerilog utiliza técnicas de programación orientada a objetos y está más estrechamente relacionado con Java que Verilog. Estas construcciones generalmente no son sintetizables.

SystemVerilog introduce por ejemplo funciones para tener pseudo-aleatoriedad. Con estas funciones se pueden generar variables aleatorias pero a partir de una semilla determinada, con lo que se puede repetir un test con los mismos valores fácilmente.

3.2. Bancos de prueba y cobertura

El código RTL define el diseño del dispositivo a probar (*Device Under Test*, DUT). El alza en complejidad de los diseños digitales provocó que la verificación funcional introdujera herramientas y metodologías para crear test, definidos en los bancos de trabajo, más automatizados y dinámicos. En este trabajo normalmente se ha instanciado en el banco de pruebas el bloque a verificar, los modelos adicionales que sean necesarios para simular alguna funcionalidad del sistema, y un módulo tester encargado de ir generando los estímulos que van al DUT (sus entradas) e ir leyendo sus salidas y comprobando si son las que se esperaba. En la verificación, el objetivo es probar el DUT minuciosamente y asegurarse de que no tiene fallos funcionales. Mientras se hace esto, debe haber una forma de medir la completitud de la verificación [11]

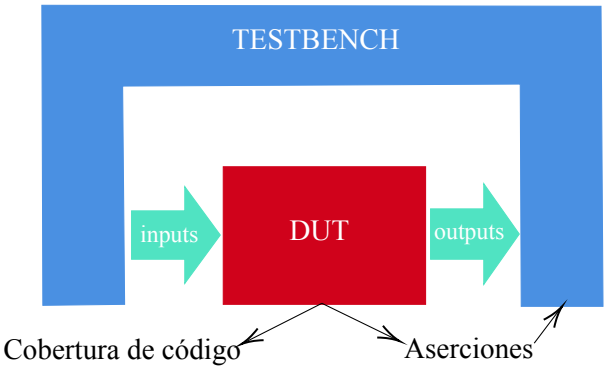


Figura 3.1: Esquema de la relación entre DUT y banco de pruebas en la verificación con Verilog.

Las herramientas de cobertura de código son una primera medida. No tienen conocimiento de la funcionalidad del diseño, pero informan de la ejecución del código línea a línea, garantizan que cada línea del DUT se ha ejecutado al menos una vez, o de si todas las transiciones posibles de bits de las señales han ocurrido. La segunda medida es la cobertura funcional, que se divide en dos ítem:

- Cobertura del protocolo: medida en que se ha ejercitado el DUT para todas las condiciones válidas e inválidas del diseño.
- Cobertura del plan de test: mide la exhaustividad del banco de trabajo, es decir, si se han creado todos los posibles tamaños de paquete, si se han escrito o leído todas las direcciones de memoria posibles...

3.3. Aserciones

Las especificaciones funcionales del diseño permiten definir una serie de propiedades que se deben cumplir. Las aserciones son las descripciones de estas propiedades del diseño, son “afirmaciones” que se deben cumplir (ser verdaderas), o bien en forma de comportamientos que se deben cumplir cuando se espera, o bien como prohibiciones que no deben ocurrir. Hay dos tipos de aserciones:

- Aserciones inmediatas: basadas en la semántica de los eventos de simulación, evaluaciones inmediatas atemporales que se colocan en bloques procesales de código. Se usan solo en simulaciones dinámicas.
- Aserciones concurrentes: basadas en ciclos de reloj, se evalúan en los flancos del reloj a partir de los valores muestreados de las variables involucradas. Se usan tanto en simulaciones dinámicas como estáticas.

La funcionalidad de un modelo se representa como una combinación de eventos lógicos. Los eventos pueden ocurrir durante varios ciclos de reloj o ser simples expresiones booleanas que se cumplen en el mismo flanco. Estos eventos se definen en forma de secuencias (*sequence*). Las secuencias se pueden combinar lógica o secuencialmente para describir comportamientos más

complejos, llamados propiedades (*property*). Es la propiedad lo que se afirma como una aserción (*assert*) para que se cumpla durante una simulación (fig. 3.2).

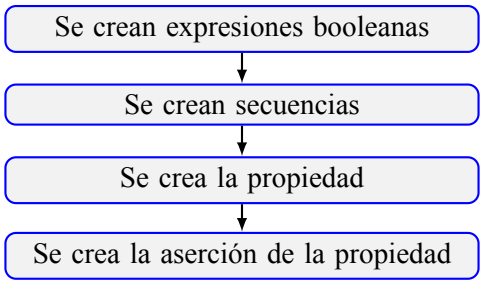


Figura 3.2: Diagrama de los pasos para crear una aserción en SystemVerilog.

A continuación se muestra un ejemplo de aserción. Permanece activada siempre que no haya *reset* y los comprobadores estén activados (functCheckers_enable a 1). Se evalúa cada flanco positivo de reloj. Si cuando se evalúa "inA" está a nivel alto y "in_disable" a nivel bajo, se comprueba inmediatamente que se cumpla la secuencia "seqA". Esta se cumple si "a" está a 0 y al menos un ciclo después pasa a nivel alto (puede ser muchos ciclos más, $ indica que no hay límite). Debe permanecer así tres ciclos, y un ciclo después, "b" debe estar a nivel alto. Por último, entre 5 y 10 ciclos después, "b" debe desactivarse. Como se puede ver en el código, si no se cumple la propiedad, salta un error en la simulación con un mensaje indicando la causa.

```
sequence seqA; // declaración de la secuencia
 !a ##[1:$] a[*3] ##1 b ##[5:10] !b;
endsequence
property pA; // declaración de la propiedad
  @(posedge clk) disable iff(~`RSTN | !functCheckers_enable)
    (inA & !in_disable) |-> (seqA);
endproperty
assert property (pA) // comprobación de que la propiedad se cumple
    else $error("Error, no se cumple pA");
}
```

3.4. Covergroup

Un *covergroup* es un tipo de *construct* que encapsula las especificaciones de un modelo de cobertura. Es un tipo definido por el usuario, por lo que una vez se ha definido, se pueden crear múltiples instancias en diferentes contextos. Cada especificación del *covergroup* puede contener estos componentes [12]:

- Un evento de reloj que sincroniza el muestreo de puntos de cobertura
- Un conjunto de puntos de cobertura (*coverpoint*)
- Cobertura cruzada entre estos puntos (*cover_cross*)
- Argumentos formales opcionales

- Opciones de cobertura

Los *coverpoint* especifican expresiones integrales que se deben cubrir. Cada punto de cobertura incluye una serie de *bins* asociados con los valores muestreados o transiciones de valores que se esperan de la expresión. Se puede dejar que el lenguaje los cree automáticamente o definirlos explícitamente, si se desean unos concretos. La evaluación del *coverpoint* ocurre cuando el *covergroup* al que pertenece es muestreado a través de un bloque procesal.

La especificación de cobertura cruzada dentro de un *covergroup* consiste en usar un *cross construct* para medir, entre dos o más *coverpoint*, la cobertura de todas las posibles combinaciones de todos los *bins* de todos los puntos. Estos puntos deben pertenecer siempre al mismo grupo que la especificación.

En el siguiente ejemplo, se muestrea cada ciclo de reloj un contador de diez bits. Sin embargo, en lugar de cubrir los 1024 valores posibles que puede tener esta cuenta, se especifica un cuatro dentro de los corchetes de *range*, lo que indica que se debe dividir el rango que se defina en cuatro secciones de igual tamaño, y únicamente se debe verificar que la cuenta esté al menos una vez en cada una de las secciones.

```
covergroup cgr_counter; // Grupo de cobertura de un contador
  cpn_enable: coverpoint en; // habilitador del contador
  cpn_count: coverpoint count { // cuenta de diez bits
    bins range[4]  = {[0:1023]}; // rango de cero a mil veintitrés
  }
endgroup
cgr_counter cgr_counter_inst = new();

initial
begin
  forever begin
    @(posedge clk);                 // se muestrean cada ciclo de reloj
    cgr_lvds_config_inst.sample();  // los coverpoint del covergroup
  end
end
}
```

Por lo tanto, este *coverpoint* se cubrirá al 100 % solamente si se muestrea al menos una vez la cuenta con un valor entre 0 y 255, al menos una vez con un valor entre 256 y 511, al menos una vez con un valor entre 512 y 767, y por último al menos una vez con un valor entre 768 y 1023. También se podría haber especificado que se quiere además cubrir los valores máximo y mínimo en particular, o cualquier otro caso de especial importancia.

3.5. Metric Driven Verification

La verificación basada en métricas (*Metric Driven Verification*, MDV) es una disciplina que permite conseguir una buena cobertura funcional cuando los diseños tienen espacios de estados tan grandes que solo se puede simular una fracción de todas las combinaciones posibles [13]. La MDV se guía por las especificaciones funcionales, en lugar de por la implementación del diseño. Se analizan las especificaciones como una jerarquía de características en un plan de verificación,

donde se puede comprobar que cada característica cumple con una especificación mediante alguna medida. Esta serie de medidas representan por tanto la cobertura funcional del diseño.

Inicialmente, se crean bancos de pruebas para ejercitar el diseño, comprobar su funcionalidad y medir coberturas. Después, se agregan capas de automatización para poder ejecutar una gran cantidad de simulaciones con perturbaciones aleatorias. La sesión con todos los grupos de test que se simulan un número de veces se denomina regresión. Se suma toda la cobertura y se compara con el plan de verificación (diagrama de la figura 3.3). Por último, se dedica un esfuerzo adicional en las áreas con menos cobertura, con el fin de conseguir un resultado general equilibrado.

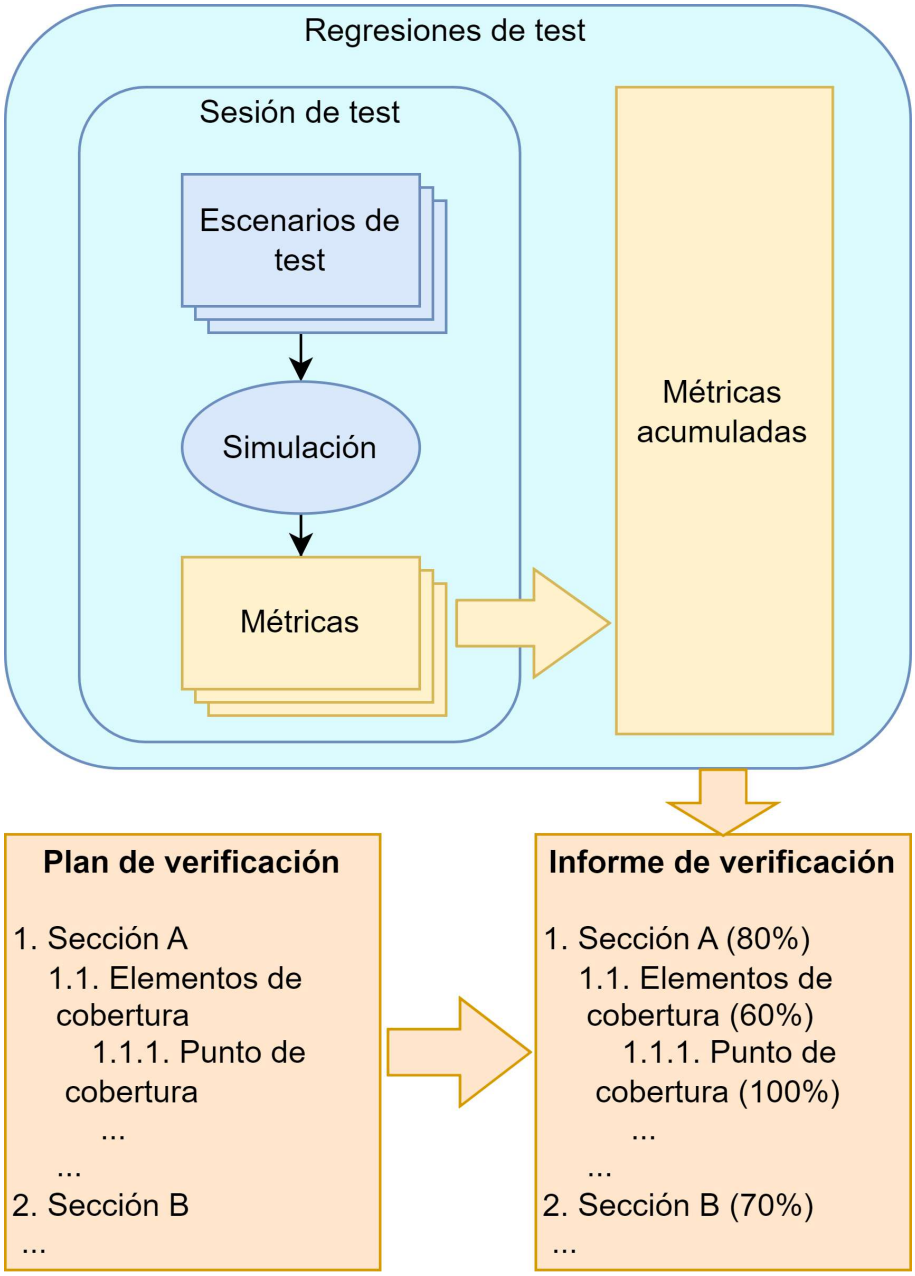


Figura 3.3: Metodología de la MDV mediante gestión de regresiones.

Las capas de automatización deben poder generar estímulos aleatorios restringidos de forma que las simulaciones exploren distintos comportamientos que cubran el espacio funcional. A la vez, el banco de pruebas debe auto-verificarse, en lugar de requerir una inspección manual de los resultados.

Capítulo 4

Verificación

4.1. Plan de verificación

Partiendo de la documentación con la información y especificaciones, se han condensado todas las funcionalidades y restricciones temporales del DUT en una lista de elementos a comprobar para asegurarse de que se cumplen perfectamente. Esto es el plan de verificación, y en él se va confirmando que cada ítem se comprueba en algún test, sea con aserciones, covergroup u otro método válido. A continuación se muestra un resumen de este plan, los elementos están categorizados por bloques.

1. **ADC**
    - 1.1 **ADC trig**: pulsos en trig detectados durante una conversión del ADC son ignorados.
    - 1.2 **ADC start**: el comienzo de la conversión ocurre el número de ciclos correcto después de la caída de trig, teniendo en cuenta los registros ADC_start1/2/3 y ADC_offset_comp_long.
    - 1.3 **ADC OSR config**: la conversión del ADC dura ADC_OSR_1/2/3 + 2 ciclos. En este caso y el anterior se debe escoger el registro 1, 2 o 3 dependiendo de la duración del pulso trig.
    - 1.4 **ADC data**: la conversión termina en el tiempo adecuado y después los datos están listos para ser transmitidos.
    - 1.5 **ADC power down ch_sel 1**: si un canal tiene su ch_sel (que viene de REGCS) activado, entonces su señal de apagado no lo puede estar.
    - 1.6 **ADC power down ch_sel 0**: si, en cambio, un canal tiene su ch_sel desactivado y el registro general ADC_pwr_dwn está a 1, entonces su señal de apagado debe estar activada.
    - 1.7 **ADC config covergroup**: grupo de cobertura con las posibles configuraciones del ADC.
    - 1.8 **ADC conectivity**: conectividad del bloque ADC.
    - 1.9 **ADC supplies**: los ADC tienen alimentaciones separadas (VDDA_B y VSSA_B).
2. **API**

2.1 **API ch_sel 0 curr**: si ch_sel está desactivado, la fuente de corriente no puede estar disponible en ese canal.

2.2 **API BIASMIR**: si API_ON es 0 (un registro que habilita este bloque de entrada de píxeles activos), una señal relacionada con el espejo de corriente también debe ser 0.

2.3 **API bias OFF**: si API_ON es 0, la fuente de corriente tampoco puede estar disponible.

2.4 **API bias ON**: si API_ON es 1, el modo API y la fuente de corriente deben estar activados.

2.5 **API ON 0**: Si API_ON es 0, los interruptores del bloque deben estar abiertos.

2.6 **API bias filter imp**: en API_BIASMIR, la selección de impedancia para el filtro de generación de la polarización del API es proporcional al valor del registro correspondiente.

2.7 **API connectivity**: conectividad del bloque API.

3. **Current mirrors**

3.1 **Curr mirr power down**: los espejos de corriente del ADC se ponen en modo apagado si los registros ADC_pwr_dwn o sobrescritura a ceros en ch_sel están activados.

4. **Digital**

4.1 **Dig OTP**: los registros OTP funcionan correctamente.

4.2 **Dig regs POR**: los registros de configuración se restablecen solo por la señal POR, una señal que se activa al reiniciar la alimentación (VDD).

4.3 **Dig regs default**: los valores por defecto de los registros son los adecuados.

4.4 **Dig SPI**: comunicación SPI funciona correctamente.

4.5 **Dig trig length**: distintas longitudes de trig consecutivas con un tiempo mínimo.

4.6 **Dig ch_sel_i**: la señal de selección que tiene cada canal (ch_sel_i) reacciona correctamente a la señal de sobrescritura a unos y a la de sobrescritura a ceros.

4.7 **Dig RST sync**: RST es síncrono con el reloj del sistema.

4.8 **SCLK coverpoint**: cobertura de las posibles frecuencias de SCLK (2 a 15 MHz).

4.9 **Dig connectivity**: conectividad del bloque digital.

5. **Dummy APS**

5.1 **Dummy APS Vrstcol1**: establece el modo de test del dummy APS a Vrstcol1, aplica distintos niveles en Vrstcol1 y debería generarse una imagen uniforme (en este modo las entradas se conectan al pin Vrstcol1).

5.2 **Dummy APS Vrstcol1 ch**: lo mismo que el anterior pero deshabilitando algunos canales (imagen uniforme en los canales restantes).

5.3 **Dummy APS odd**: establece el modo de test tm_dummy_aps_odd y debería generarse una imagen cebra, con los canales impares conectados a VDD y los pares a Vrstcol1.

5.4 **Dummy APS even**: establece el modo de test tm_dummy_aps_even y debería generarse una imagen cebra, con los canales pares conectados a VDD y los impares a Vrstcol1.

6. **Level shifters**

6.1 **Lev shift connectivity**: conectividad del bloque cambiador de nivel.

7. **LVDS**

7.1 **LVDS frame double**: en modo doble una trama de datos consiste en 108 palabras (ver fig. 2.7).

7.2 **LVDS frame quad**: en modo cuádruple una trama de datos consiste en 56 palabras (ver fig. 2.8).

7.3 **LVDS MSB**: el MSB de cada palabra es enviado primero.

7.4 **LVDS data_start**: los pulsos de la señal data_start mientras una transmisión de datos está en curso son ignorados.

7.5 **LVDS zebra quad**: test imagen cebra leyendo por las cuatro salidas LVDS.

7.6 **LVDS idle**: el bloque LVDS se puede poner en modo inactivo y diferentes palabras *idle* para este modo pueden ser programadas.

7.7 **LVDS frame single**: en modo simple una trama de datos consiste en 216 palabras (ver fig. 2.9).

7.8 **LVDS data DCLK**: ocurre transmisión de datos cada flanco de subida y de bajada del reloj DCLK.

7.9 **LVDS low single**: en modo simple la segunda, tercera y cuarta salidas permanecen apagadas (polos negativo y positivo a 0).

7.10 **LVDS low double**: en modo doble la tercera y cuarta salidas permanecen apagadas.

7.11 **LVDS double 1**: en modo doble la primera salida transmite la trama correcta.

7.12 **LVDS double 2**: en modo doble la segunda salida transmite la trama correcta.

7.13 **LVDS single 1**: en modo simple la primera salida transmite la trama correcta.

7.14 **LVDS quad 1**: en modo cuádruple la primera salida transmite la trama correcta.

7.15 **LVDS quad 2**: en modo cuádruple la segunda salida transmite la trama correcta.

7.16 **LVDS quad 3**: en modo cuádruple la tercera salida transmite la trama correcta.

7.17 **LVDS quad 4**: en modo cuádruple la cuarta salida transmite la trama correcta.

7.18 **LVDS data stream start**: la transmisión en las salidas comienza, como mucho, 5 ciclos después de que se detecte data_start activado en un flanco de bajada de DCLK.

7.19 **LVDS power down**: si lvds_pwr_dwn está a 1 el bloque LVDS debe estar apagado.

7.20 **LVDS diff low**: cuando no transmiten datos, las salidas LVDS activadas deben estar en bajo diferencial (polo positivo a 0 y negativo a 1). Las desactivadas deben estar totalmente apagadas para reducir el consumo.

7.21 **LVDS echo config**: el reloj echo LVDS está disponible y se puede desactivar con el registro tm_echo_clk_pd. Este reloj es una copia del DCLK de entrada pero retrasada para ir sincronizada con la transmisión LVDS.

7.22 **LVDS echo clock**: el reloj echo clock sigue con precisión a DCLK (si no está desactivado).

7.23 **LVDS config covergroup**: grupo de cobertura de las posibles configuraciones LVDS.

7.24 **DCLK coverpoint**: se usan todas las posibles frecuencias del DCLK (167 a 250 MHz).

7.25 **LVDS connectivity**: conectividad del bloque LVDS.

8. **Pads**

   8.1 **Pads connectivity**: conectividad de los pads.

9. **POR**

   9.1 **POR VDD**: POR conmuta en los flancos de subida o de bajada de VDD.

   9.2 **POR reset**: el bloque digital, ADC y LVDS son reiniciados por la misma señal, una función de RST y RCLK y POR, teniendo está última más prioridad que RST.

   9.3 **POR sync**: el flanco de subida de POR (fin del reset) está sincronizado al segundo flanco de bajada de RCLK.

   9.4 **POR connectivity**: conectividad del bloque POR.

10. **Reference**

    10.1 **Ref ADC_ibias**: al configurar ADC_ibias se observan 4 niveles de corriente distintos, proporcionales al valor del registro.

    10.2 **Ref REF_Iapi**: al configurar REF_Iapi se observan 64 niveles de corriente distintos, proporcionales al valor del registro.

    10.3 **Ref bandgap trim**: recorte de la tensión de banda prohibida (*bandgap*) de referencia para el filtro.

    10.4 **Ref bandgap gain**: ajuste de la ganancia de esta tensión *bandgap*.

    10.5 **Ref Vref_bg_filter**: al configurar el registro de selección de impedancia para esta tensión *bandgap*, se observa una impedancia proporcional a lo configurado.

    10.6 **Ref Vref_adc_filter**: lo mismo que antes pero ahora para la impedancia de la tensión de referencia del ADC.

    10.7 **Ref auto-zeroing clock**: el reloj de puesta a cero automática tiene una frecuencia de RCLK/FDIV.

    10.8 **Ref bias curr trim**: recorte de la corriente de polarización.

    10.9 **Ref power down**: el bloque de referencia está en modo apagado si se activa la señal REF_pwr_dwn.

    10.10 **Ref pad_vref_ext**: activar un registro desconecta la tensión *bandgap* interna y permite el forzado externo mediante el pad pad_vref_ext.

    10.11 **Ref pad_vref_adc**: activar un registro desconecta la tensión del ADC interna y permite forzado externa mediante el pad pad_vref_adc.

    10.12 **Ref clock**: la frecuencia del reloj de referencia CLK_REF es RCLK/FDIV.

    10.13 **Ref connectivity**: conectividad del bloque de referencia.

11. **Sensor**

    11.1 **Sensor en mirror**: comprueba que el sensor se puede activar/desactivar con la señal Sensor_en_mirror.

12. **Temperature sensor**

12.1 **Temp clock**: el reloj del sensor de temperatura tiene la frecuencia RCLK/FDIV.

12.2 **Temp data TEMP**: los datos de temperatura sensados se almacenan en el registro de configuración TEMP.

12.3 **Temp CONT**: activa el registro modo continuo y comprueba que TEMP se actualiza a los 4092 ciclos.

12.4 **Temp TRIG**: activa el registro de modo disparo y comprueba que el modo continuo es ignorado, a los 4092 ciclos se actualiza TEMP y el registro se desactiva.

12.5 **Temp power down**: el sensor de temperatura está apagado si TS_pwr_dwn es 1.

12.6 **Temp connectivity**: conectividad del bloque de temperatura.

13. **Timing block**

13.1 **Timing block**: verificación del bloque de temporización.

14. **Top**

14.1 **Top lev shift**: verificación de los cambiadores de nivel.

14.2 **tdig qodc and qodi**: verificación de las salidas para modos de test tdig_qodc_out y tdig_qodi_out.

15. **Track and Hold**

15.1 **TnH selCsample 0**: si TnH_SerialMode es 1 y TnH_selCsample es 0, los interruptores TnH_s|r1 deben estar abiertos.

15.2 **TnH selCsample 1**: si TnH_SerialMode es 1 y TnH_selCsample es 1, los interruptores TnH_s|r0 deben estar abiertos.

15.3 **TnH api bypass**: el interruptor de derivación (*bypass*) debe estar cerrado en modo API solo si TnH_api_bypass es 1.

15.4 **TnH S_bypass**: si el interruptor *bypass* está cerrado, los S_s|r* (0 o 1) conectados a IN_API están abiertos.

15.5 **TnH S_rst**: si el interruptor *bypass* está cerrado, el de rst no puede estar cerrado.

15.6 **TnH SerialMode 0**: si TnH_SerialMode es 0, los interruptores TnH_s0_api y TnH_s|r1 deben conmutar.

15.7 **TnH 0 and 1 non overlap**: los interruptores TnH_s0|1 y TnH_r0|1 no pueden coincidir (el 0 de ambos no puede estar activado al mismo tiempo e igual con el 1).

15.8 **TnH OSR**: el número de ciclos que uno de los interruptores TnH_r* (0 o 1) está activados es el del OSR del ADC.

15.9 **TnH rst_in**: TnH_r0|1 y rst_in del ADC no pueden estar activados a la vez.

15.10 **TnH pad_in**: si TnH_api_bypass es 1, el interruptor *bypass* está cerrado y la entrada del ADC es la del pad de entrada (pad_in).

15.11 **TnH connectivity**: conectividad del bloque de rastreo y retención de las señales de entrada.

Algunos de estos elementos se comprueban con test directos, que pueden ser sobre bloques concretos o el sistema completo, mientras que otros se comprueban con aserciones que se evalúan cada flanco de subida del reloj de aserciones, que es igual que el del sistema pero con un retraso de unos nanosegundos. Sin embargo, los que se verifican con aserciones también necesitan bancos de test, puesto que requieren que se excite el chip para que pase por todas las posibles situaciones de fallo y que activan las situaciones concretas de cada aserción.

4.2. Bancos de test

A continuación se exponen los grupos de test principales, donde dentro de cada uno se llevan a cabo una serie de distintos test que tienen en común el bloque de diseño que están estimulando.

4.2.1. Filtro digital

Este banco de test verifica dos módulos, el módulo del filtro que hay en cada canal, pero también un módulo general que sirve de máquina de estados para controlar todos los filtros a la vez. El funcionamiento de este test es el siguiente: se recorren en bucle todos los OSR, mientras se van activando cada ronda las señales apropiadas para hacer funcionar el filtro. Durante toda la conversión, cada ciclo se le da como entrada uno de los valores de una tabla de valores predefinida. El valor se escoge según el ciclo de conversión y uno de los patrones disponibles.

Se han calculado en una hoja de cálculo todas las entradas cada ciclo para ocho patrones distintos. Estos patrones son una distribución de los posibles niveles de tensión analógica en la entrada, desde el mínimo hasta el máximo pasando por algunos valores intermedios. La hoja de cálculo da los valores que tiene que haber en la salida del filtro para cada caso, por lo que el test simplemente proporciona el flujo de bits en la entrada que se recibirían del convertidor Sigma-Delta, y comprueba que en la salida del filtro haya la misma salida que en la esperada según el *golden model* (la hoja de cálculo).

4.2.2. Modelo LVDS

Este es un módulo que simula lo que haría la FPGA o procesador que estuviera conectada al ROIC para leer la transmisión LVDS e interpretar las tramas de datos. Cuenta con una aserción que hace saltar un error si una trama no acaba con el CRC correcto. Hay un banco de trabajo que instancia también un modelo emisor de datos LVDS, y los recibe con este modelo receptor y comprueba que la trama recibida es igual a la enviada, según la configuración LVDS que se esté usando.

Prueba con transmisión por dos salidas LVDS (doble), una (simple) y cuatro (cuádruple). Para comparar las palabras recibidas con las enviadas usa la tarea checkData, como otros bancos de test, que aumenta la cuenta de errores si las dos palabras no son iguales, teniendo en cuenta el número de bits y la máscara, si hay.

4.2.3. Bloque LVDS

Este sí que prueba el bloque LVDS del DUT. Puesto que solo instancia este bloque con el tester, el test genera directamente una variable matriz (*array*) de 210x16 bits que va conectada al LVDS, como si se tratara de la señal que vendría del filtro digital. La lectura de las transmisiones se realiza con la tarea lvds_read, la comprobación de que son correctas con checkCRC. Estas se explican en el apartado de la implementación ya que son aprovechadas también por otros bancos de test.

Además de la transmisión normal, el LVDS tiene otros dos modos: inactivo y test de imagen. Durante el inactivo simplemente se envía siempre la misma palabra por las salidas activadas, esta viene de un registro configurable. El de test de imagen es parecido, pero permite llevar la cuenta de la continuidad de las tramas. Cada línea completa transmitida en este modo el LVDS aumenta un contador interno de 8 bits, y en lugar de la palabra inactiva, se envía una palabra formada por el número de canal en cada caso y el valor de esta cuenta, que vuelve a cero al salir del modo o reiniciarse el chip.

Así, por ejemplo, si se hubieran recibido ya en modo test de imagen 6 tramas completas, en la siguiente trama las primeras palabras recibidas por la primera salida en formato hexadecimal serían:

<palabra start>, <palabra con la temperatura>, <0106>, <0206>, <0306>, <0406>...

Por último, cuando se reinicia el chip y aún no le han llegado datos del ADC, usa como datos iniciales un valor formado por el número del canal (primeros 8 bits el número en reverso y últimos 8 el número normal). Los test que se realizan, que se iteran varias veces con distintas temporizaciones, son estos:

1. Modo salida doble, usando datos iniciales
2. Modo salida doble, usando datos aleatorios
3. Modo salida simple, usando datos iniciales
4. Modo salida simple, usando datos aleatorios
5. Modo salida cuádruple, usando datos iniciales
6. Modo salida cuádruple, usando datos aleatorios
7. Test de la parte SPI para el registro ReadADC
8. Modo test de imagen, salida simple
9. Modo test de imagen, salida doble
10. Modo test de imagen, salida cuádruple
11. Modo inactivo, salida cuádruple
12. Modo inactivo, salida doble
13. Modo inactivo, salida simple
14. Modo test de imagen (de nuevo, para ver si el contador vuelve a cero), salida simple

15. Modo test de imagen, salida doble
16. Modo test de imagen, salida cuádruple

El test SPI consiste en activar una entrada que habilita el acceso, darle una dirección que le pediría el bloque SPI (el número de canal que quiere saber), y el LVDS debe devolver por la salida que iría conectada al SPI la palabra ADC adecuada.

4.2.4. Bloque SPI

Este banco de test verifica el bloque SPI de forma aislada e independiente. Para esto pone a prueba la comunicación SPI y los registros de configuración, tanto los normales como los OTP de la P2RAM. Cualquier test de comunicación SPI usa dos herramientas importantes. Por un lado una referencia de los registros, que contiene los valores por defecto tras un reinicio, y va cambiando al unísono que se escriben los registros del bloque real. Por otro lado unas máscaras de los registros, estas se explican más en detalle en el apartado de la implementación, pero sirven para impedir escribir en los modelos de referencia donde el DUT no debería estar escribiendo sus registros, por mucho que los estímulos que se le están mandando se lo pidan.

De esta forma si por ejemplo en un test se trata de escribir un 1 en una dirección ilegal, al leer luego esa dirección se espera que esa solicitud incorrecta haya sido ignorada. Los test que se efectúan en este banco son los siguientes:

1. **TestRegisterRand**: antes de nada, el test prueba a escribir cada bit de cada dirección, y después lee si se ha escrito. Luego ya inicia el proceso de escribir y leer datos aleatorios de direcciones aleatorias de registros, actualizando la referencia si escribe, pero teniendo en cuenta la máscara. Esto se repite 10 veces.
2. **TestRegisterP2RAMRand**: lo mismo que antes, pero esta vez con la P2RAM habilitada, para lo que se entra al modo reservado para la fundición del chip, mediante una orden de escritura en el registro de funciones especiales (SFR), y se sale al terminar el test. En este caso se usa otra máscara con la que sí se pueden escribir los registros OTP. También se repite 10 veces.
3. **TestP2RamBurn 1**: verifica el quemado de la P2RAM en sus 72 bits fusible, 40 de REGID y 32 de REG3. Para ello, primero habilita de nuevo la P2RAM y escribe datos aleatorios en los bits de estos registros. Luego activa el modo quemado (poniendo a 1 el bit concerniente en el SFR), quema la P2RAM bit a bit (o fusible a fusible), y desactiva este modo. Tras esto trata de sobrescribir la P2RAM, y al leerlos esta sobrescritura debería haber sido ignorada.

   Luego intenta volver a entrar en modo de quemado, pero debería leer del SFR que la P2RAM está bloqueada. Seguidamente inicializa la P2RAM de nuevo, forzando internamente a 0 todos sus fusibles (deshace el quemado por tanto). Ahora habilita la P2RAM, escribe 1 en todos sus registros, vuelve a quemarla y los lee. Por último vuelve al estado inactivo y comprueba que el SFR esté en ese estado.
4. **TestP2RamOTP_LOCK**: comprueba el funcionamiento del bit OTP_LOCK. Tras deshacer el quemado del anterior test y volver a quemar la P2RAM, este bit OTP debe quedar activado, protegiendo a los fusibles no quemados (ceros) de un posterior quemado accidental. Además,

si luego se vuelve a habilitar la P2RAM y se escriben ceros, se leen esos ceros, pero al activar el bit de *load* del SFR, los valores que se habían quemado en los fusibles deben volver a los registros REGID y REG3.

5. **TestP2AnalogRead**: verifica la función de test analógico de la P2RAM. Comienza repitiendo el proceso de escribir valores aleatorios y quemando REGID, y después activa el modo analógico. Al mandar al DUT la instrucción de que quiere escribir en REGID, se activa este modo de test en el que lee secuencialmente en un pin de señal analógica el valor de cada uno de los ocho primeros bits de cada dirección de memoria (va cambiando cada ciclo del reloj SPI).

   Después escribe y quema REG3, y provoca que la P2RAM se ponga en estado bloqueado activando un modo inválido en el SFR. Vuelve a poner el modo analógico y a repetir el test anterior. Al terminar y desactivar este modo, la P2RAM debería volver a estar habilitada (modo fundición), no bloqueada.

6. **TestP2RamBurn 2**: lo mismo que el primer TestP2RamBurn, pero solo para REG3.

7. **TestP2RamBurn 3**: en este caso solo para REGID.

8. **TestErrorInjection**: se intentan inyectar errores aplicando señales de entrada incorrectas. Se intenta de cuatro formas distintas:

   8.1 Accediendo a direcciones de registros inválidas. Se comienza escribiendo datos aleatorios en todos los registros (con la P2RAM habilitada) y leyéndolos. Luego se manda la instrucción de escribir en sitios aleatorios ilegales varias veces, y por último se comprueba que al leer las direcciones correctas los registros siguen correctos

   8.2 Haciendo lo mismo que el caso anterior pero con la P2RAM deshabilitada, por lo que se usa la máscara normal que no espera poder escribir en los registros OTP.

   8.3 Estimulando al azar SDI (MOSI) y SCLK mientras CS_n (CS negado) está alto, y por lo tanto el DUT no está seleccionado para comunicarse con el maestro. Esto debería ser ignorado, y al leer posteriormente los registros no deberían haber cambiado.

   8.4 Mandando un bit de paridad incorrecto con la instrucción. Al intentar leer o escribir errando a propósito el bit de paridad, deberían recibirse del DUT solo ceros, y si luego se lee el registro donde se guarda la palabra de *status* (usando ya un bit de paridad correcto), el bit de error de paridad de la instrucción debería estar activado, pero en la siguiente lectura ya no debería estarlo.

9. **Check ch_sel**: este test verifica el registro REGCS y la salida que genera, ch_sel. Primero prueba si el estado inicial es correcto, y después va escribiendo datos aleatorios y observando que la salida o cualquier lectura SPI posterior cambia acorde.

10. **Check adc_read**: este otro el registro ReadADC. Llena la red de datos ADC en cada canal con su número correspondiente. Después lee toda la memoria, comprobando que ReadADC sea correcto cuando pasa por él, y que REGCS no haya cambiado en este proceso.

4.2.5. Bloque digital

Este test utiliza toda la parte digital del ROIC, incluyendo el bloque LVDS y el SPI, pero también muchos otros, como la red de canales, cada uno con su filtro, el control del ADC, la

temporización, la sincronización de *reset*, los multiplexores de entrada y de salida... El banco de pruebas que incluye el módulo tester y el DUT también instancia un modelo muy simple que devuelve las señales que daría la parte analógica, ADC_model, y modelos de algunos pads, además del lvds_model, que sirve de seguro independiente de que los datos LVDS que se reciben del chip son correctos, aunque el tester ya lee e interpreta los datos en las salidas él mismo, con las tareas de lvds_read y checkCRC.

Los test que ejecuta el módulo tester secuencialmente (los ejecuta todos en la misma simulación, uno detrás de otro) son los siguientes:

1. **Test LVDS**

   Utiliza la tarea set_lvds_readout_data para establecer el TEMP, la palabra *start*, y adc_data_array. Las dos primeras las configura por SPI, pero en este test la matriz de datos ADC se fuerza en la salida del filtro de cada canal. Esto significa que el filtro, módulo de control ADC, y demás, no se usan, el sistema completo se verifica en test posteriores.

   Usa lvds_read para almacenar en data_rec1/2/3/4 los datos recibidos por las salidas 1 a 4, y por último la tarea checkCRC para la comprobación de estos datos. De vez en cuando prueba a mandar pulsos de data_start durante una transmisión, que deberían ser ignorados. Prueba las siguientes secuencias completas de transmisión de datos LVDS:

   1.1 2 salidas LVDS con valores iniciales

   1.2 1 salida LVDS con valores iniciales

   1.3 4 salidas LVDS con valores iniciales

   1.4 2 salidas LVDS con valores aleatorios

   1.5 1 salida LVDS con valores aleatorios

   1.6 4 salidas LVDS con valores aleatorios

   1.7 2 salidas LVDS con modo inactivo activado

   1.8 1 salida LVDS con modo inactivo activado

   1.9 4 salidas LVDS con modo inactivo activado

   1.10 2 salidas LVDS con test de imagen activado

   1.11 1 salida LVDS con test de imagen activado

   1.12 4 salidas LVDS con test de imagen activado (260 veces, lo que verifica el desbordamiento del contador de 8 bits)

2. **Test de registros SPI**. Verifica la comunicación SPI y los registros del DUT. Es similar al banco del bloque SPI, pero todas las tareas varían moderadamente y aprovechan que ahora se tiene casi todo el sistema para probar aspectos más singulares. Las tareas son estas:

   2.1 Comprobar todas las direcciones (valores de reinicio): hace un genPor y lee todas las direcciones de memoria posibles. La tarea genPor se detalla en la implementación.

   2.2 Probar a escribir las direcciones no utilizadas.

   2.3 Comprobar de nuevo todas las direcciones (valores de reinicio): tras el intento de escritura intenta leer de nuevo todas las direcciones posibles, de las no utilizadas debe leer solo ceros.

2.4 TestRegisterRand: lo mismo que el de antes, pero en este caso también se puede comprobar en REG1 que la palabra de status contenga la paridad correcta de todos los registros (el tester tiene un modelo que se mantiene actualizado respecto van cambiando los registros de referencia).

2.5 TestRegisterP2RAMRand: mismas pruebas que el del bloque SPI.

2.6 TestErrorInjection: también intenta provocar los mismos tipos de error.

A diferencia del banco con solo el bloque SPI, aquí escribir los registros tiene unas consecuencias. Para evitar no poder comprobar correctamente el resto de registros, hay cuatro registros que se evita escribir. Como ya tienen sus propios test, esto no bloquea su verificación. Estos registros especiales son el del modo ATPG y el del modo árbol NAND, porque derivan las salidas y las entradas respectivamente, y los dos registros que accionan el sensor de temperatura, pues su medida en el registro TEMP cambiaría a un valor inesperado.

3. **Test SPI de REGCS y ReadADC**:

   3.1 Check ch_sel: en este banco este test es más extenso que antes, incluye la comprobación de dos bits de registro que sirven para sobrescribir todo REGCS con unos o con ceros. También se asegura de que la salida ch_sel esté desactivada durante una lectura o escritura SPI.

   3.2 Check ReadADC: lo mismo que antes, se fijan los valores de los datos que vienen del ADC forzadamente, pero en este caso se hacen varias repeticiones con valores aleatorios. Además, también se prueba que si cuando se ha ordenado leer ReadADC, al llegar al último canal, si no se levanta CS_n, manteniéndose así el chip seleccionado, el SPI debería volver a comenzar desde el canal 0.

4. **Test de la P2RAM**. Hace los mismos test que antes en el bloque SPI:

   4.1 TestP2RamBurn 1

   4.2 TestP2RamOTP_LOCK

   4.3 TestP2AnalogRead

   4.4 TestP2RamBurn 2

   4.5 TestP2RamBurn 3

5. **Test de las salidas**. Escribe algunos registros y observa que las salidas cambien acorde. En concreto verifica:

   5.1 Señales ADC (ADC_clkoff y ADC_ib_conf)

   5.2 Señales API (API_bias_on, API_mode y API_rst_col1/2)

   5.3 La señal dividida del reloj resultado del registro FDIV (FDIV veces más lenta).

   5.4 La señal que indica que la transmisión LVDS ocurre solo por una salida.

   5.5 Señales pwr_dwn (de apagado de distintas partes del chip)

   5.6 Señales REF (para el bloque de referencia)

   5.7 Señales tm y tdig (activar modos de test y test digitales)

   5.8 TnH_bypass_api

También comprueba que el reinicio con RST no cambie la configuración pero con POR sí.

6. **Test de las salidas de modos de test**. Cubre los registros tm (REGTM) no probados en el test de las salidas:

   6.1 Modo de test árbol NAND: activa un registro y comprueba que cambiar data_start, integration, trig y rst afecta a sdo_pad correctamente (se conectan las entradas a este pad mediante puertas NAND).

   6.2 Modo de test ATPG: entra en modo ATPG (generación automática de patrones de test) y comprueba que todas las salidas estén en su valor para este modo (gracias al multiplexor de salida).

7. **Test del sensor de temperatura**. Realiza las siguientes pruebas:

   7.1 Ejecutar el sensor de temperatura en modo disparo (se activa el registro de disparo, el sensor funciona una sola vez, y después el registro se pone a cero), pero con pwr_dwn_TS activado.

   7.2 Ejecutar ahora sí en modo disparo y comparar el registro TEMP con un valor esperado.

   7.3 Ejecutar en modo continuo (se activa un registro que establece funcionamiento continuo) y comparar el registro TEMP con un valor esperado.

   7.4 Disparar el sensor con un valor máximo de temperatura (y probar a leerlo).

   7.5 Disparar el sensor con un valor mínimo de temperatura (y probar a leerlo).

8. **Test de patrones de operación**

   Prueba la operación de varias líneas completas seguidas usando distintas configuraciones (patrones en apartado 2.3). Configura los registros con la tarea set_line_config, que establece una configuración de línea predefinida de acuerdo con las especificaciones del dispositivo. Hay muchas variables aleatorias que permiten combinar los ajustes dentro de unos límites precalculados. Los casos que realiza son:

   8.1 Totalmente paralelo.

   8.2 Totalmente serie.

   8.3 Casos especiales para probar situaciones raras de las máquinas de estados del DUT.

   8.4 Casos de potencia: alta, media y baja potencia.

   8.5 Casos LVDS: inactivo, test de imagen o normal (aleatorio). En serie o parcialmente paralelo.

   Cabe remarcar que cada caso se itera muchas veces y que en cada iteración se realizan entre 3 y 8 líneas seguidas. En los casos de completo paralelo y serie, el tester calcula cuantos ciclos tardará según la configuración exacta que haya tocado, y tras cada disparo espera esa cantidad para hacer el siguiente. Los disparos y las esperas, es decir, la operación de cada línea, se hace en la tarea line_gen.

9. **Test de paridad LVDS**: usa lvds_read con distintas configuraciones y va leyendo REGTM, ya que al final de este registro están guardados los bits de paridad de cada canal LVDS. Compara estos bits con los que generan los modelos LVDS conectados a las salidas. Las configuraciones son:

9.1 4 lecturas consecutivas con 2 salidas LVDS (paridad par).

9.2 4 lecturas consecutivas con 1 salida LVDS (paridad impar).

9.3 4 lecturas consecutivas con 4 salidas LVDS (paridad impar).

9.4 4 lecturas consecutivas con 4 salidas LVDS (paridad impar, modo inactivo).

4.2.6. Top

Este es el banco donde se pone a prueba el sistema completo. Además del bloque digital, incluye un modelo del bloque analógico. El objetivo es verificar como funciona la parte digital cuando está en contacto con la analógica, por lo que para está última se usa un modelo programado en Verilog-DMS (*Digital and Mixed-Signal*) que lidia con las señales analógicas tomándolas como variables reales. Es algo más simple que el sistema real, pero mucho más complejo y completo que el ADC_model que se usa en el anterior banco de pruebas.

En las entradas se aplica siempre uno de los mismos ocho patrones de tensión que se usaban para el test del filtro. Deberían generar los mismos códigos digitales en las salidas de los filtros si todo funciona correctamente. Las tensiones de alimentación se aplican con una rampa creciente en el encendido y una descendiente en el apagado. Las tareas relacionadas con el SPI y el LVDS son similares al banco del bloque digital.

Este banco recibe como argumento de entrada el nombre de un test, y se ejecuta ese una sola vez. En la práctica esto es lo mismo que antes, puesto que se pueden ejecutar todos los test en serie, pero asegura más la independencia de cada uno. Ambas soluciones permiten disponer si un test se realiza un número de veces (combinando así posibilidades con las variables aleatorias), o si otro test no debe ejecutarse. La diferencia es que en el banco del bloque digital esto se configura desde dentro, modificando el tester, y en este banco top lo que se modifica es cómo se llama desde fuera a simularlo.

Los test del top son estos:

1. **Totalmente paralelo**
2. **Totalmente serie**
3. **Parcialmente serie (transmisión LVDS durante la integración)**
4. **Parcialmente serie (conversión ADC durante la transmisión LVDS)**
5. **Parcialmente serie (integración durante la conversión ADC)**
6. **Apagado (test alimentaciones)**: se prueban todas las combinaciones posibles de registros que controlan señales de apagado.
7. **Apagado API**: el mismo test pero en modo API.
8. **Configuración**: ahora se prueban las señales de referencia poniendo valores aleatorios en sus registros.
9. **Conectividad**: se prueban otras señales que no se han probado en los anteriores, como las de modos de test.

10. **Selección de canal API**: se opera el chip con el patrón paralelo pero solo activando una selección aleatoria de los canales.

11. **Ajuste de señales**: se configuran en la P2RAM los registros OTP de ciertas señales de ajuste. Luego se prueba el modo de test de esta memoria en el que transmite su contenido por un pin analógico (el valor del pin va cambiando cada ciclo del reloj SPI)

12. **P2RAM**: se "queman" los fusibles de unos bits de repuesto del REG3 y se entra al modo bloqueado de la P2RAM al mandar una instrucción incorrecta. Después se prueba que al reiniciar el chip, si se manda la instrucción de cargar los valores quemados, se lee en estos bits de repuesto lo que se había escrito antes.

13. **Sensor de temperatura**: se prueban temperaturas y ajustes aleatorios, tanto en modo disparo como continuo. Se provocan situaciones críticas que deberían ser ignoradas, como solicitar un disparo cuando ya se está ejecutando otro. Por último se verifica que al ordenar una transmisión LVDS la palabra de temperatura contenga el valor correcto.

14. **Frecuencia de reloj**: configura todos los posibles valores de FDIV para ralentizar la frecuencia de los relojes internos.

15. **Dummy APS**: primero opera varias líneas configurando una imagen uniforme negra, después una blanca. Luego activa los modos de test, debería obtener negro o blanco en los canales pares o impares dependiendo del modo. Por último desactiva estos modos e intenta operar de nuevo de forma normal.

16. **Reloj ECHO**: se activa y desactiva el reloj echo del LVDS en diversos momentos mediante el registro de configuración.

17. **Referencias externas**: se opera el chip en un modo de test en el que se usan referencias externas para la tensión de banda prohibida y la referencia de tensión del ADC, en lugar de usar las que genera el bloque de referencia.

18. **Pistas API**: activando el registro TnH_api_bypass, se opera en un modo que deriva el API con el ADC, es decir, conecta las entradas directamente con los convertidores sin pasar por la fase de seguimiento y retención (TnH).

19. **LVDS**: se operan aleatoriamente todas las configuraciones LVDS, incluyendo los modos inactivo y test de imagen, el número de salidas, y las palabras de *start* e *idle*.

20. **Árbol NAND**: se activa el modo de test que conecta con puertas NAND las entradas que se han mencionado antes a sdo_pad. Se van poniendo valores aleatorios en las entradas y comprobando que la salida tenga el valor correcto cada vez.

Los test que no se han descrito simplemente funcionan configurando los registros del DUT según lo que se quiere probar, para después preparar un patrón de operación con la tarea set_line_config, y disparar las tres señales que activan las tres fases (trig para iniciar la conversión, int para la integración, data_start para la transmisión LVDS) según ese patrón con line_gen durante un número de repeticiones. Cada repetición, las señales de entrada cambian, los patrones van rotando en cada canal según el número de canal y el número de repetición.

La mayor parte de la verificación la realizan las aserciones. Muchos de los elementos del plan de verificación se van revisando en cada pulso del reloj de aserciones permanentemente. En los test solo se cambian señales y registros si hay una aserción asociada que verifique su funcionalidad.

Hay un grupo de aserciones que comprueban cada repetición que las salidas de los filtros sean las esperadas, según cada número de canal y repetición. Y un segundo grupo comprueba que lo leído por LVDS del DUT sea eso mismo también. Estas aserciones deben tener en cuenta si el canal concreto está activado (seleccionado) en primer lugar.

Es muy importante tener en cuenta que estos dos grupos no tienen por qué cumplirse desde la primera operación. Si por ejemplo se está trabajando totalmente en paralelo (fig. 4.1, al final del primer ciclo los datos de entrada solo han pasado por el integrador. El segundo ciclo esa salida del integrador se convierte en el ADC, y es en el tercer ciclo cuando ya se transmite por LVDS.

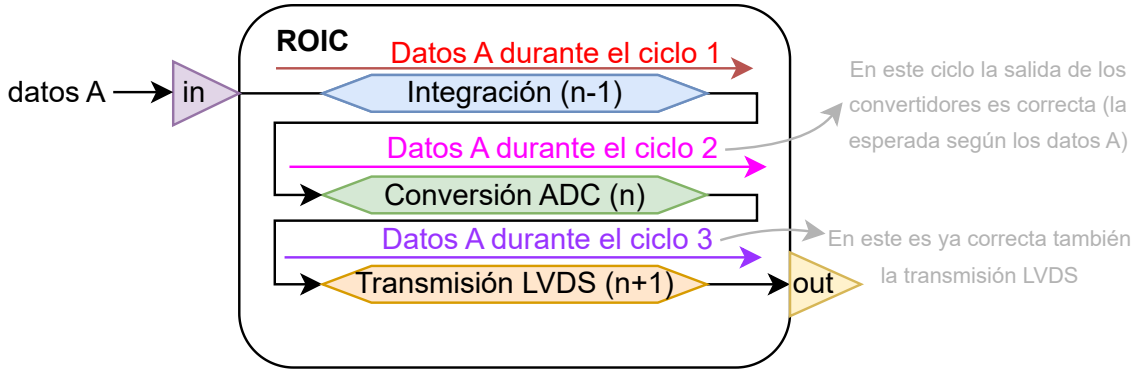


Figura 4.1: Diagrama del flujo de unos datos en el DUT al trabajar totalmente en paralelo.

Por lo tanto, en cada test los dos grupos de aserciones que verifican los datos se habilitan en distintos ciclos de operación, según el patrón que usa ese test. Además, según el patrón también cambia con qué hay que comparar las salidas. En modo totalmente en serie, la salida de los filtros y de la transmisión LVDS es la que toca para la entrada que se ha aplicado ese mismo ciclo, pero en otros casos no.

4.3. Implementación

A la hora de desarrollar los test, cabe resaltar tres propiedades que se han buscado:

- **Flexibilidad**: si cada aspecto se va a probar de múltiples maneras, lo más práctico es escribir código adaptable, desarrollar tareas que pueden ser reutilizadas, hacer sencilla la personalización de cada parte.
- **Robustez**: no deben ser ambiguos, ni tener éxito por muy poco, deben por ejemplo seguir funcionando cuando se incluyan los retardos *post-layout*.
- **Confiabilidad**: es importante que el éxito y la cobertura sean reales, es decir, cuando se estimula una variable, deben verificarse exhaustivamente sus efectos y utilidades. Debe ser seguro que si hacer algo en el DUT no ha dado errores, es porque realmente su funcionalidad ha sido correcta.

4.3.1. Escritura y lectura SPI

El primer aspecto que ha permitido flexibilidad ha sido tener en el tester un modelo de los registros. Por cada registro de 40 bits (o más en REGCS y ReadADC), una variable *logic* lo representa, y los elementos en cada registro que usan uno o más de sus bits tienen así mismo variables *logic* a su nombre. El registro completo de 40 bits funciona como un *wire*, ya que se establece su valor con el de todos los elementos concretos, mediante una declaración *assign*. En cambio, los elementos concretos funcionan como un *reg*, se les da un valor inicial y es en los bloques procesales (*initial* o *always*) donde van cambiando. Son estos últimos los que se usan cuando se quiere modificar el registro interno del DUT en un test.

Si quiero por ejemplo activar el registro que habilita el estado inactivo del LVDS, pongo a uno la variable de este modelo de LVDS idle, que está asignando un bit de REG6, y después ejecuto la tarea de spi_write en REG6. No habrá habido que definir el valor del resto de los bits de REG6, porque si no se han alterado aún están en su valor por defecto, que es el mismo que deben tener en el DUT en ese momento, así que no se verán afectados por la comunicación SPI que ocurra.

Sin embargo, puede ocurrir que el tester trate de escribir un registro del DUT, pero esta orden sea ignorada, en este caso las máscaras esclarecen qué bits van a poder cambiar y cuáles no. No se usan para evitar escribirlos, sino para que al verificar la escritura se sepa cuáles se espera que se hayan escrito. La máscara normal tiene un 1 en todos los bits que pueden ser escritos normalmente de cada registro, y un 0 en el resto. La otra máscara es especial para el modo fundición, en esta los bits OTP tienen también un 1, mientras que los bits que no se usan o los que son solo de lectura siguen teniendo un 0.

Cabe aclarar que las máscaras permiten comprobar que bits toca que estén escritos en el DUT, pero no impiden que el tester trate de escribir bits que van a ser ignorados. Es decir, delimitan la verificación de la lectura, no los estímulos de escritura.

El segundo aspecto flexible de este proyecto son las tareas, que no solo evitan tener que repetir el mismo código una y otra vez, además simplifican la comprensión y lectura de este. Como lenguaje de programación, Verilog tiene funciones y tareas. Mientras que una función está pensada para realizar algún procesamiento de la entrada y devolver un valor, una tarea es más general y puede calcular múltiples variables de resultado y devolverlas utilizando argumentos de tipo output e inout. Las tareas pueden contener elementos de simulación que esperen un tiempo, como esperar al flanco de una señal.

Las tareas spi_write y spi_read son muy reutilizadas por los test, una cuando se quiere escribir un registro de configuración, la otra cuando se quiere leer. Para enviar o recibir cada bit las dos tareas usan a su vez spi_rw_bit, una tarea que conmuta el reloj SPI, pone en el pin SDI el bit a enviar, ya que es la entrada de datos del DUT (MOSI), y lee el pin SDO (MISO) para saber el valor del bit recibido.

Tanto spi_write como spi_read comienzan aleatorizando el periodo del reloj SPI dentro de los márgenes permitidos. Después bajan la señal cs_n para iniciar la comunicación, y envían el byte de instrucción o comando, que es un 1 para lectura y un 0 para escritura, seguido de los 6 bits para la dirección del registro, y por último el bit de paridad de la instrucción, calculado de los 7 bits anteriores. Seguidamente se envían o reciben todos los bits de ese registro, para después poner la señal del reloj a 0, elevar otra vez cs_n, y esperar un tiempo mínimo, antes del cual no se pueden mandar más comandos.

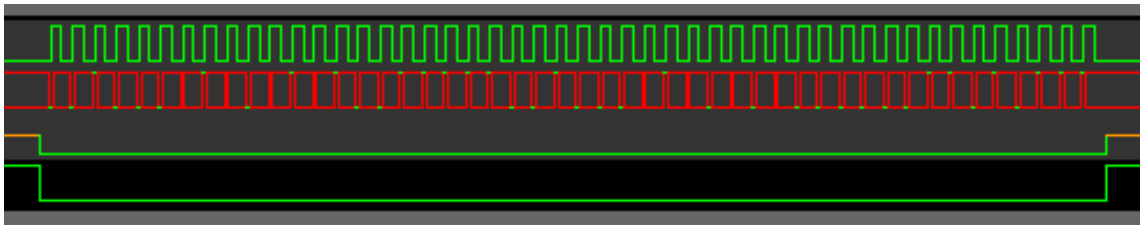


Figura 4.2: Formas de onda de una comunicación SPI con el esclavo (el DUT). Las ondas son, de arriba a abajo, SCLK, MOSI, MISO y cs_n.

La figura 4.2 es un ejemplo de spi_write funcionando. Al querer escribir, se baja cs_n (CS invertido) y se manda la instrucción de escritura (un 0 como primer bit) en la entrada de datos del esclavo, seguida de la dirección, la paridad, y los datos. Como consecuencia de ser una escritura, el DUT no envía nada de respuesta, y su salida de datos permanece a 0. Se puede observar que en el flanco de subida del reloj siempre se pone un 0 o un 1, pero en otros momentos la señal MOSI tiene un valor desconocido (X). Esto se hace para cerciorarse de que el DUT solo está leyendo el valor del pin en el momento permitido, pues de lo contrario leería la X y esta se propagaría a los registros, causando que fallaran los test.

4.3.2. Lectura LVDS

Se ha creado una tarea configurable llamada lvds_read, encargada de aleatorizar el periodo que va a usar para el reloj LVDS, generar el pulso de data_start que inicia la transmisión LVDS, ir guardando los bits recibidos en las cuatro salidas como palabras de 16 bits en una matriz por salida, e ir calculando el CRC que va saliendo de las palabras recibidas. Por otro lado, la tarea checkCRC es la encargada de comprobar que esas palabras recibidas sean correctas, comparándolas con las de referencia y asegurándose de que el CRC que ha ido calculando es el mismo que el leído como palabra final de la trama.

Para esta tarea y lvds_read es clave basarse en las tramas de las figuras 2.7, 2.8 y 2.9, puesto que son las que indican las especificaciones técnicas. Los datos ADC que se usan de referencia en este caso son la matriz que se envía como entrada, pero asignada a otra matriz bidimensional. Una dimensión es la de cada una de las 210 palabras, y la otra la de los 16 bits por palabra. De esta forma es más fácil comprobar la recepción en checkCRC. Esta transformación se ha hecho de la siguiente forma, aprovechando el bloque Verilog *generate* para condensar 210 líneas de asignaciones en una sola:

```
1 generate for (genvar g_i = 0; g_i < 210; g_i++) begin
2   assign adc_data[(16*(g_i+1)-1):16*g_i] = adc_data_array[g_i] ;
3 end
4 endgenerate
```

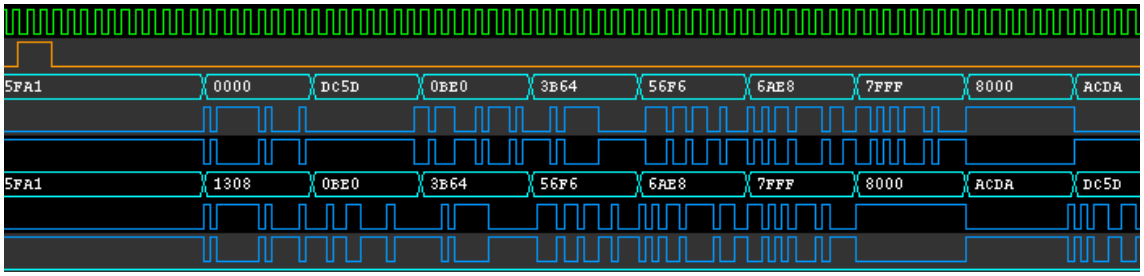


Figura 4.3: Formas de onda de una transmisión LVDS doble (dos salidas LVDS). La onda verde cuadrada es el reloj de la transmisión, DCLK. La señal naranja es data_start. Las azul claro son las próximas palabras que se van a transmitir (cuando se acabe con la actual) en cada línea, los pares polares azules son las dos líneas diferenciales de salida.

La figura 4.3 es un ejemplo de transmisión LVDS doble. Como se puede comprobar, unos ciclos después de que el test mande un pulso por data_start, las salidas diferenciales comienzan a enviar palabras. En este caso la palabra *start* está configurada como $5FA1_{16}$, y eso es lo primero que transmiten ambas líneas. La primera línea transmite después un 0, porque no se ha activado el sensor de temperatura, así que el registro está vacío.

4.3.3. Configuración y operación del ROIC

Cuando se aleatorizan variables, se hace siempre dentro del rango permitido por las especificaciones del chip y por las condiciones del test. En set_line_config se definen unos ajustes según el patrón, y siempre que esto no impida probar ese patrón, se aleatorizan las configuraciones dentro de los límites. En muchas aleatorizaciones se usan estructuras como esta:

```
1 randcase
2     1: adc_start  = 0;
3     1: adc_start  = 1023;
4     8: adc_start  = $urandom_range(1023);
5 endcase
```

De esta forma se cubre todo el rango, pero se hace más probable que en alguna iteración ocurran los casos extremos, que siempre es importante verificar. Para el patrón totalmente en paralelo se aleatoriza esta palabra, que le indica al DUT cuanto debe esperar desde que recibe el disparo hasta iniciar la conversión. También se aleatoriza el OSR y ADC_offset_comp_long. Una vez se ha establecido el valor de estas variables, se calcula el tiempo de conversión, lo mismo que el tiempo de transmisión, sabiendo el modo configurado (simple, doble, cuádruple) y el periodo actual del reloj del sistema y el del LVDS. De estos dos tiempos, se establece como tiempo de línea el más largo de los dos, y se configura el tiempo de integración para que dure tan solo un poco menos.

En cambio, en el patrón totalmente en serie el tiempo de integración también puede ser aleatorio, y el tiempo de línea será la suma de los tres tiempos. Este tiempo de línea es el que usa line_gen para esperar antes de la siguiente iteración, en la que se vuelven a disparar las tres fases. Como se ha mencionado antes, line_gen es la tarea encargada de lanzar los tres disparos, según los tiempos configurados en set_line_config.

Esto se consigue gracias a la estructura *fork-join* de Verilog, que permite ejecutar varios bloques procesales de forma concurrente. Así, un bloque dispara trig (inicia conversión) tras la espera

especificada y con un pulso tan largo como se haya predispuesto, otro bloque activa int (habilita integración mientras se mantenga alta la señal), y otro dispara data_start (que inicia la transmisión LVDS). Un cuarto bloque sería simplemente la espera del tiempo de línea. Hasta que no acaban los cuatro bloques, que han comenzado a la vez, no termina el *fork-join*.

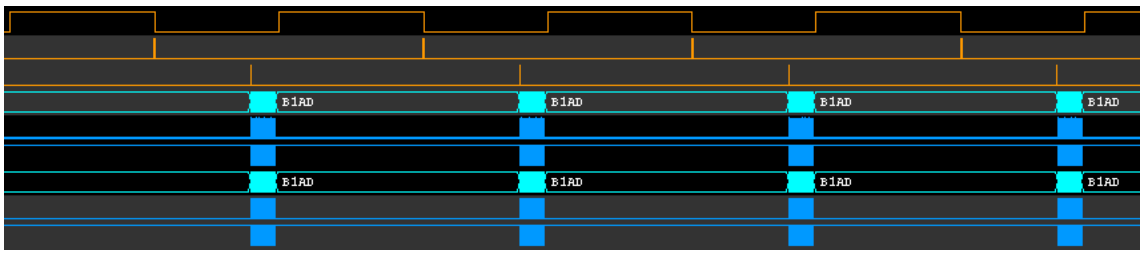


Figura 4.4: Formas de onda de una operación totalmente en serie del ROIC. La señales naranjas son, de arriba a abajo, int, trig y data_start, que se activan en ese orden. Las azules son otra vez las salidas diferenciales LVDS.

La figura 4.4 es un ejemplo de operación en el que se realizan las tres fases en serie, y hasta que no se ha terminado una, no se comienza la siguiente. Esto conlleva que los datos que se han integrado son los mismos que los que se transmiten ese ciclo de operación.

Durante la fase de conversión de las operaciones, un bloque *always* verifica de vez en cuando que nuevos disparos de trig son ignorados. Para esto, una señal en el test indica cuando se está en medio de una conversión, al activarse al inicio de una y desactivarse antes del fin de la conversión. Cada ciclo de reloj que está señal esté activada, se usa la condición "if($urandom%2000==0)" para que haya una posibilidad muy baja de entrar en este bloque.

Hay aserciones que escuchan a trig para contar ciclos, estas leen un registro que indica que los trig recibidos son válidos. Así que en este bloque se desactiva ese registro, pues el DUT va a ignorar el próximo trig, o debería, pues es "inválido" al haber sido enviado durante una conversión. Tras eso, se manda un pulso trig de longitud aleatoria y luego se vuelve a activar el registro.

4.3.4. Reinicio

Las tareas para reiniciar son distintas en el banco digital y en el top. En el digital se aplica directamente la señal de reset que sea. Hay una asíncrona más general, POR, y una síncrona que solo reinicia algunos elementos, RST. Se ponen en su valor por defecto los mismos registros que lo deban hacer en el DUT, y se aleatoriza el periodo del reloj del sistema, para ir probando distintas frecuencias.

En el del top se debe simular el comportamiento de las alimentaciones, por lo que se aplica en las señales concretas una rampa de tensión creciente o decreciente para encenderlas o apagarlas. En este caso solo existe la entrada de RST, ya que la de POR es generada internamente por el DUT en un bloque cuando se reinician las alimentaciones (cuando se enciende el dispositivo después de haberlo apagado). Cuando se está en estas tareas, se deshabilitan la mayoría de las aserciones, puesto que no están pensadas para cumplirse en este estado.

4.3.5. Aserciones concurrentes

En un archivo separado, tanto el banco digital como el top declaran muchas aserciones que equivalen a elementos concretos del plan de verificación. Están definidas como propiedades concurrentes, y en su gran mayoría se evalúan cada flanco de subida del reloj de aserciones. Como se ha mencionado antes, este reloj es una versión retrasada del reloj del sistema. La clave con los cambios de señales es que ocurran antes del siguiente flanco de reloj, por lo que este retraso hace que se evalúen las aserciones algo antes del próximo flanco, y no justo después de cada flanco de reloj, donde muchas pueden no haber tenido tiempo para cambiar, especialmente en las simulaciones *post-layout*.

Se han clasificado la mayoría de aserciones como funcionales o de alimentaciones, y hay dos variables que habilitan cada uno de los dos tipos. Además, algunas aserciones no se evalúan tampoco si se cumplen otras circunstancias.

Por ejemplo, hay una aserción que comprueba que la primera salida diferencial LVDS sea realmente diferencial en todo momento (siempre un polo alto y otro bajo). Al ser la primera salida, esto debe ocurrir independientemente de si se está en modo simple, doble o cuádruple. Sin embargo, no ocurrirá si el bloque LVDS se encuentra apagado, por lo que el registro que lo apaga deshabilita esta aserción, además de la variable de las aserciones funcionales.

En el siguiente código de ejemplo se define la misma aserción, pero para la segunda salida LVDS (p_lvds_two_xor). En este caso hay una tercera posibilidad que desactiva la aserción: que se esté trabajando en modo simple. La forma en que se comprueba que los dos polos sean distintos el uno del otro es mediante el operador XOR (disyunción exclusiva).

```
property p_lvds_two_xor ;
  @(posedge dclk_rec) disable iff(~functCheckers_enable |
  lvds_pwr_dwn | LVDS_singleOut) (dout_p[2] ^ dout_n[2]) ;
endproperty
property p_lvds_two_off ;
    @(posedge clk_assertion) disable iff(!functCheckers_enable)
    (lvds_pwr_dwn | LVDS_singleOut) |->
    (dout_p[2]===0 & dout_n[2]===0) ;
endproperty
property p_lvds_two_low ;
    @(posedge clk_assertion) disable iff(!functCheckers_enable)
        ($past(lvds_end_flag) & lvds_end_flag &  !lvds_pwr_dwn &
        !LVDS_singleOut) |-> (dout_p[2]===0 & dout_n[2]===1) ;
endproperty

assert property (p_lvds_two_xor) ;
assert property (p_lvds_two_off) ;
assert property (p_lvds_two_low) ;
```

Las otras dos aserciones del ejemplo comprueban otros dos posibles estados de la misma salida LVDS. p_lvds_two_off se asegura de que los polos están apagados (ambos a 0) justamente en las situaciones donde no deben ser diferenciales, es decir, las circunstancias que en la primera aserción la deshabilitaban, en esta segunda la accionan. La tercera aserción verifica que justo después de terminar una transmisión (momento en el que hay un pulso en lvds_end_flag), la salida se pone en bajo diferencial, independientemente del valor del último bit transmitido (un fallo típico sería que se quedara el último 1 congelado).

Las aserciones tratan de verificar que la funcionalidad que se especifica en un elemento del plan se cumple correctamente. Como se ha podido ver, algunos elementos se han verificado con más de una aserción, normalmente con el fin de simplificar la comprensión. Es decir, se quiere evitar que por integrar toda la comprobación en una sola aserción, no sea esta demasiado extensa y compleja.

Hay aserciones duplicadas que comprueban que una consecuencia lógica se cumple en el sentido opuesto. Esto no es cierto siempre (véase falacia de la afirmación del consecuente), pero cuando sí debe ser así, se ha verificado como en el ejemplo siguiente:

```
1  property pX ;
2      // si A es cierta, inmediatamente B debe serlo
3      @(posedge clk_assertion) A |-> B ;
4  endproperty
5  property pX_comp ;
6      // si B es cierta, inmediatamente A debe serlo también
7      @(posedge clk_assertion) B |-> A ;
8  endproperty
9
10 assert property (pX) ;
11 assert property (pX_comp) ;
```

Hay algunas aserciones que verifican la conectividad de ciertas señales comprobando que si cambian en un bloque, en otro bloque del chip también lo hacen. La conectividad de cierto bloque se asegura al comprobar todas sus señales y hacer una inspección visual de los esquemáticos.

4.3.6. Grupos de cobertura

Como especifica el plan, hay que asegurarse que ciertas propiedades son apropiadamente cubiertas por los test en los momentos adecuados. Es importante esto último, pues no sirve de mucho por ejemplo comprobar que una configuración LVDS ha sido utilizada, si no ha sido durante una transmisión LVDS.

Cuando el punto de cobertura es un bit, es sencillo, se verifica que el bit se ha muestreado al menos una vez como un 0 y una vez como un 1. Pero cuando son varios bits, a no ser que se quiera comprobar también todo el rango de valores, se ha determinado cierto rango concreto, basándose en las especificaciones y el funcionamiento de esa variable.

Se han definido dos *covergroup* y tres *coverpoint* particulares. El primer grupo es para la configuración del convertidor. Este toma sus muestras diez ciclos después de cada flanco de bajada del trig válido (el falso inyectado durante algunas conversiones es ignorado). Los *coverpoint* que contiene son estos:

- El registro ADC_offset_comp_long, que aumenta el tiempo de compensación.
- El registro TnH_SerialMode, que activa el modo en serie.
- El registro TnH_selCsample, que selecciona entre dos condensadores de muestreo.
- El registro TnH_api_bypass, que conecta la API con el ADC, omitiendo el TnH.

- La espera antes de comenzar la integración, al menos deben cubrirse las esperas muy breves (entre cero y diez ciclos).
- La duración de la integración, debe cubrir ocho secciones repartidas entre unos 60 y 400 ciclos.
- La espera programada en ADC_start, se deben cubrir tanto ocho secciones repartidas entre 1 y 1022, como los valores mínimo y máximo (0 y 1023).
- El OSR, se deben cubrir los cinco posibles valores.
- La duración del pulso en trig, deben ocurrir pulsos de disparo que duren un, dos, tres y tres o más ciclos.

A su vez, define también cobertura cruzada entre:

- ADC_offset_comp_long y el OSR
- ADC_offset_comp_long y TnH_SerialMode
- TnH_SerialMode y el OSR

Se ha definido cobertura cruzada entre los elementos más importantes y que puedan estar relacionados entre sí. Se quiere, por ejemplo, probar todos los OSR tanto en el modo en serie como con el modo desactivado, pero no es necesario probar todas las esperas ADC_start con todos los OSR, cobertura que requeriría además una cantidad enorme de simulaciones.

El segundo grupo es para la configuración de la transmisión LVDS, toma sus muestras diez ciclos del reloj LVDS después de un flanco de subida de data_start. Contiene los siguientes *coverpoint*:

- El registro LVDS_singleOut para usar una sola salida LVDS en la transmisión.
- El registro LVDS_quadOut para usar cuatro salidas LVDS (este tiene menor prioridad).
- El registro LVDS_image_test para establecer el modo test de imagen.
- El registro LVDS_idle para establecer el modo inactivo.
- La palabra inactiva de 16 bits LVDS_idle_word, se deben cubrir tanto ocho secciones repartidas entre 1 y 65534, como los valores mínimo y máximo (0 y 65535).
- La palabra de inicio de la transmisión lvds_startword, se debe cubrir igual que la anterior, excepto que su valor mínimo es 1 (es necesario un patrón para reconocer el inicio de la trama).

Como ocurre antes con el ADC, al ser medida la configuración LVDS durante una transmisión, es seguro que se está utilizando, ya que si no fueran realmente así, las tareas y aserciones que verifican esta comunicación fallarían. Se define cobertura cruzada entre:

- LVDS_singleOut y LVDS_idle.

- LVDS_quadOut y LVDS_idle.
- LVDS_singleOut y LVDS_image_test.
- LVDS_quadOut y LVDS_image_test.

Esto asegura que los modos inactivo y test de imagen se verifican para todos los números de salida LVDS posibles. Por último, los tres *coverpoint* adicionales son las frecuencias de tres relojes:

- RCLK: el reloj del sistema debe estar al menos una vez entre 4 y 4.4 MHz, al menos una vez entre 8 y 8.6 MHz, y al menos una vez entre 16 y 17.2 MHz. Se muestrea diez ciclos después de un reinicio.
- SCLK: el reloj SPI debe estar al menos una vez a máxima velocidad (entre 14.9 y 15.1 MHz) y deben cubrirse ocho secciones repartidas entre 2 y 14.99 MHz. Se muestrea diez ciclos después de un flanco de bajada de cs_n (inicio de comunicación SPI).
- DCLK: el reloj LVDS debe estar al menos una vez a máxima velocidad (250 MHz) y deben cubrirse ocho secciones repartidas entre 150 y 250 MHz. Se muestrea diez ciclos después de un inicio de transmisión LVDS.

La forma en que se muestrean estas frecuencias es igual en los tres casos. Se mide el tiempo en un flanco del reloj, se mide el tiempo en el flanco siguiente, y la frecuencia es simplemente el inverso multiplicativo de la diferencia entre estos dos tiempos.

$$f_{clk} = \frac{1}{t_1 - t_0} \tag{4.1}$$

4.3.7. Ajustes y mejoras

A medida que se iban implementando test, ya se estaban usando las herramientas de cobertura para ir buscando aspectos sin verificar todavía. vManager da varios tipos de medidas que permiten descubrir posibilidades que los test no están cubriendo. Estas medidas que resultan de las simulaciones se explican en la sección de resultados, pero ya han estado ayudando en la propia implementación.

Es de esta forma que se aplica la metodología MDV. El plan de verificación se materializa en forma de aserciones, casos de test, y coberturas, y los test se van ampliando para completar la ocurrencia de estos elementos. Una vez finalizado este proceso, cuando se han conseguido simulaciones del RTL satisfactorias, se ha pasado a realizar simulaciones denominadas *post-layout* (PL).

Son simulaciones posteriores al posicionado y enrutamiento (*place and route*), su diferencia principal es que incorporan análisis de tiempos en las *netlist* para simular los retardos temporales que tendrá necesariamente el chip real, además de otros efectos, como los problemas parasíticos (resistencia de los cables, inductancia mutua entre cables adyacentes, capacitancia de acoplamiento, etc. [14]). En la tabla 4.1 se comparan ambos tipos de simulación.

Aspecto	**Simulación RTL**	**Simulación PL**
Objetivo	Verificación funcional	Verificación temporal y física
Errores	Lógicos y funcionales	Tiempos y efectos parasíticos
Duración	Corta	Larga
Precisión	Media	Alta

Tabla 4.1: Tabla comparativa entre las simulaciones RTL y *Post-Layout*.

En estas simulaciones no se mide la cobertura, no tiene mucho sentido si son los mismos test que ya la han cubierto en las RTL. Sin embargo, se deben implementar los test pensando también en ellas. En general, entrar dentro del DUT en simulaciones PL puede ser problemático y, por ende, es mejor siempre que sea posible estimular solo los pines de entrada y leer los de salida. Se ha intentado evitar escribir variables internas, sin embargo, en alguna parte del banco digital no ha quedado más remedio. En ese caso se ha hecho forzando cada bit con la sentencia *force*.

Además, muchas aserciones leen variables internas para verificar ciertas funcionalidades. En un mundo ideal, donde la información se propagase instantáneamente, si A debiera provocar B automáticamente, en el momento que cambiara A (en el flanco de reloj por ejemplo), B también lo haría. Pero esto no es así, en el mundo real B se retrasa un tiempo, cuyo factor principal son los tiempos de las puertas entre ellos.

Es debido a esto que se usa el reloj de aserciones. Como se ha explicado en el apartado de las aserciones, este se retrasa cierta cantidad respecto al del sistema. Las simulaciones PL influyen en esta cantidad, muchas veces, si estas fallan, cambiar este retraso puede permitir que todas se cumplan. La restricción clave es que la aserción se debe evaluar después del flanco del reloj y antes del flanco siguiente (puesto que eso ya correspondería al siguiente ciclo de reloj).

Por otro lado, hay alguna otra espera en según que tareas que puede ser importante en estas simulaciones. Por ejemplo, en la tarea spi_rw_bit para enviar y/o recibir un bit, todas las asignaciones a los pines SPI usan unas esperas definidas por las especificaciones. Si los retardos provocados por el PL no permiten tener a tiempo el dato en el MISO (SDO) en una situación concreta, la tarea spi_read dará bits incorrectos, lo que sería un fallo del DUT detectado con la verificación.

4.4. Automatización

La automatización de las simulaciones se consigue con el procesamiento por lotes (*batch*). Se han escrito archivos con órdenes que son ejecutadas secuencialmente por el sistema operativo. Se puede llamar a este archivo manualmente, o darle la ruta al vManager para que lo use como *run_script*. Este archivo es de tipo *makefile*, compila el código RTL y el del banco de pruebas y genera la simulación.

Estos *makefile* han sido programados con argumentos de entrada opcionales para configurar la simulación. Estos argumentos son:

- El nombre del test que se quiere que el banco ejecute.

- El tipo de simulación (RTL o PL).
- En caso de PL el modo definido en SDF (formato estándar de retrasos), que puede ser el peor caso de temporizaciones, un caso típico, o el mejor caso.
- Una semilla concreta para generar los estímulos aleatorios, si no se desea una cualquiera (porque se quiere reproducir una simulación que falló anteriormente, por ejemplo).
- La cobertura, si se desea, que puede ser funcional o total.

En vManager, cada run_script es usado por un grupo que contiene uno o varios test y está configurado también con un *timeout*, el tiempo máximo que puede durar el proceso antes de que el programa lo mate forzadamente, y una cuenta, el número de veces que se ejecutan los test por defecto en ese grupo.

Los grupos definidos en un archivo conforman una regresión (también llamada sesión). Esta tiene un número, que es simplemente la suma total de ejecuciones de cada uno de los test de cada uno de los grupos de la regresión. Al lanzarse la regresión, cada una de estas ejecuciones puede fallar o pasar satisfactoriamente.

El run_script de cada grupo que llama al *make* para lanzar la simulación tiene un argumento para seleccionar la cobertura (solo funcional o toda). La solo funcional habilita el tanteo de todas las aserciones y *covergroup* compilados. Cuando se selecciona toda la cobertura se habilita no solo la funcional, sino también el tanteo de la cobertura de cualquier módulo, instancia o bloque que se especifique.

Para cada grupo de test se ha programado el tanteo de la cobertura de solo el DUT que usa ese banco de prueba. No tendría sentido habilitar esta cobertura en el módulo testador, pues no se necesita que todas sus líneas de código del test sean recorridas, por ejemplo. Lo que se desea medir es la exhaustividad con la que el test ejercita el diseño del chip, no el test que lo prueba. La funcional sí que busca y mide sus aserciones y grupos de cobertura en todas partes.

Además, cada grupo tiene también un scan_script, que es una ruta al archivo donde se han definido manualmente dos filtros que usa vManager para escanear la salida de ejecutar la simulación. El primero de los dos busca palabras de error impresas durante la simulación. Comandos de error como "$error" ya son detectados automáticamente, pero hay tareas que simplemente imprimen un error en pantalla para no detener la simulación e incrementan una cuenta de errores, este filtro se asegura de que no se haya impreso ninguno de estos errores. El segundo filtro busca la frase "Simulation passed successfully", que se imprime al final de todas las simulaciones si estas terminan con cero errores. La utilidad de este filtro es asegurarse de que la simulación realmente ha finalizado satisfactoriamente, y no se ha quedado colgada.

Cuando se mide la cobertura total del DUT, pueden definirse excepciones para elementos que no es necesario cubrir por diversas razones. Esto se puede conseguir escribiendo en el código directivas llamadas *pragma*, instrucciones comentadas que son por tanto ignoradas para generar el diseño RTL, pero que utiliza el compilador que las entienda, en este caso el que mide la cobertura de la simulación.

Otra opción, la que se ha usado en este proyecto, es usar la herramienta de *refinements* de vManager. Esta permite indicar las excepciones desde el programa, y las guarda en un archivo con extensión ".vRefine". Esto tiene numerosas ventajas, permite ver la cobertura antes de aplicar el archivo y después, tener más de un archivo de excepciones para ver la cobertura con cada uno, etc.

Algunos ejemplos de excepciones que hay en este proyecto son estos:

- Secciones de código importado de otros proyectos donde hay un *pragma* que informa al compilador del diseño que las líneas siguientes no deben ser sintetizadas. Es código que no se usa en este caso, así que no tiene sentido medir si ocurre.
- Variables concretas de muchos bits en las que no se necesita recorrer absolutamente todos los bits, por ejemplo porque ya se usaban en otros proyectos o porque no son esenciales para el funcionamiento correcto del chip.
- Algunos bits de ciertas variables que nunca cambian porque sus posibles valores no lo permiten. Por ejemplo, si una de 3 bits solo puede ser 3 (011_2) o 6 (110_2), el segundo bit nunca puede ser 0.
- Algunas condiciones alternativas implícitas (cuando no se define un bloque *else* tras uno *if* o el caso *default* de una estructura *case*, el compilador los considera implícitamente como bloques vacíos). La mayoría de estas deben ocurrir para que haya cobertura total, pero hay casos donde el diseño se quiere asegurar de que algo se cumpla para efectuar una acción, pero siempre que el chip funcione sin fallos, debería ocurrir, por lo que es imposible crear un test externo que lo haga ocurrir solo controlando las entradas y salidas del DUT.
- Otras condiciones alternativas definidas como casos de error. Si estos errores no se pueden provocar desde fuera, no se pueden cubrir externamente a menos que se fuercen valores dentro que no ocurrirían normalmente.
- Variables utilizadas por la parte analógica real, pero que el modelo analógico que se usa no emplea.

El test de registros SPI (banco digital), al probar a escribir en todas las direcciones de los registros y leerlas, sin tener en cuenta en ningún caso la funcionalidad de ese registro, puede ocasionar una cobertura total engañosa. Al ver en vManager que un registro ha tenido al menos tanto un flanco de subida como uno de bajada durante la simulación, se puede pensar que el funcionamiento de ese registro se ha comprobado, pero tal vez simplemente ese test haya cubierto esa medición.

Para evitar esto, en el grupo general que prueba toda la parte digital y mide la cobertura total se ha desactivado este test, así los registros cambian de valor solo cuando se usan en su contexto. El test sí se ha incluido en otro grupo que solo activa la cobertura funcional (aserciones), y en otra sesión con todos los test activados, donde ya no preocupa este problema, pues ya se comprueba en la primera sesión.

Capítulo 5

Resultados

A lo largo de la verificación se han encontrado varios fallos y aspectos a mejorar del diseño. Estos son algunos ejemplos de *bug* encontrados (que fueron consecuentemente resueltos):

- En el modo inactivo la palabra inactiva sustituía todas las palabras, no solo las de datos.
- El valor por defecto de algún registro.
- Al leer el registro ReadADC por SPI, la secuencia de direcciones que usaba para ir leyendo cada canal era incorrecta.
- el controlador LVDS tenía un *bug* cuando se habilitaban al mismo tiempo el registro del modo simple y el del modo cuádruple.
- La cobertura descubrió partes que ya no eran necesarias del diseño, como estados obsoletos.

Una vez resueltos todos los errores, y cubierto el plan de verificación, se ha evaluado la verificación en sí. Esto se ha hecho usando las métricas del vManager.

Dejando de lado las aserciones, en las que las métricas simplemente comprueban que se hayan ejecutado en algún momento de la simulación y que el resultado haya sido correcto, las métricas de cobertura del vManager se pueden separar en tres categorías. La primera es la cobertura de bloques, que comprueba que todos los bloques de código se hayan ejecutado. Esto incluye los implícitos, es decir, si se define una condición *if*, pero no se define su alternativa *else*, se entiende que hay un bloque *else* vacío después del *if*, y este también se debe ejecutar al menos una vez.

La segunda categoría son las expresiones, donde se mide la exhaustividad con la que el test está ejercitándolas. Esto se mide identificando todas los argumentos que controlan la expresión y comprobando que influencian el resultado al menos una vez. Es decir, se asegura de que las variables que la controlan realmente están sirviendo de algo. Si por ejemplo quitar una de las variables diera el mismo resultado en el conjunto de casos que efectúa la simulación, o bien esa variable es innecesaria en la expresión, o bien el test no está contemplando todos los casos posibles.

En tercer lugar está la conmutación (*toggle*), mide que todos los bits de todas las variables de todos los módulos tengan al menos un flanco de subida (cambien de 0 a 1) y uno de bajada (cambien de 1 a 0). Hay una cuarta categoría, cobertura de declaraciones, pero está desactivada por defecto.

Si en la categoría de bloques se cuenta que se ha ejecutado un bloque, en esta categoría se cuenta que se han ejecutado todas sus líneas internas, por lo que no ha sido necesario usarla.

La sesión con simulaciones RTL y cobertura se han cargado varias veces, así se ha asegurado que se tiene varias sesiones con todas las ejecuciones exitosas, habiendo en cada sesión varias repeticiones de cada test de cada banco de test (más o menos repeticiones en función de cuantos componentes aleatorizables tienen, y de la importancia del test en la verificación).

Por lo que respecta a la sesión PL, con tener un par completas correctas se ha dado por buena. Esto es debido a que la mayoría de la verificación se ha hecho ya con las RTL, el tipo de fallos que aparece en las PL pero no sucedía en las RTL ocurre con mucha facilidad, ya se ha programado la sesión para que una sola cubra toda la cobertura del diseño.

Además, las PL son mucho más lentas, si por ejemplo el test top de la operación totalmente en paralelo dura con la RTL unos 20 segundos, con la PL dura unos 200. Esto provoca que la sesión completa dure varias horas, aún teniéndola configurada para poder realizar hasta cinco ejecuciones a la vez.

La cobertura total del DUT con el banco top ha sido mayor del 99 %. Según el ámbito, el porcentaje mínimo que se recomienda cubrir del código puede variar bastante. En cualquier caso, en nuestro caso se deseaba alcanzar al menos un 95 %, por lo que este resultado es más que satisfactorio. Los elementos que faltan por cubrir pertenecen a las posibles excepciones que se han mencionado antes. Si no se han puesto como tales en vManager es porque ya se ha comprobado visualmente que no son relevantes.

Como excepciones se han definido los elementos que claramente no se iban a cubrir, estos que faltan por cubrir son casos que o sí se van a cubrir en otras sesiones (o en otros proyectos si es un bloque importado), o no se van a cubrir pero se quiere dejar a la vista este hecho.

Capítulo 6

Conclusiones

El objetivo de este proyecto era verificar que las especificaciones técnicas planeadas para un nuevo ROIC han sido conseguidas en su diseño (y *layout*). Este TFM describe este proceso de verificación, en el que se ha usado la metodología basada en métricas.

El proceso ha comenzado estudiando bien la funcionalidad que se requiere del circuito integrado, para lo que hay que comprender bien aspectos como la interfaz periférica en serie (SPI), la transmisión con señales diferenciales de bajo voltaje (LVDS), el convertidor analógico-digital (ADC) Sigma-Delta y su filtro, o el circuito de seguimiento y retención (TnH). En menor medida, también es bueno entender la aplicación del dispositivo, ya que puede esclarecer el por qué de algunos requerimientos.

La implementación en sí ha empezado con el desarrollo del plan de verificación, un conjunto de elementos a comprobar basados en la hoja de especificaciones. Luego se han creado unos grupos de test que al simularse junto con el DUT, excitan sus entradas para poder ponerlo en la situación que requiere cada comprobación, y a la vez efectúan esas comprobaciones de forma automática, o bien comparando lecturas con unas esperadas (valores ideales), o bien con aserciones.

Se ha tratado de aleatorizar las variables y factores del test todo lo que fuera posible. Sin embargo, se le han dado prioridad a los peores casos, programando los test para que ocurrieran más a menudo que el resto, para que fuera más probable que el test simulara situaciones extremas.

El plan de verificación ha sido inicialmente una guía mínima de la que apoyarse para crear test y aserciones. Sin embargo, a lo largo del proyecto se han ido descubriendo nuevas posibles situaciones y variables que comprobar. Para esto, las medidas de cobertura han sido una forma muy eficaz de averiguar deficiencias en los test. Una vez se han automatizado las regresiones con test en vManager, se han comenzado a examinar los elementos que aún estaban sin cubrir. O bien era necesaria su cobertura, y por tanto era necesario mejorar algún banco de test, o bien no era necesario cubrirlos, y se podían definir como excepciones.

Por otro lado, en vManager se ha creado también el plan de verificación, y se ha enlazado cada componente con uno o varios test o aserciones que lo comprobaran. Esto facilita la tarea de ir viendo que partes del plan están ya cubiertas, es una forma más de asegurarse de que se está probando todo lo necesario. En resumen, el plan asegura que se prueban todas las funcionalidades del chip, y una alta cobertura asegura que se prueban todas sus situaciones posibles en general.

El resultado de todo es que se ha podido confirmar satisfactoriamente el desempeño del disposi-

tivo diseñado, lo que permite lanzar su fabricación con un grado suficientemente alto de confianza. Como grado de seguridad adicional a cumplir con todas las especificaciones, se ha medido que las simulaciones llevadas a cabo con los test creados obtienen una cobertura superior al 99 % en todos los módulos del DUT, y se ha comprobado que la cobertura restante es irrelevante.

Analizando la realización de este proyecto, la mayor parte de fallos en el diseño del DUT se encontraron durante los primeros meses, y cada vez estos defectos eran más complejos y se daban en situaciones más raras. Los últimos han sido de índole temporal, al detectar que una especificación de tiempo no se cumplía. Según el caso hay que tomar una decisión, o bien se mejora el diseño o *layout* (el que sea el causante de este desfase), o bien se rebajan las restricciones temporales del chip.

Cuando se desarrollaban nuevos test, la práctica más prudente si surgían fallos ha sido primeramente revisar el test, y asegurarse de que hace apropiadamente lo que se deseaba. Si es así, se busca la causa de este fallo, en este proyecto ha ocurrido a menudo que fuera un uso indebido del DUT por parte del test, o una falsa detección de fallo, por tener una referencia ideal incorrecta, por ejemplo. Sin embargo, en ocasiones sí que era un defecto en el diseño o en el *layout*. Es por esto que hasta que no se han resuelto absolutamente todos los errores, no se puede asegurar que el chip funciona en las situaciones verificadas.

Las herramientas de cobertura son especialmente útiles cuando ya se han desarrollado los test que se planificaron inicialmente con el plan de verificación, pues dan una indicación de cómo de bien están verificando. Dan pistas muy ilustrativas de qué partes se podrían ampliar. Si llevar los errores al mínimo confirma el funcionamiento en las situaciones simuladas, llevar la cobertura al máximo confirma que las situaciones simuladas son todas las posibles, o al menos todas las relevantes al definir excepciones.

Una de las limitaciones de este trabajo ha sido que, al usarse un entorno de simulación digital-mixto (DMS), los bloques analógicos se han representado con modelos simplificados. Estos bloques han sido verificados por separado en un dominio analógico, pero para la verificación que concierne a este trabajo, donde se ha usado el *top* del chip, combinando bloques digitales y analógicos, estos últimos han tenido que ser modelos que imitan el comportamiento de los reales, pero que pueden funcionar aquí.

En esta situación, cabe resaltar la importancia de comprobar el diseño analógico de otra forma, y tratar de minimizar las diferencias entre el circuito real y su modelo. Con uno lo suficientemente parecido, ya se puede cumplir el objetivo de este trabajo, que es verificar el funcionamiento del diseño digital, y también del conjunto total, en el que ambas partes interactúan entre sí.

Otra posible limitación ha sido no usar UVM. Este puede resultar ventajoso si por ejemplo el proyecto reusa muchos bloques IP, como procesadores, buses, memorias, etc. Sin embargo, este no era el caso de este proyecto. En general, UVM permite crear una plataforma de verificación sobre la que se puede construir en cada proyecto que sustituye a uno anterior. Por lo tanto, si este proyecto ya viniera de una serie que se estuviera verificado con UVM, usar UVM también en este sí que hubiera sido lo más conveniente.

Realizar una verificación formal con soluciones como OneSpin hubiera sido otra posibilidad. Este método tiene las ventajas de ser rápido y ágil, a la vez que asegura mejor la completitud de la verificación. El problema que tiene es que se hace exponencialmente más complejo al aumentar el diseño, el de este proyecto es demasiado grande. Además, no es adecuado para funciones de datos complejas como las de los filtros de este DUT. Por último, tampoco sirve para validación.

Dicho, esto, si se usó la herramienta para comprobar módulos concretos de forma aislada, y resultó conveniente por ejemplo para saber si era posible alcanzar todos los posibles estados de una máquina de estados.

En el campo de la programación, diseñar los test de forma modular, de forma que se pudieran llamar cada uno separadamente, ha demostrado ser mucho más práctico. Un beneficio de hacerlo así es que cuando se quiere comprobar algo de un test no es necesario realizar la simulación completa, solo la de ese test. Otro es que se pueden programar las regresiones para que ejecuten más veces algunos test.

También ha sido efectivo organizar bien los archivos de test. Es decir, declarar las aserciones en un archivo, los parámetros en otro, en otro las tareas, etc. Esto requiere un esfuerzo inicial algo mayor, pero se evita acabar con un archivo demasiado largo, en el que realizar cambios es cada vez más difícil, y se consigue en su lugar una solución flexible y fácilmente ampliable.

Además de mejorar los modelos analógicos, otra expansión de este trabajo sería reforzar la parte que será usada para la validación, incluyendo más test orientados a este ámbito, y tal vez desarrollando el programa que deberá usar el maestro para configurar el ROIC o procesar sus salidas, pudiendo ser este maestro por ejemplo una FPGA. A su vez, puede ser necesaria la verificación de un proyecto más grande, que incluya el maestro, uno o varios ROIC, el sensor, y cualquier otra circuitería. Es importante remarcar que en esta situación habría que verificar las nuevas funcionalidades, el diseño superior (*top*), y no lo que ya se ha verificado del ROIC en este proyecto.

Bibliografía

[1] HandWiki. *Active Pixel Sensor*. Accedido el 8 de Enero de 2024. URL: https://encyclopedia.pub/entry/36045.

[2] Anahita Khoshakhlagh. *Design of a Readout Integrated Circuit (ROIC) for Infrared Imaging Applications*. 2011. URL: https://digitalrepository.unm.edu/ece_etds/135.

[3] Arthur Pini. "Analog Fundamentals: How Sample and Hold Circuits Work and Ensure ADC Accuracy". En: *DigiKey* (2020). URL: https://www.digikey.dk/da/articles/analog-fundamentals-sample-and-hold-circuits-work-adc-accuracy.

[4] Max W. Hauser. "Principles of Oversampling A/D Conversion". En: *Journal Audio Engineering Society* 29.1/2 (1991), págs. 3-26. URL: https://classes.engineering.wustl.edu/2012/fall/ese488/Lectures/DeltaSigmaHandout.PDF.

[5] Bashir, Saima et al. "Analog-to-digital converters: A comparative study and performance analysis". En: 2016. URL: https://ieeexplore.ieee.org/document/7813861.

[6] Walt Kester. "ADC Architectures III: Sigma-Delta ADC Basics". En: (ene. de 2011). URL: https://www.analog.com/media/en/training-seminars/tutorials/MT-022.pdf.

[7] Cadence PCB. *The LVDS Interface*. Accedido el 8 de Enero de 2024. URL: https://resources.pcb.cadence.com/blog/2023-the-lvds-interface.

[8] Nick Holland. *Interfacing Between LVPECL, VML, CML, and LVDS Levels*. Inf. téc. Texas Instruments, 2002. URL: https://www.ti.com/lit/an/slla120/slla120.pdf.

[9] Piyu Dhaker. "Introduction to SPI Interface". En: *Analog Dialogue* (2018). URL: https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html.

[10] Frederic Leens. "An introduction to I2C and SPI protocols". En: *IEEE Instrumentation & Measurement Magazine* 12.1 (2009), págs. 8-13. DOI: 10.1109/MIM.2009.4762946.

[11] Srikanth Vijayaraghavan y Meyyappan Ramanathan. *A Practical Guide for SystemVerilog Assertions*. Springer New York, NY, 2005, págs. 1-88. ISBN: 0-387-26049-8. URL: https://link.springer.com/book/10.1007/b137011.

[12] "IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language". En: *IEEE Std 1800-2017 (Revision of IEEE Std 1800-2012)* (2018), págs. 1-1315. DOI: 10.1109/IEEESTD.2018.8299595.

[13] Neyaz Ahmed Khan y Hao Fang. "Metric Driven Verification of Mixed-Signal Designs". En: 2011. URL: https://dvcon-proceedings.org/wp-content/uploads/metric-driven-verification-of-mixed-signal-designs.pdf.

[14] Ahmet F. Budak et al. “Joint Optimization of Sizing and Layout for AMS Designs: Challenges and Opportunities”. En: *Proceedings of the 2023 International Symposium on Physical Design*. ISPD ’23. New York, NY, USA: Association for Computing Machinery, 2023, págs. 84-92. ISBN: 9781450399784. DOI: 10.1145/3569052.3578929. URL: https://doi.org/10.1145/3569052.3578929.

Parte II

Anexos

ANEXO I. RELACIÓN DEL TRABAJO CON LOS OBJETIVOS DE DESARROLLO SOSTENIBLE DE LA AGENDA 2030

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No procede
ODS 1. Fin de la pobreza.				X
ODS 2. Hambre cero.				X
ODS 3. Salud y bienestar.	X			
ODS 4. Educación de calidad.				X
ODS 5. Igualdad de género.				X
ODS 6. Agua limpia y saneamiento.				X
ODS 7. Energía asequible y no contaminante.				X
ODS 8. Trabajo decente y crecimiento económico.				X
ODS 9. Industria, innovación e infraestructuras.		X		
ODS 10. Reducción de las desigualdades.			X	
ODS 11. Ciudades y comunidades sostenibles.				X
ODS 12. Producción y consumo responsables.				X
ODS 13. Acción por el clima.				X
ODS 14. Vida submarina.				X
ODS 15. Vida de ecosistemas terrestres.				X
ODS 16. Paz, justicia e instituciones sólidas.				X
ODS 17. Alianzas para lograr objetivos.				X

Descripción de la alineación del TFM con los ODS con un grado de relación más alto.

El ODS con el que más se alinea este TFM es el de salud y bienestar. Este consiste en dar acceso a servicios sanitarios esenciales a la parte de la población mundial que aún no lo posee, con el objetivo final de salud para todas las personas. La imagen médica (de la que la radiología digital forma parte) es una ayuda muy valiosa, tanto de forma práctica, con propósitos clínicos como diagnosticar o examinar enfermedades, como de forma teórica, con la ciencia médica que requiere el estudio de la anatomía humana.

El ROIC es solo una pequeña parte del sistema completo que genera la imagen, pero tiene un papel imprescindible. Mientras que el diseño de este puede permitir un sistema más rápido, eficiente y preciso, es la verificación la que garantiza que se cumple lo prometido. Solo asegurando su robustez y fiabilidad puede verse oportuno usar este ROIC en lugar de otros

anteriores.

Este trabajo también se relaciona medianamente con el ODS de industria, innovación e infraestructuras. Además del avance que supone poder producir un chip nuevo y mejorado, la disciplina de verificación es uno de los grandes pilares donde se apoya la industria de semiconductores. Herramientas de verificación como las que se han usado aquí permiten lanzar la compleja y cara producción de un chip con mayor seguridad de que el resultado final será satisfactorio.

Por último, también se podría interpretar una pequeña relación con el objetivo de reducción de las desigualdades. Mejorar la eficiencia y abaratar el coste de la imagen médica favorece que esta pueda ser usada por un mayor sector de la población. En el mismo sentido, la verificación también posibilita chips más baratos, confiables y duraderos, cualidades que les dan acceso a las personas más vulnerables.

ANEXO II. Encuesta anual a los alumnos del MUISE. 05-10-2023

Esta encuesta va dirigida a los alumnos del Máster Universitario en Ingeniería de Sistemas Electrónicos de la UPV que cursaron las asignaturas durante el curso 2022/23.

INSTRUCCIONES: Añade tus respuestas a este documento de Microsoft Word en color azul, añadiendo todas las líneas que necesites. Tras añadir las respuestas, imprímelo en formato PDF y envíalo antes del 18 de octubre de 2023 a las direcciones de email:

ggarcera@eln.upv.es

depeln@upv.es

Preguntas sobre tu situación personal. Añade los detalles que creas oportunos:

SP1) ¿Cuál es tu titulación de acceso al MUISE? ¿Qué especialidad del MUISE has cursado? **Estudié el grado en Ingeniería Industrial y Automática (mención electrónica), he cursado la especialidad digital del MUISE.**

SP2) ¿Has realizado o estás realizando actualmente prácticas de empresa relacionadas con el MUISE? SÍ ¿En qué empresa(s) y qué tipo de actividad? **En ams-OSRAM, en ingeniería de verificación digital.**

SP3) ¿Tienes experiencia profesional anterior a tu ingreso en el MUISE? SÍ ¿En qué empresa(s) y qué tipo de actividad? **En la empresa mallorquina Sampol, en ingeniería de control.**

SP4) ¿Estás actualmente trabajando con una actividad profesional relacionada con el MUISE? SÍ ¿En qué empresa(s) y qué tipo de actividad? **Fui contratado tras las prácticas, por lo que trabajo como ingeniero de verificación en ams-OSRAM.**

SP5) ¿Estás actualmente realizando una tesis doctoral o tienes previsto comenzarla en breve? NO ¿En qué universidad/programa de doctorado? ¿Sobre qué tema?

A continuación te pedimos que rellenes una encuesta para que valores cada apartado con una puntuación de 1 a 5, donde 1 punto="muy poco satisfecho" y 5 puntos="muy satisfecho". En cada pregunta, además de la puntuación sobre el máster en general, puedes añadir comentarios en formato libre sobre asignaturas sueltas.

1) Satisfacción con los conocimientos adquiridos y las competencias desarrolladas. Contenido adecuado de la teoría, las prácticas y los trabajos de diseño. **4, algunas asignaturas me han resultado mucho más útiles, como SDP o PdSFPGA, pero de todas formas estoy satisfecho de haber hecho también el resto. La mayor pérdida de tiempo fue CEP, pues ya había hecho varias asignaturas casi iguales en la carrera, y no estaba interesado más en ese ámbito, pero entiendo su utilidad para los alumnos del grado de telecomunicaciones.**

2) Satisfacción con organización de la enseñanza: horarios, distribución del temario, tiempos, carga lectiva, prácticas… **3. Hubiera preferido que fuera por las mañanas, pero entiendo que este formato es más cómodo para los que trabajan durante el máster. De todas, formas, veo imposible cursar el máster con normalidad (todas las asignaturas) y trabajar al mismo tiempo, aunque sea menos de 8 h, al menos si se pretende disfrutar de los estudios y hacer los trabajos lo mejor posible.**

3) Satisfacción con la coordinación entre los profesores de una misma asignatura, entre asignaturas (que no se repitan contenidos ni coincidan actos de evaluación). Satisfacción con la secuenciación y complementariedad de las asignaturas. Adecuación de los contenidos del máster a la formación previa en función de la titulación de origen de los alumnos. **4. Satisfecho con la coordinación dentro del máster, aunque algunos contenidos fueran algo repetitivos con respecto al grado.**

4) Satisfacción con el proceso enseñanza-aprendizaje: metodologías, actividades formativas, tutorías, seguimiento por parte del profesorado, movilidad e internacionalización, prácticas externas, trabajo fin de máster, etc. **4. Muy satisfecho con todo. Mi única pega sería con las dinámicas estilo flip-teaching, o de tareas a hacer en casa. Estas dinámicas solo funcionan si los profesores asumen que la gente las ha hecho, si repiten lo que se debería haber visto en casa, por si algún alumno no lo hubiera visto, al final solo consiguen que la semana siguiente nadie se moleste.**

5) Satisfacción con los sistemas de evaluación. **3. Los test de PoliformaT pueden ser algo frustrantes, con preguntas ambiguas y que dependen de que el estudiante recuerde un dato casual y en ocasiones irrelevante. Ejemplares los exámenes de Vicente Torres.**

6) Satisfacción con los canales de comunicación empleados por el título y el contenido de la información que facilita (web del título, PoliformaT y avisos por Email). **5**

7) Satisfacción con las instalaciones e infraestructuras destinadas al proceso formativo: aulas, laboratorios, biblioteca, espacios de trabajo, centros colaboradores y asistenciales, etc. **5. La disponibilidad del laboratorio potenció mi aprendizaje y curiosidad por las materias.**

8) Satisfacción por la orientación que les ha suministrado: el DAT (Director Académico del Título), la ERT/Servicios Administrativos del Departamento de Ingeniería Electrónica (Gestión de TFMs, convocatorias y tribunales, actas de asignaturas, …), la Subdirección de Relaciones externas de la Escuela de Ingenieros de Telecomunicación ETSIT (prácticas en empresa y movilidad) y los Servicios Administrativos de la ETSIT (matrícula, altas/bajas de asignaturas, …). **5**

9) Opinión sobre los seminarios profesionales, si procede. **Si has participado en los seminarios del Aula Microelectrónica 2023, también nos gustaría conocer tu opinión**. **3. Estaría bien algún tipo de coordinación para impedir que en dos seminarios se explicaran exactamente las mismas cosas, que sean tan pasivos también los puede hacer algo pesados.**

10) Opinión de las prácticas en empresa, si procede. **5**

11) Opinión sobre la asignatura TFM (Trabajo Fin de Máster) y sobre cómo facilitar que los alumnos finalicen el TFM en un plazo razonable. ¿Qué añadirías a la información sobre el TFM y sus mecanismos de búsqueda de tema/tutor y realización según se describe en la web del MUISE? **4**

12) Si tienes experiencia profesional, has realizado/realizas prácticas en la empresa o conoces los perfiles demandados por las empresas: opinión sobre el perfil de los egresados del MUISE. ¿Se adapta el perfil de los egresados a las demandas de la empresa? ¿Qué aspectos piensas que se podrían reforzar o son redundantes? **4, se adapta bastante dentro de la limitación que conlleva la versatilidad del máster. Tal vez las asignaturas que trabajan con Verilog podrían no aplicarse tanto en el tema de las FPGA, dar algo sobre el diseño microelectrónico, aunque tengo entendido que se va a abrir un máster con ese tema.**

PREGUNTAS SOBRE UN NUEVO PLAN DE ESTUDIOS

El actual plan de estudios del MUISE es de 72 ECTS (60 ECTS en asignaturas + 12 en el TFM). Para adaptarnos al nuevo real decreto que regula los estudios universitarios, el MUISE está en proceso de reforma del Plan de Estudios para arrancar el curso 2024/25 con un nuevo plan de estudios de 90 ECTS. Para idear la reforma pedimos en 2022 y años anteriores la opinión de los alumnos egresados y la de las empresas colaboradoras.

El paso a 90 ECTS se ha realizado de la siguiente manera, siguiendo la opinión de alumnos, empresas y la de los propios profesores del MUISE:

1) El alumnado cursará de manera obligatoria las asignaturas actuales "Diseño Microelectrónico" e "Instrumentación en red y comunicaciones Industriales", una en el primer semestre y la otra en el tercer semestre. En el plan de 72 ECTS el alumnado optaba por una de esas dos asignaturas.

2) Se ha añadido una asignatura común obligatoria nueva de 6 ECTS sobre "Sistemas Embebidos Avanzados", a cursar en el tercer semestre.

3) En la especialidad SED se cursará en el tercer semestre una asignatura nueva de 6 ECTS sobre "Microelectrónica Avanzada".

4) En la especialidad SECE se cursa en el tercer semestre una asignatura nueva de 6 ECTS sobre "Electrónica de Potencia Avanzada".

5) Los dos primeros semestres son idénticos a los del plan de estudios de 72 ECTS. En el tercer semestre se cursará el TFM (12 ECTS) y 18 ECTS en 3 asignaturas: "Sistemas Embebidos Avanzados" (común), "Microelectrónica avanzada" (especialidad SED) o "Electrónica de Potencia Avanzada" (especialidad SECE), y la que no se haya cursado en el primer semestre ("Diseño Microelectrónico" o "Instrumentación en red y comunicaciones Industriales").

El título de 90 ECTS ofrece una formación más completa, la misma duración (3 semestres) y un coste algo mayor que el título de 72 ECTS.

¿Qué opinas del plan de estudios de 90 ECTS anteriormente descrito? ¿Qué contenidos en asignaturas transversales nuevas crees convenientes? ¿Prevés algún problema con ese planteamiento? ***Razona las respuestas y expresa libremente tu opinión****.*

Lo veo correcto, justamente son las asignaturas que me parece que tiene sentido que tuviera también este máster. Esto tal vez además ayude a que muchos estudiantes terminen antes el TFM, esperando a hacer prácticas después del tercer semestre, y haciéndolo con los conocimientos más frescos.

OPINIÓN EN FORMATO LIBRE

Expresa tu opinión en formato libre sobre cualquier otro aspecto del MUISE o de sus asignaturas que no se haya considerado en las preguntas anteriores.