Diseño y caracterización del sistema de instrumentación para un arreglo de microbolómetros

por

Julisa Verdejo Palacios

Tesis presentada en cumplimiento parcial de los requisitos para el grado de:

Maestría en Ciencias en la Especialidad de Electrónica

en

Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE)

Septiembre, 2024

Santa María Tonantzintla, Puebla

Asesor:

Dr. Mario Moreno Moreno
Departamento de Electrónica INAOE

Agradecimientos

- A mi familia, mis padres Rafael y Pilar, mis hermanas Alejandra y Edith y a mi sobrino Diego, gracias por siempre estar al pendiente de mi, por motivarme a seguir adelante, por su amor y apoyo.
- A mi compañero de vida y mejor amigo, Ciro Bermúdez, gracias por tu amor, comprensión y ternura.
- A la Sra. Juana, gracias por su comprensión y por brindarme todo su apoyo en la recta final de mi trabajo.
- A mis amigos, Juan, Yair, Javier, Hitan Divi, Nancy, Vicky y Jorge, gracias por sus consejos, apoyo y compañía.
- A Intesc, gracias por brindarme soporte y asesoría.
- A mis compañeros de laboratorio Victor y Julio, gracias por ayudarme resolviendo mis dudas y por hacer un ambiente agradable de trabajo.
- A mi asesor, gracias por su paciencia y apoyo, el trabajo que realizamos juntos representa el inicio de mi carrera profesional.

Índice general

Índice de figuras

Índice de tablas

Índice de códigos

Resumen

En este trabajo se diseñó e implementó un sistema de adquisición de datos para obtener imágenes a partir de un arreglo de fototransistores que emulan el comportamiento de un microbolómetro. Se utilizó una FPGA Artix 7 para controlar y sincronizar todos los módulos necesarios para la adquisición de datos, entre los cuales se incluyen un módulo ADC, un módulo DAC, dos módulos multiplexores y una PCB con la matriz de fototransistores. Para cumplir con este objetivo, se desarrolló un protocolo SPI personalizado que controla los convertidores analógico-digital (A/D) y digital-analógico (D/A). Además, se llevó a cabo la caracterización de resistencias y multiplexores mediante un circuito de lectura, lo cual incluyó un barrido de voltaje para evaluar las características I/V. También se caracterizó una fotorresistencia bajo diferentes condiciones de iluminación, utilizando un PWM con diversos duty cycles. Finalmente, se diseñó una PCB para la matriz de fototransistores, destinada a obtener imágenes a través de la detección de variaciones lumínicas.

Capítulo 1

Introducción

Los sistemas de adquisición de datos (DAQs) son un conjunto de herramientas utilizadas para recopilar, procesar y analizar datos de fenómenos físicos. Los elementos básicos de un sistema de adquisición de datos incluyen sensores que capturan señales físicas, hardware de acondicionamiento de señales para procesar las salidas de los sensores, y una computadora que ejecuta software para adquirir y registrar los datos.

El concepto de adquisición de datos se remonta a tiempos antiguos, cuando los seres humanos comenzaron a observar y registrar eventos naturales, como el movimiento de los cuerpos celestes, los cambios en el clima y el paso del tiempo. Las primeras civilizaciones desarrollaron métodos para la recolección de datos, como calendarios, instrumentos mecánicos, como el reloj solar, y libros de registros. Con la llegada de dispositivos analógicos y digitales, la adquisición de datos se hizo cada vez más sistemática y automatizada.

Algunos ejemplos de los sistemas de adquisición de datos aplicados en distintos sectores son: estaciones meteorológicas, utilizan DAQs para monitorear parámetros ambientales como la temperatura, la humedad, la velocidad del viento y la presión atmosférica. Estos sistemas suelen incluir sensores como termómetros, higrómetros y anemómetros, los datos se procesan y registran en ordenadores para hacer un seguimiento de los cambios climáticos. En la medicina, los electrocardiógrafos son un tipo de DAQs, miden la actividad eléctrica del corazón, procesan las señales y las muestran en forma de ondas en una pantalla. En la industria automotriz, durante la fase de desarrollo de los vehículos, los DAQs monitorean parámetros como el rendimiento del motor, la eficiencia del combustible y las emisiones de gases para garantizar el cumplimiento en los estándares de calidad y seguridad.

El uso de DAQs para la creación de imágenes con detectores infrarrojos, tiene múltiples aplicaciones, que van desde la seguridad y vigilancia en hogares [1] hasta el ámbito médico, donde se utilizan para el diagnóstico de enfermedades como el cáncer de mama, diabetes [2] y patologías dermatológicas [3]. También se emplean en la detección de

gestos para dispositivos inteligentes [4] y en aplicaciones espaciales, como el monitoreo de las emisiones de dióxido de carbono en la atmósfera provocadas por la actividad humana [5]. En la agricultura, para identificar las zonas del campo que sufren estrés hídrico o exceso de riego, lo que permite una mejor gestión del agua [6].

Los detectores infrarrojos permiten capturar radiación que no es visible al ojo humano y la transforman en señales eléctricas que pueden ser medidas. Desarrollar un sistema de adquisición de datos para obtener imágenes infrarrojas es crucial, ya que estos sistemas interpretan las señales generadas por los detectores y las convierten en una imagen visible, proporcionando información valiosa sobre la distribución de la temperatura en un escenario específico.

Debido a la importancia de los sistemas de adquisición de datos en aplicaciones que utilizan detectores infrarrojos, es fundamental comprender el comportamiento de estos detectores. Conocer cómo responden a diferentes condiciones de luz y temperatura permite diseñar sistemas más eficientes y precisos para capturar imágenes térmicas.

1.1. Radiación Electromagnética

La radiación electromagnética es la emisión y transmisión de energía en forma de ondas electromagnéticas. El espectro electromagnético es una representación de los diversos tipos de radiación existentes, en él se definen los intervalos de longitudes de onda o frecuencia que cada una de ellas abarca [7].

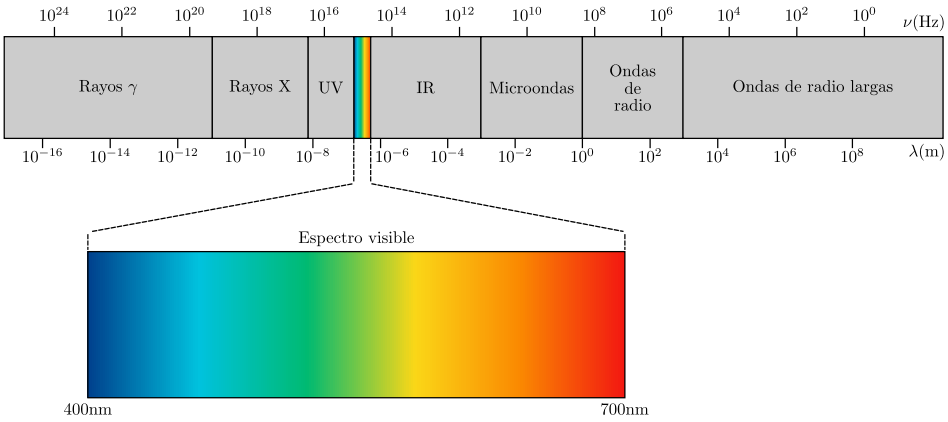


Figura 1.1: Espectro electromagnético.

Estas radiaciones obedecen las mismas leyes, la diferencia entre ellas radica en su longitud de onda o frecuencia, así como en la manera en que interactúan con los materiales ópticos, incluída la atmósfera [8].

Todos los cuerpos emiten radiación e.m y debido al movimiento de sus átomos y moléculas se genera una temperatura en ellos. Los cuerpos con una temperatura mayor a 0K emiten radiación térmica por medio de ondas e.m. La radiación térmica es la radiación que emite un cuerpo por su temperatura [9].

La capacidad que tiene un cuerpo para emitir radiación está fuertemente relacionada con su capacidad de absorberla [10].

Una superficie ideal que absorbe toda la radiación que incide sobre él se denomina *cuerpo negro* y el espectro de radiación que emite se llama *radiación de cuerpo negro* [11].

A pesar de que en la naturaleza no existe un objeto físico que pueda absorber toda la radiación incidente [12], este puede representarse como un objeto hueco con una pequeña apertura donde cualquier radiación que incida en ella ingresa a la cavidad donde queda atrapada hasta que es absorbida [10], [12]. En equilibrio térmico la radiación emitida por el cuerpo será exactamente igual a la absorbida.

El cuerpo negro fue creado como una herramienta auxiliar para entender como los objetos emiten y absorben radiación. Hacia finales del siglo XIX la radiación de cuerpo negro ya había sido estudiada y dos leyes importantes sintetizaron los descubrimientos experimentales sobre este tema: La *Ley de Stefan-Boltzmann* y la *Ley de desplazamiento de Wien* [12].

La *Ley de Stefan-Boltzmann* plantea que la intensidad de la radiación emitida por un cuerpo negro depende de su temperatura. La intensidad es proporcional a la cuarta potencia de la temperatura absoluta del cuerpo:

$$I = \sigma T^4 \tag{1.1}$$

donde:

T es la temperatura del cuerpo negro en K.

σ es la constante de Stefan-Boltzmann, $\sigma = 5.670 \times 10^{-8}\ W/m^2K^4$

La intensidad de la radiación no se distribuye uniformemente a lo largo de todas las longitudes de onda. En cambio, su distribución puede medirse y describirse utilizando la intensidad por intervalo de longitud de onda, $I(\lambda)d\lambda$. La Figura 1.2 muestra las

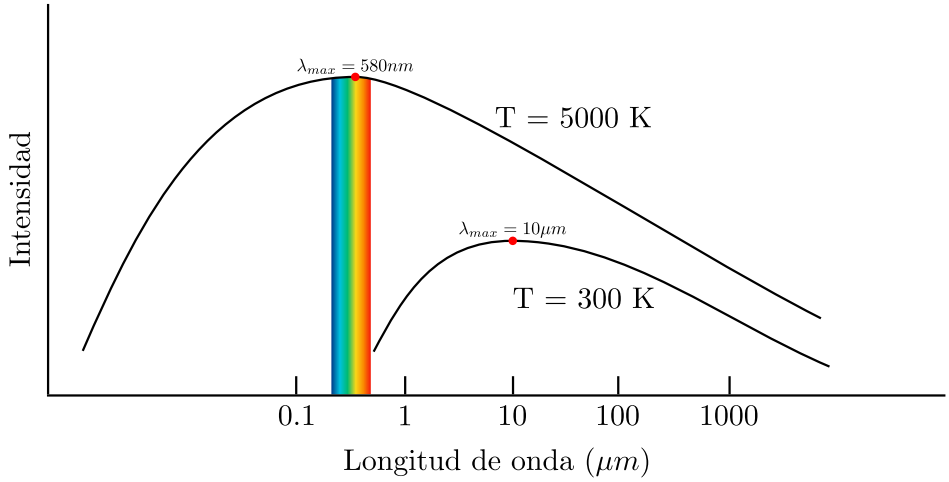


Figura 1.2: Emitancia espectral de un cuerpo negro.

intensidades registradas a dos temperaturas diferentes. En cada caso hay una longitud de onda específica λ_m en la que la intensidad de la radiación emitida es máxima.

La *Ley de desplazamiento de Wien* indica que cuando la temperatura de un cuerpo negro aumenta, la longitud de onda en la que se emite la radiación máxima se desplaza hacia valores más cortos:

$$\lambda_m = \frac{2.90 \times 10^{-3}\ mK}{T} \tag{1.2}$$

donde:

λ_m es la longitud de onda en la que un cuerpo negro emite la mayor cantidad de radiación a una determinada temperatura [11], [12].

De las leyes anteriores y la Figura 1.2 se puede deducir que la radiación UV, los rayos X y Gamma, son radiaciones más cálidas (radiaciones de alta energía), mientras que la radiación infrarroja está asociada a fenómenos con temperaturas cercanas a la temperatura ambiente [7], [13].

1.2. Radiación Infrarroja

La radiación infrarroja es un tipo de radiación electromagnética que cuenta con longitudes de onda mayores que las del rango visible. Se encuentra en el rango de $0.78\mu m$ - $1mm$ [13], y a su vez se divide en varias regiones las cuales se muestran en la Tabla 2.1 [14].

Tabla 1.1: División de la radiación infrarroja.

Region	**Rango de frecuencia (μm)**
Near infrared (NIR)	0.78 - 1
Short wavelength IR (SWIR)	1 - 3
Medium wavelength IR (MWIR)	3 - 6
Long wavelength IR (LWIR)	6 - 15
Very long wavelength IR (VLWIR)	15 - 30
Far infrared (FIR)	30 - 100
Submillimeter (SubMM)	100 - 1000

Algunas de las aplicaciones de la radiación infrarroja son:

- **Visión nocturna**: Las cámaras de visión nocturna trabajan en el espectro infrarrojo para permitir la visión en la oscuridad, estas capturan la radiación térmica emitida por objetos y seres vivos.
- **Medicina**: Es utilizada para hallar cáncer y diabetes en el cuerpo humano.

- **Industria**: Inspección del estado de equipos eléctricos y mecánicos.
- **Conservación de energía**: Con escáneres IR se detectan pérdidas y fugas de calor en casas o industrias.
- **Ambientales**: Medición de la concentración de diversos gases contaminantes en la atmósfera.
- **Agricultura**: Monitoreo del estado de los cultivos y la salud de las plantas, la humedad del suelo y la presencia de plagas o enfermedades.
- **Astronomía**: Los telescopios infrarrojos permiten estudiar regiones del espacio donde se están formando estrellas.
- **Espectroscopía**: Usada en química y biología para identificar y analizar estructuras moleculares de sustancias.

[14], [13].

La mayoría de las aplicaciones en detección de radiación infrarroja requieren que esta se transmita a través del aire [15]. La atmósfera terrestre se compone de ozono (O_3), dióxido de carbono (CO_2) y vapor de agua (H_2O). Estas moléculas bloquean algunas regiones del espectro infrarrojo, impidiendo la transmisión de la radiación IR a la atmósfera. Las longitudes de onda que no son afectadas por estas moléculas reciben el nombre de ventanas atmosféricas [14], [16]. En la Figura 1.3 podemos observarlas.

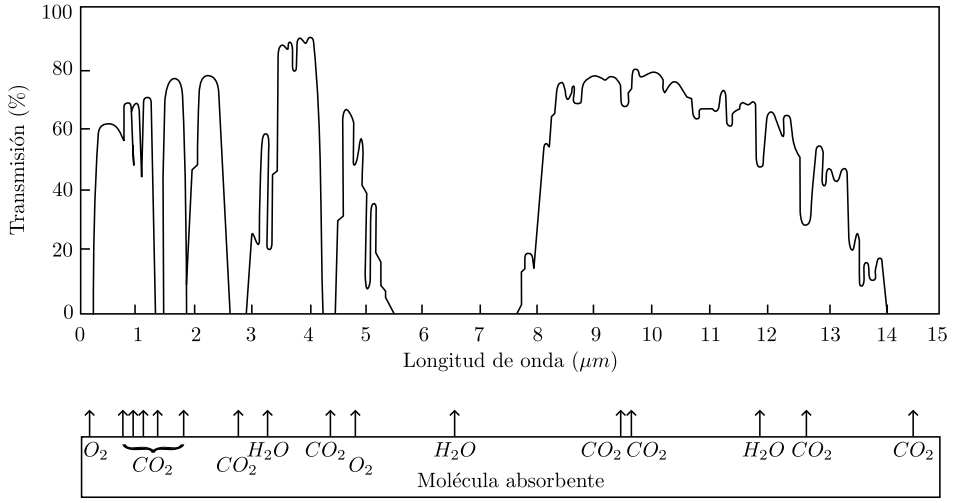


Figura 1.3: Transmisión de la atmósfera.

Despejando la temperatura de la ecuación 1.2 [13]:

$$T_{max} = \frac{2.90 \times 10^{-3}\ mK}{8\mu m} = 362.5K \tag{1.3}$$

$$T_{min} = \frac{2.90 \times 10^{-3}\ mK}{14\mu m} = 207.14K \tag{1.4}$$

podemos deducir que los detectores infrarrojos que operan en la ventana de 8 - 14 μm tienen mayor sensibilidad a la temperatura ambiente [14].

El material de fabricación y longitud de onda de operación de los detectores infrarrojos cambian de acuerdo a la aplicación en la que se desee trabajar [14].

1.3. Detectores Infrarrojos

Un detector infrarrojo es un dispositivo capaz de absorber parte de la energía infrarroja radiada hacia él, provocando una variación en alguna de sus propiedades eléctricas [13]. Podemos pensar en un detector infrarrojo como un transductor, el cual convierte un tipo de señal en otra; el detector infrarrojo convierte la radiación infrarroja incidente en una señal eléctrica [8].

Dependiendo de la aplicación, el rango del espectro electromagnético y la temperatura en los que se desee trabajar, se debe diseñar o utilizar un detector específico que cumpla con los requerimientos, ya que cada aplicación requiere características diferentes a las demás [14], [13].

Los detectores infrarrojos se pueden clasificar en dos categorías: *Detectores de fotones* y *detectores térmicos*.

1.3.1. Detectores de fotones

En los detectores de fotones, la formación de pares electrón-hueco es consecuencia de la detección de radiación absorbida [13]. En estos dispositivos, el fotón que incide sobre la superficie del material puede ser absorbido, lo que resulta en la remoción de un electrón desde la banda de valencia o desde un estado localizado dentro de la banda prohibida hasta la banda de conducción.

Generalmente, los detectores de fotones que operan a longitudes de onda de aproximadamente $3\mu m$ requieren de un sistema de enfriamiento. Esto es necesario para prevenir la generación térmica de portadores de carga, la cual es su principal desventaja, ya que aumenta su costo y complejidad de operación. No obstante, su resolución y desempeño son bastante buenos [8], [13].

1.3.2. Detectores térmicos

El funcionamiento de los detectores térmicos se basa en la generación de una salida eléctrica medible como resistencia o diferencia de potencial debido al cambio de temperatura del material causado por la radiación incidente. Estos detectores pueden

funcionar a temperatura ambiente, son económicos, compactos, de bajo consumo energético y tienen una larga vida útil, pero su capacidad de detección es baja [14], [13]. Algunos de los principales detectores térmicos infrarrojos son explicados a continuación.

1.3.2.1. Celda de Golay

La celda de Golay es un detector térmico principalmente utilizado en espectroscopía infrarroja, la celda consiste en una cámara herméticamente cerrada llena de gas y un espejo flexible. A medida que la radiación infrarroja incide, el gas se calienta y se expande, provocando que el espejo flexible se mueva. El movimiento del espejo desvía un haz de luz que incide sobre otro detector generando un cambio en su irradiancia, después esa señal es procesada [8], [13].

1.3.2.2. Bolómetros y Microbolómetros

Los bolómetros y microbolómetros son dispositivos que detectan la radiación infrarroja mediante el cambio en la resistencia eléctrica de un material al variar su temperatura. Conforme la resistencia absorbe calor, su temperatura aumenta y su resistencia se modifica. Este tipo de sensores deben polarizarse con corriente o voltaje para que puedan funcionar. Si son polarizados con corriente, el cambio de la resistencia se puede detectar y medir como un cambio de tensión, pero si se polariza con voltaje entonces se detectará como un cambio de corriente [14], [8].

1.3.2.3. Termopares y Termopilas

Los termopares y las termopilas están formados por la unión de dos conductores diferentes unidos por un extremo. Cuando la temperatura aumenta en esa unión, se genera una diferencia de potencial. Al conectar en serie varias uniones de conductores, se produce una tensión más elevada y, por lo tanto, medible [14], [8].

1.3.2.4. Detectores piroeléctricos y ferroeléctricos

Estos detectores pueden visualizarse como un capacitor con dos electrodos metálicos colocados perpendicularmente. Cuando un detector piroeléctrico absorbe radiación, su temperatura cambia, lo que genera una carga en el capacitor y una corriente que depende del cambio de temperatura. Si la temperatura permanece constante, la corriente será nula.

Los detectores ferroeléctricos operan bajo el mismo principio que los piroeléctricos, pero con la diferencia de que el efecto se produce mediante un campo eléctrico [14], [13].

En los últimos años, ha habido un creciente interés en el uso de microbolómetros para generar imágenes infrarrojas. En el ámbito médico, se han utilizado para medir

la temperatura corporal [17] y, de manera innovadora, en el diagnóstico de la diabetes, mediante la captura de imágenes térmicas de la lengua [18] o del pie [19], permitiendo identificar variaciones térmicas que pueden indicar problemas de salud. Otra aplicación importante es en cámaras instaladas en vehículos, donde los microbolómetros se emplean para detectar personas o mascotas una vez detenido el vehículo, ayudando a prevenir accidentes ocasionados por quedar atrapados dentro [20].

Este creciente interés se debe, en parte, a su bajo costo, peso ligero y dimensiones reducidas, que pueden ser menores a 12 μm, lo que permite una mayor cantidad de pixeles en un arreglo. Además, mientras que algunos sensores infrarrojos están limitados a resoluciones de 120×84 pixeles, los microbolómetros pueden alcanzar resoluciones mucho más altas, como 1280×1024 o más, obteniendo imágenes más detalladas y nítidas. Estas ventajas hacen de los microbolómetros una opción preferida en tecnologías de imágenes térmicas de alta precisión. Por este motivo, en este trabajo se desarrollará un sistema de adquisición de datos modular, que puede ser utilizado con un microbolómetro. En la siguiente tabla se presentan trabajos similares.

Tabla 1.2: Resumen de trabajos relacionados son sistemas de lectura para detectores.

Artículo	Dispositivo de implementación	Tecnología de detección	Sistema configurable	Interfaz gráfica	Arreglo máximo de detectores
[20]	TSK3000 Microcontrolador incluido en FPGA Altera company	Microbolómetro no refrigerado de Silicio-amorfo (a-Si)	No	Integrada	168x120
[21]	FPGA EPC2C35F672 de Altera company	UL 03 04 (comercial) Microbolómetro de Silicio-amorfo (a-Si)	No	No	384x288
[22]	FPGA EPC3C40 de Altera company	Microbolómetro con pozo cuántico de Silicio (Si), Silicio-germanio (SiGe)	No	Sí	384x288
[23]	Microcontrolador Cortex M3 STM32F103	Bolómetro en sustrato de Óxido de aluminio (Al2O3), Silicio (Si)	No	No	1x1
[24]	FPGA Cyclone IV EP4CE55 de Altera company	GWR1020 IRFPA de Óxido de vanadio (VOx)	No	No	384x288
[25]	DSP TMS320DM6467	Microbolómetro UL04371 (comercial)	No	No	160x120
[26]	Raspberry Pi 2	Lepton FLIR 2.5 (comercial): Microbolómetro de VOx no refrigerado	No	Si	80x60
[2]	Raspberry Pi 3 B+	Lepton FLIR (comercial): Microbolómetro de VOx no refrigerado	No	No	160x120
[19]	Raspberry Pi	Lepton FLIR 2.5 (comercial): Microbolómetro de VOx no refrigerado	No	No	80x60
[27]	DROIC	Infrared pixel array	Si	Si	1024x1024

1.4. Objetivos

1.4.1. Objetivo general

- Diseño de un sistema de adquisición de datos basado en FPGA para la medición de una matriz de pixeles de un microbolómetro.

1.4.2. Objetivos específicos

- Analizar las propiedades físicas y características de los microbolómetros, así como su funcionamiento y sus diferentes circuitos de lectura con el fin de fundamentar las decisiones en el diseño del sistema.

- Estudiar las máquinas de estado finito (FSM) de tipo Moore y Mealy, los tipos de codificación en Verilog y las técnicas de diseño, con el fin de aplicar este conocimiento en la implementación y optimización del sistema de adquisición y procesamiento de datos.

- Diseñar un firmware en el lenguaje Verilog para implementar los protocolos de comunicación SPI y UART reconfigurables y robustos, que permitan adaptabilidad y confiabilidad en aplicaciones de adquisición de datos y control.

- Seleccionar los convertidores analógico-digital (ADC) y digital-analógico (DAC) más adecuados para el sistema de adquisición de datos y probarlos utilizando el firmware previamente diseñado.

- Diseñar una PCB con una matriz de 8x8 fototransistores que funcione como banco de prueba para validar el funcionamiento del sistema de adquisición de datos.

- Integrar los convertidores ADC y DAC, multiplexores, y la matriz de fototransistores en un sistema unificado para realizar pruebas y validar su capacidad de adquirir y procesar imágenes de manera eficiente.

Capítulo 2

Microbolómetros y descripción de especificaciones

Los arreglos de microbolómetros son detectores de infrarrojos ampliamente utilizados en aplicaciones de imágenes térmicas. Están formados por una matriz bidimensional de microbolómetros, que son diminutos resistores sensibles a la temperatura diseñados para captar la radiación infrarroja que emiten los objetos dentro del campo de visión del detector. Estos sensores están montados sobre un sustrato de silicio y generalmente se fabrican utilizando tecnología MEMS (sistemas electromecánicos). Los detectores de este tipo ofrecen una resolución espacial y sensibilidad bastante altas, lo que permite que las matrices detecten y diferencien con precisión objetos con mínimas variaciones de temperatura.

Una ventaja adicional es que no necesitan enfriamiento externo, lo que hace que los sistemas de imágenes térmicas sean mas portátiles, fáciles de operar y adecuados para una amplia gama de aplicaciones [28].

Este dispositivo opera mediante una corriente o voltaje de polarización controlada que atraviesa el detector mientras se monitorea el voltaje o corriente de salida. En este proceso, la energía radiante genera calor en el material, lo que altera la resistencia, sin interacción directa entre fotones y electrones.

El astrónomo S.P. Langley diseñó el primer bolómetro en 1880, utilizando platino ennegrecido como material absorbente y un puente de Wheatstone como circuito de detección. La tecnología moderna de bolómetros comenzó en la década de los 80's con los avances de Honeywell en óxido de vanadio (VOx) y de Texas Instruments en silicio amorfo (a-Si). Mucho de este desarrollo ocurrió bajo proyectos militares clasificados en Estados Unidos, por lo que la divulgación de esta información a la comunidad científica data desde 1992.

Es muy difícil tener una representación exacta de un microbolómetro, pero este se puede representar como un circuito divisor de voltaje (Figura 2.1). Si el circuito está

abierto y no hay radiación presente, el microbolómetro se encontrará a temperatura ambiente T_0, si el circuito está cerrado, la corriente fluirá y la resistencia del microbolómetro, R_B, se calentará, esto provocará que la temperatura incremente a T_1. Si ahora la radiación cae sobre el microbolómetro, la temperatura cambiará en un factor de ΔT. Esto causará una modificación en la resistencia del microbolómetro, lo que a su vez generará una variación en el voltaje (V_{RL}) a través de la resistencia de carga R_L [14].

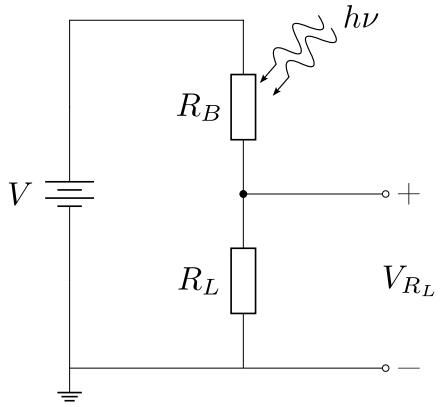


Figura 2.1: Representación de un microbolómetro como circuito eléctrico.

2.1. Diseño de un arreglo de microbolómetros

El diseño de un microbolómetro generalmente incluye un material absorbedor y un material que funge como termómetro, lo que lleva a un incremento de la temperatura debido a la absorción de radiación infrarroja, provocando finalmente un cambio en la resistencia de sus elementos. La variación en la resistencia se transmite eléctricamente al circuito integrado de lectura (ROIC) para su posterior procesamiento. Para aumentar la sensibilidad, el termómetro se mantiene aislado térmicamente del sustrato del ROIC [29].

El diagrama esquemático de la estructura típica de un microbolómetro se muestra en la Figura 2.2.

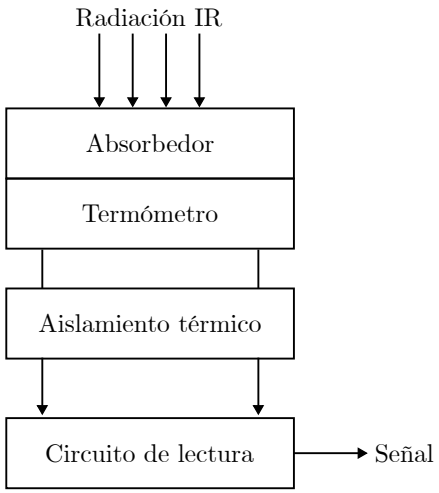


Figura 2.2: Diagrama esquemático de un microbolómetro.

Un píxel en un arreglo de microbolómetros puede tener un área que varía entre 15×15 μm^2 y 55×55 μm^2. La estructura de un pixel, mostrada en la Figura 2.3, generalmente es descrita como un *puente*, el cual se trata de una membrana suspendida que incluye una capa absorbente para la radiación infrarroja y un elemento termosensible que transforma el cambio de temperatura de la membrana en una señal eléctrica de salida. El *piso* de este *puente* es un circuito de lectura unitario y las dos *rampas* son brazos de soporte utilizados para la conexión eléctrica, estos ayudan a mejorar el aislamiento térmico [8], [30].

Para un microbolómetro es esencial contar con una temperatura estable, esto con la finalidad de tener una mejor detección de radiación infrarroja. El vacío provee un buen aislamiento térmico, debido a que la pérdida de calor por conducción o convección es mínima. Mientras más vacío se tenga en el envase del detector mejor será la eficiencia [31]. Generalmente el microbolómetro requiere una atmósfera de vacío menor a 1 Pa para aislar la transferencia de calor y mejorar su respuesta [32].

2.2. Propiedades

En esta sección se definirán algunas propiedades importantes que son utilizadas para caracterizar el desempeño de microbolómetros.

2.2.1. Responsividad

La responsividad (R) se refiere a la capacidad que tiene un detector de convertir radiación incidente en una señal eléctrica

$$R = \frac{señal\ de\ salida}{radiación\ de\ entrada} \tag{2.1}$$

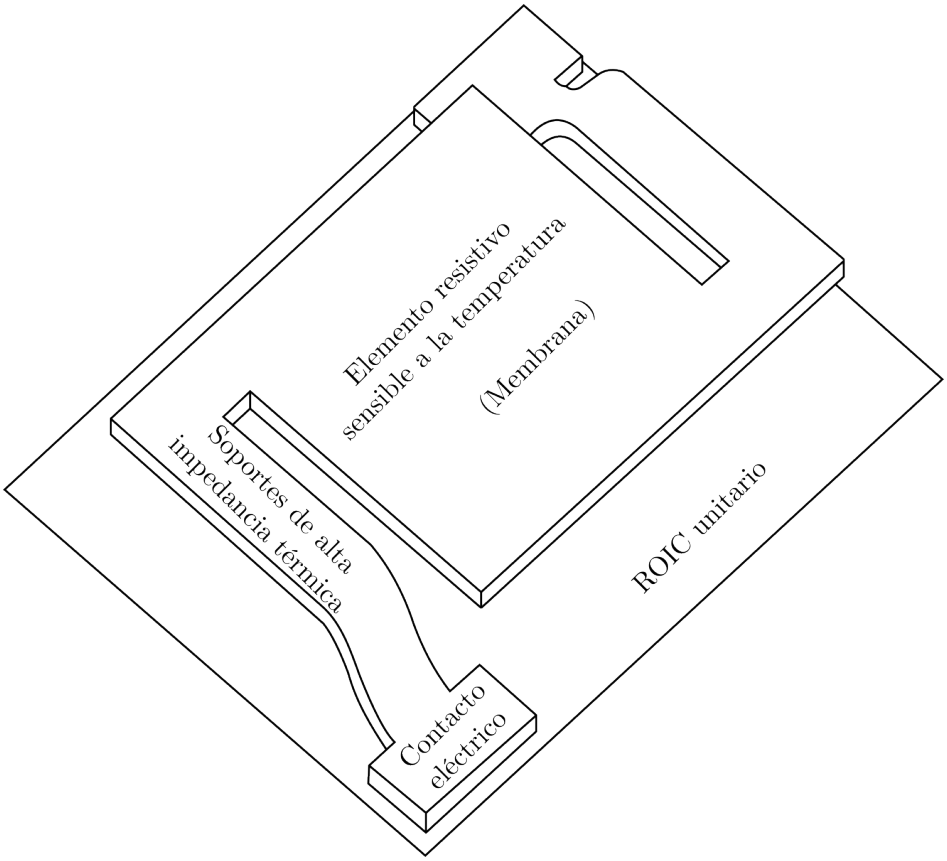


Figura 2.3: Estructura de un pixel de un arreglo de microbolómetros.

Para los microbolómetros, generalmente la radiación de entrada se define en términos de flujo radiante (Φ_S), el cual es el producto de la irradiancia (E) por el área del detector (A_d) y la señal de salida puede ser voltaje (V_S) o corriente (I_S).

$$R = \frac{S}{\Phi_S} \tag{2.2}$$

La responsividad de voltaje R_V se define como:

$$R_V = \frac{V_S}{EA_d} \quad [V/W] \tag{2.3}$$

La responsividad de corriente es:

$$R_I = \frac{I_S}{EA_d} \quad [A/W] \tag{2.4}$$

La responsividad es un parámetro crucial para un detector, ayuda a anticipar la sensibilidad del circuito de medición necesario para observar la salida esperada o a decidir el nivel de ganancia del amplificador requerido para amplificar la señal adecuadamente [8], [30].

2.2.2. Diferencia de temperatura equivalente al ruido

La diferencia de temperatura equivalente al ruido (Noise Equivalent Temperature Difference - $NETD$), indica el cambio mínimo de temperatura que un microbolómetro puede detectar, reflejando su capacidad para distinguir pequeñas diferencias en la radiación térmica. Un valor de $NETD$ más bajo significa mayor sensibilidad térmica [15], [30].

El NETD se define con la siguiente ecuación

$$NETD = \frac{4F^2 V_N}{\tau_0 A_D R(\Delta P/\Delta T)_{\lambda_1-\lambda_2}} \tag{2.5}$$

Donde V_N es la tensión del ruido total; F y τ_0 son parámetros que tienen en cuenta la óptica del sistema; $(\Delta P/\Delta T)\lambda_1 - \lambda_2$, es el cambio en la potencia emitida por unidad de superficie de un cuerpo negro dentro de una banda espectral específica, de λ_1 a λ_2, en función de la temperatura, y se expresa en Kelvin [28].

2.2.3. Detectividad

La detectividad (D) es un parámetro utilizado para comparar el desempeño entre distintos detectores con diferentes tamaños. Mientras más elevada sea la detectividad, mejor será el detector.

La detectividad se calcula de la siguiente manera:

$$D = \frac{R_V\sqrt{A\Delta f}}{V_N} \quad [cm\sqrt{Hz}/W] \tag{2.6}$$

Donde R_V es la responsividad de voltaje; A es el área del detector; ΔF es el ancho de banda del ruido del detector y V_N es ruido total del detector [8], [15], [30], [29].

2.2.4. Conductancia térmica

La conductancia térmica (G_{th}) mide la facilidad con la que fluye el calor a través de un material. En el caso de un microbolómetro, cuantifica la velocidad a la que la energía térmica se transfiere del elemento detector a su entorno o al disipador de calor. La conductancia se puede obtener mediante la siguiente ecuación:

$$G_{th} = 2K\frac{A}{l} \quad [W/K] \tag{2.7}$$

Donde K es el coeficiente de conductividad térmica del material de los brazos de soporte del microbolómetro; A es el área transversal del brazo del detector; l es la longitud del brazo [13], [15] [29].

2.2.5. Capacitancia térmica

La capacitancia térmica (C_{th}) determina cuánto calor puede almacenar el detector. Con la siguiente ecuación podemos calcularla.

$$C_{th} = c\rho v \quad [J/K] \tag{2.8}$$

Donde c es el calor específico del material; ρ es la densidad del material con el que esté fabricado el pixel del microbolómetro; v es el volumen de la membrana del microbolómetro [13], [15] [29].

2.2.6. Coeficiente de temperatura de la resistencia

El coeficiente de temperatura de la resistencia (Thermal Coefficient of Resistance - TCR) se define como la variación de la resistencia del microbolómetro (R_B) debido al cambio de temperatura:

$$\alpha_B = TCR = \frac{1}{R_B}\frac{dR_B}{dT_S} \tag{2.9}$$

El cambio en la resistencia del microbolómetro en función de la temperatura depende de si el material es un metal o un semiconductor [30], [33].

2.2.7. Tiempo de respuesta térmico

El tiempo de respuesta térmico, τ_{th}, se refiere al período que el sensor necesita para estabilizar la señal de corriente o voltaje que se mide tras un cambio en la temperatura. El cálculo de τ_{th} se obtiene con la siguiente ecuación [15].

$$\tau_{th} = \frac{C_{th}}{G_{th}} \quad [s] \tag{2.10}$$

Si la membrana es demasiado voluminosa, el detector responderá lentamente a los cambios de temperatura, mientras que una alta conductancia térmica permitirá una respuesta rápida. Por lo tanto, el diseño del microbolómetro implica un equilibrio entre la velocidad de respuesta y la sensibilidad [15].

Todos estos parámetros pueden ser evaluados mediante la obtención de las curvas V-I del microbolómetro mientras opera en un entorno de vacío y está expuesto a iluminación infrarroja [34].

La clave para desarrollar microbolómetros altamente sensibles es tener un coeficiente de temperatura alto α, una masa térmica muy baja C_{th} y un excelente aislamiento térmico (baja conductancia térmica) G_{th} [14].

2.3. Circuitos de lectura para un microbolómetro

Un circuito integrado de lectura (*Readout Integrated Circuit* - ROIC), es un elemento esencial en los sistemas de imagen térmica. En los microbolómetros está integrado debajo de cada pixel. Los ROICs contienen todos los módulos necesarios para alimentar el microbolómetro, captar la variación de resistencia generada por la radiación infrarroja incidente, ajustarla y amplificarla para su posterior procesamiento [13], [30], [28]. El principal reto de un ROIC es disminuir el calentamiento del detector, para evitar información térmica errónea y que se incremente el riesgo de daño permanente [35].

A continuación se presentan algunas topologías utilizadas como ROICs.

2.3.1. Divisor resistivo

Esta topología está diseñada para medir la tensión en la salida de un divisor resistivo formado por un dispositivo de referencia y un microbolómetro, como se muestra en la Figura 2.4. A medida que aumenta la radiación incidente en el microbolómetro, su resistencia disminuye proporcionalmente, lo que hace que la tensión de salida dependa de la radiación recibida por el sensor [13].

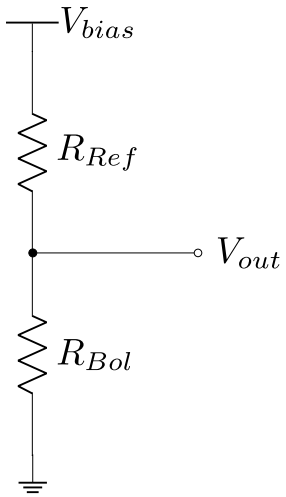


Figura 2.4: Divisor resistivo.

La tensión de salida se expresa con la siguiente ecuación:

$$V_{out} = \frac{V_{bias}}{R_{ref} + R_{bol}} = V_0 - \Delta V_{out} \tag{2.11}$$

Donde:

V_0 - Valor de la tensión de salida en equilibrio térmico (sin radiación incidente)

ΔV_{out} - Valor de la disminución de la tensión de salida cuando hay radiación incidente sobre el microbolómetro.

Esta topología destaca por su simplicidad, bajo consumo de potencia, por atenuar el efecto de calentamiento en cada pixel del microbolómetro y por ocupar un área pequeña, no obstante, no tiene una restricción de frecuencia para el ruido, lo que impacta negativamente en la calidad de la señal de salida.

2.3.2. Puente de Wheatstone

Este tipo de ROIC obtiene la tensión de forma diferencial utilizando un puente Wheatstone, que está compuesto por dos ramas: una de referencia y otra que contiene el microbolómetro, como se muestra en la Figura 2.5.

La parte superior de cada rama está compuesta por una resistencia de referencia, ambas con la misma magnitud resistiva ($R_{Ref_1} = R_{Ref_2}$). En la parte inferior de una de las ramas se encuentra el microbolómetro sensor, mientras que en la rama de referencia se ubica un microbolómetro aislado de la potencia incidente [13].

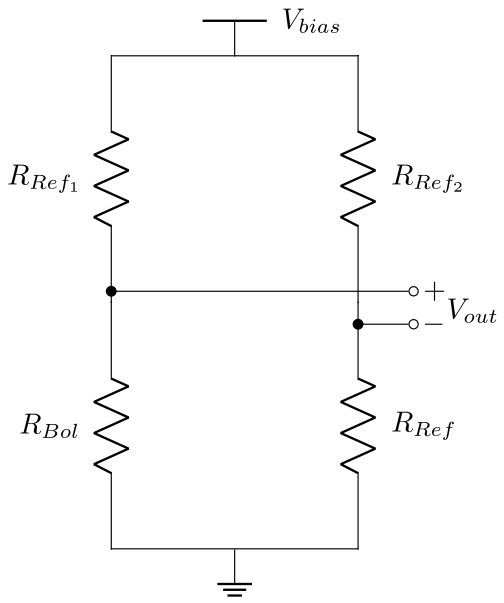


Figura 2.5: Puente de Wheatstone.

La tensión de salida es

$$V_{out} = \left(\frac{V_{bias}}{R_{Ref_1} + R_{Bol}} R_{Bol} - \frac{V_{bias}}{R_{Ref_2} + R_{Ref}} R_{Ref} \right) \tag{2.12}$$

La implementación del puente de Wheatstone como readout posibilita eliminar las contribuciones de voltaje en la salida causadas por el autocalentamiento debido a la polarización y las variaciones en los terminales de alimentación. No obstante, para alcanzar este objetivo, se incrementa tanto el área ocupada como el consumo de potencia.

2.3.3. BCDI (Bolometer Current Direct Injection)

Este circuito de lectura emplea un capacitor en el nodo de salida, el cual facilita el filtrado de frecuencias, lo que contribuye a disminuir el nivel de ruido. No obstante, la frecuencia de corte en la salida está determinada por el valor de la resistencia del microbolómetro. A medida que aumenta la potencia incidente, la resistencia disminuye y la frecuencia de corte se incrementa [13].

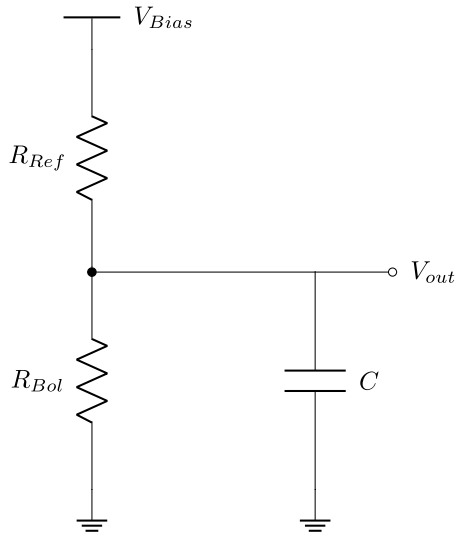


Figura 2.6: Readout BCDI.

La tensión de salida se encuentra dada por la siguiente ecuación:

$$V_{out} = \frac{V_{Bias} R_{Bol}}{R_{Ref}(sR_{Bol}C + 1) + R_{Bol}} \tag{2.13}$$

La ventaja principal del BCDI, es su capacidad para limitar la frecuencia, lo que mejora la relación señal-ruido. No obstante, las contribuciones de voltaje en la salida debido al autocalentamiento y las variaciones en los terminales de polarización no se eliminan por completo.

2.3.4. CTIA (Capacitive Trans-Impedance Amplifier)

Esta topología mostrada en la Figura 2.7, presenta dos ventajas clave en comparación con otros circuitos de lectura. Primero, el uso de un amplificador operacional reduce considerablemente el ruido de la señal, lo que hace que esta topología sea especialmente adecuada para circuitos de lectura en detectores de alta resistividad sin refrigeración, así como en detectores de tipo diodo. La segunda ventaja importante es que se mantiene un nodo a una tensión constante, compartido entre la rama de referencia y la rama donde se encuentra el sensor, lo que asegura que el dispositivo de referencia esté polarizado de la misma manera que el microbolómetro [13].

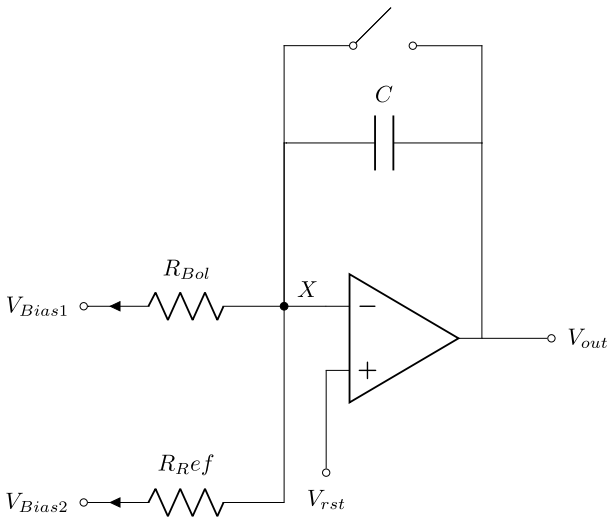


Figura 2.7: Readout CTIA.

Al mantener una tensión constante en el nodo X, se establece un nodo de tierra virtual que permite el flujo de la diferencia de corriente entre las dos ramas. Esta diferencia de corriente es integrada por el capacitor, convirtiéndose en una señal de tensión. Si la tensión V_{Bias1} se reduce a $0V$, la tensión de salida se puede describir según lo indicado en la siguiente ecuación [13]

$$V_{out} = \left(\frac{V_{Bias2} - V_{rst}}{R_{Ref}} - \frac{V_{rst}}{R_{Bol}} \right) \frac{1}{sC} \rightarrow \Delta V_{out} \approx i_c \frac{\Delta t}{C} \tag{2.14}$$

Este readout ofrece ventajas como la eliminación del efecto de calentamiento en la salida y la reducción del ruido gracias al uso de un capacitor que actúa como integrador de corriente. Sin embargo, su consumo de potencia es superior en comparación con las configuraciones anteriores, debido a la inclusión del amplificador, cuyo consumo de energía dependerá del diseño implementado.

2.3.5. WBDA (Wheatstone Bridge Differential Amplifier)

Este tipo de circuito de lectura se basa en el principio de un puente de Wheatstone, del cual se toma la tensión diferencial entre sus ramas, tal como se ilustra en la Figura 2.8. Esta diferencia de tensión se amplifica mediante un amplificador de transconductancia o un amplificador de tensión. Finalmente, la señal de salida de la etapa de amplificación es integrada para reducir el ancho de banda y mejorar la relación señal/ruido.

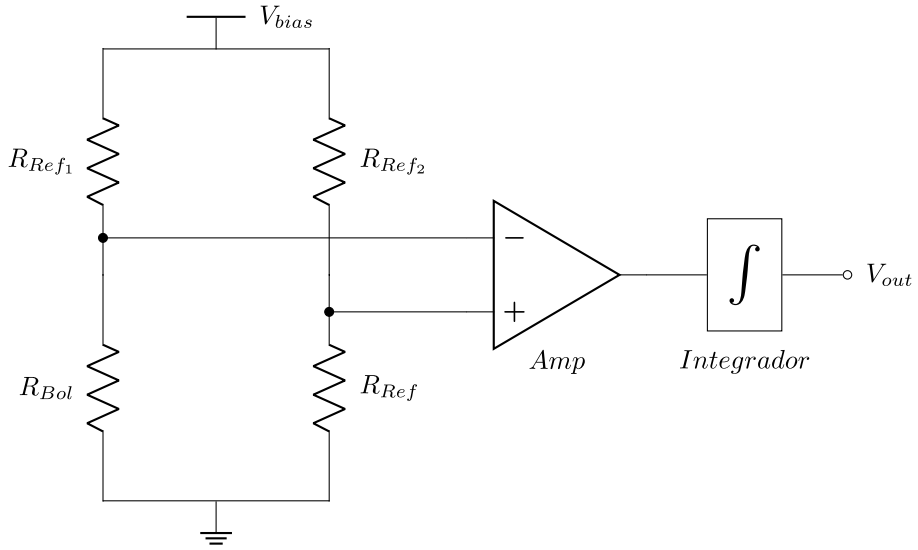


Figura 2.8: Readout WBDA.

En el caso donde la etapa de amplificación está compuesta por un amplificador de transconductancia y la etapa de integración por un capacitor, la tensión de salida se puede aproximar como

$$V_{out} = \frac{\frac{V_{Bias}}{R_{Ref}}}{1 + \frac{R_{Ref}}{R_0}} \cdot (\Delta R) \cdot G_m \cdot \frac{1}{sC} \tag{2.15}$$

El rendimiento de este readout compensa eficazmente el efecto de calentamiento y limita la banda de frecuencia para reducir el ruido en la señal. No obstante, para lograr la compensación del calentamiento, se utiliza un puente de Wheatstone, lo que incrementa el área ocupada. Además, las etapas de amplificación y filtrado de la señal también contribuyen a un mayor consumo de potencia.

2.3.6. CCBDI (Constant Current Buffered Direct Injection)

Este método de readout se basa en medir la diferencia de voltajes entre un microbolómetro polarizado con una corriente I_0 y un voltaje de referencia generado por un amplificador de transconductancia. El diagrama del circuito se presenta en la Figura 2.9.

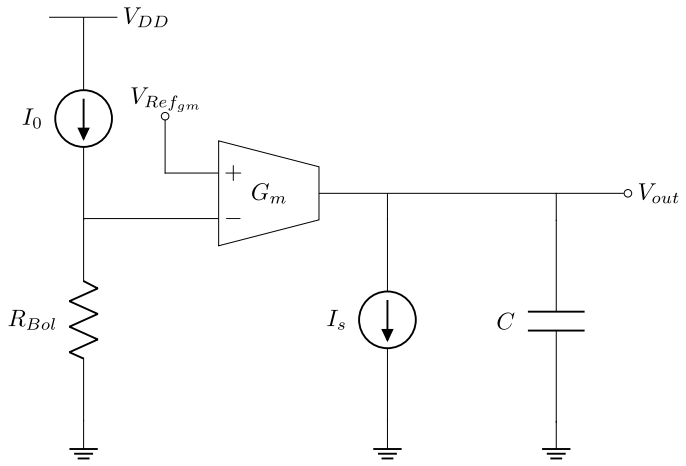


Figura 2.9: Readout CCBDI.

La diferencia de voltaje entre el voltaje aplicado del microbolómetro, V_{Bol} y el voltaje de referencia del amplificador, $V_{Ref_{gm}}$, es convertida en corriente mediante el amplificador de transconductancia. En la etapa final, se incluye una corriente de referencia, I_s, cuya función es restar parte de la corriente amplificada; finalmente, un capacitor actúa como un integrador en la salida. Idealmente, el voltaje de salida de este circuito se describe mediante la ecuación que se muestra a continuación

$$V_{out} = \frac{G_m(V_{Bol} - V_{Ref_{gm}}) - I_s}{sC} \tag{2.16}$$

Este readout reduce en su totalidad el ruido mediante la disminución del ancho de banda en frecuencia a expensas de un mayor consumo de energía y un aumento en el área ocupada.

Los circuitos de lectura empleados para detectar señales de microbolómetros tienen como objetivo principal minimizar el ruido y reducir el calentamiento del detector, lo cual es esencial para preservar la precisión y sensibilidad del dispositivo. Sin embargo, existe un compromiso inherente en el diseño de estos circuitos: cuanto más efectivo sea el circuito en disminuir el ruido y los efectos térmicos, mayor será su consumo de potencia y el área ocupada en el chip. Este compromiso entre eficiencia en la reducción de ruido y optimización de recursos es un factor crucial en el diseño de sistemas de lectura para microbolómetros.

En la siguiente tabla se muestra un resumen de las topologías mencionadas anteriormente.

Tabla 2.1: Topologías ROIC.

Topología	**Reducción de ruido**	**Reducción de calentamiento**	**Consumo de potencia**	**Área ocupada**
Divisor resistivo		•	↓	↓
Puente de Wheatstone		•	↑	↑
BCDI	•		↓	↓
CTIA	•	•	↑	↑
WBDA	•	•	↑	↑
CCBDI	•		↑	↑

2.4. Generación de imágenes infrarrojas con microbolómetros

La señal proveniente de los microbolómetros debe ser acondicionada y convertida a formato digital para su almacenamiento y procesamiento. Estos datos permiten crear una imagen térmica [35].

El sistema para generar una imagen a partir de microbolómetros incluye los componentes mostrados en la Figura 2.10.

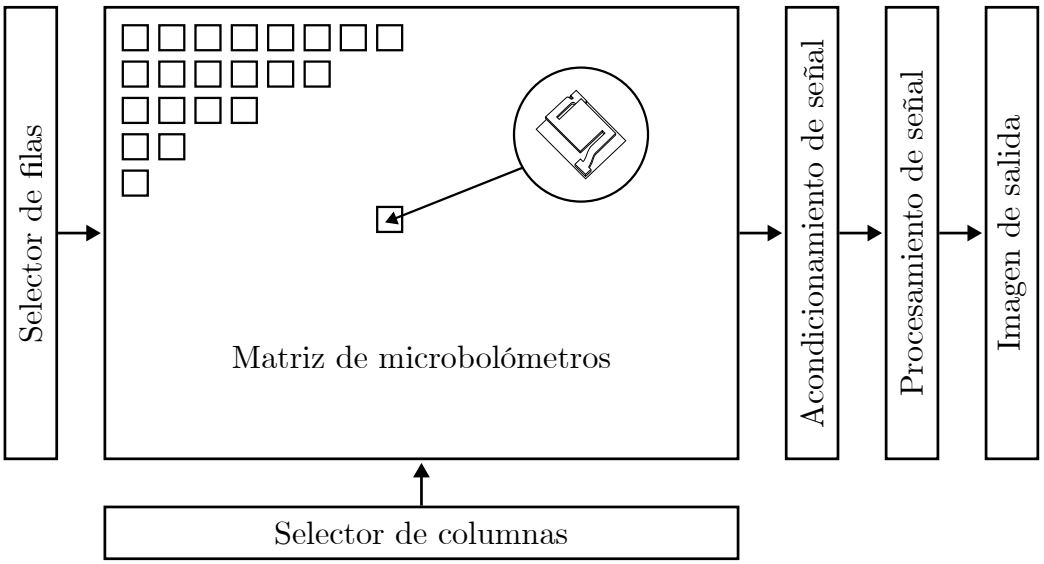


Figura 2.10: Esquema de formación de imágenes a partir de microbolómetros.

El sistema para generar una imagen requiere una matriz de microbolómetros donde cada sensor cambia su resistencia según la potencia infrarroja recibida. Esta señal debe ser procesada por un circuito de lectura. El acceso y organización de cada señal se realiza mediante el direccionamiento de filas y columnas, que permite organizar las señales de los píxeles para formar la imagen final.

El proceso de formación de la imagen se puede resumir en [13], [34]:

1. Polarización de cada microbolómetro empleando un convertidor D/A.
2. Direccionamiento y acceso a cada microbolómetro para acondicionar la señal mediante un circuito de lectura.
3. Lectura de la señal proveniente del circuito de lectura por medio de un convertidor A/D.
4. Recepción y captura de datos mediante software.
5. Interpretación y procesamiento de datos para la generación de una imagen.

Una tarjeta de adquisición de datos es responsable de direccionar cada píxel de la matriz de microbolómetros, controlando tanto los convertidores D/A y A/D. Además, recibe y envía los datos provenientes del circuito de lectura, facilitando su posterior procesamiento para generar la imagen infrarroja. Este componente es esencial para coordinar la transferencia de información y asegurar la correcta formación de la imagen [34].

Capítulo 3

Marco teórico

3.1. Máquina de estados finitos (FSM)

3.1.1. Introducción

Una máquina de estados finitos (FSM, por sus siglas en inglés) se utiliza para modelar un sistema que transita entre un número finito de estados internos, con transiciones que dependen del estado actual y de una entrada externa. A diferencia de un circuito secuencial convencional, las transiciones de estado de una FSM no siguen un patrón simple y repetitivo. La lógica de estado siguiente en una FSM generalmente se construye desde cero y a veces se conoce como lógica aleatoria. Esto contrasta con la lógica de estado siguiente en un circuito secuencial estándar, que suele estar compuesta principalmente por componentes estructurados, como incrementadores y desplazadores. En la práctica, la principal aplicación de una FSM es funcionar como el controlador de un gran sistema digital, examinando los comandos externos y el estado del sistema, y activando las señales de control adecuadas para gestionar la operación de un *data path*, que suele estar compuesta por componentes secuenciales convencionales. Esto se conoce como FSMD (máquina de estados finitos con data path) del cual se tratará en una sección posterior.

3.1.2. Tipos de máquinas de estado

El diagrama de bloques básico de una FSM es similar al de un circuito secuencial regular y se muestra en la Figura 3.1. Consta de un registro de estado, lógica de próximo estado y lógica de salida. Una FSM se denomina *máquina Moore* si la salida depende únicamente del estado, y se denomina *máquina Mealy* si la salida depende tanto del estado como de la entrada externa. En una FSM compleja, pueden coexistir ambos tipos de salida, y en ese caso, se dice que la FSM contiene una salida Moore y una salida

Mealy. Aunque las salidas Moore y Mealy son similares, no son idénticas. Entender sus diferencias sutiles es fundamental para el diseño de controladores.

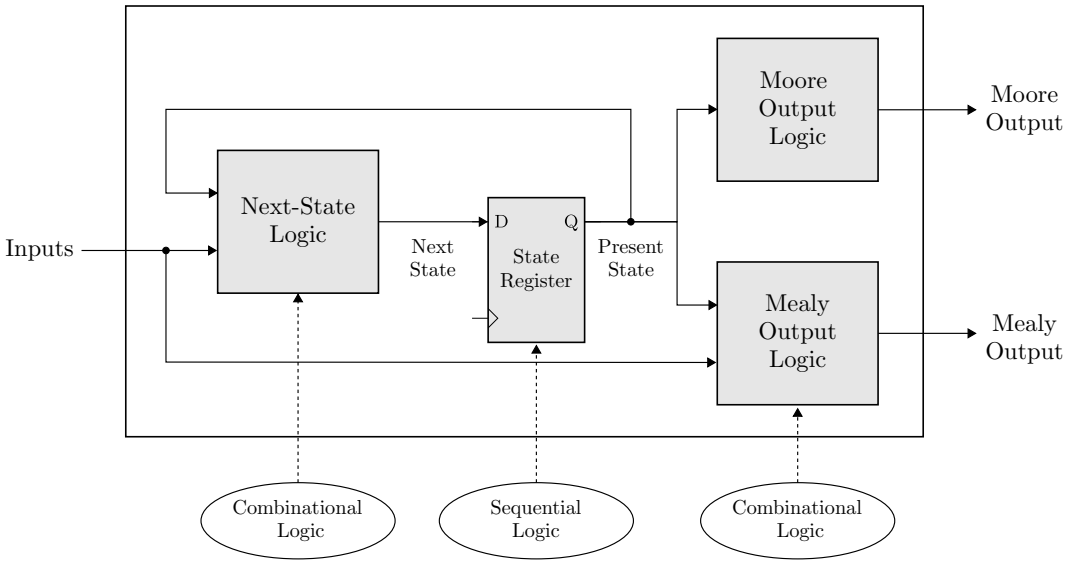


Figura 3.1: Diagrama a bloques de una FSM con salidas Moore y Mealy.

3.1.3. Moore vs Mealy

Las máquinas de estados de tipo Moore son preferidas en la industria porque las salidas disponen de un ciclo completo para estabilizarse a través de la lógica combinacional, lo que facilita el cumplimiento de los tiempos de ciclo requeridos. Por otro lado, las salidas de tipo Mealy permiten que una entrada se procese después de que el ciclo ha comenzado, y dicha entrada debe atravesar la lógica combinacional y cumplir con el tiempo de preparación para la salida Mealy. Si el diseño requiere absolutamente que una entrada llegue al chip después del flanco activo del reloj, pase por la lógica y se refleje en la salida dentro de un solo ciclo, esos diseños suelen emplear una salida Mealy.

Usando un juego de palabras en inglés, se dice que “Moore is Less”, lo que hace referencia a que las máquinas de estados de Moore dependen únicamente del estado actual, mientras que las máquinas de estados de Mealy dependen tanto del estado actual como de una o más entradas.

En general, se evitan los diseños de FSM de Mealy a menos que sean absolutamente necesarios.

3.1.4. Representación de FSM

Una FSM generalmente se describe mediante un diagrama de estados abstracto o un diagrama ASM (diagrama de máquina de estados algorítmica), ambos capturando la entrada, salida, estados y transiciones de la FSM en una representación gráfica. Ambas representaciones contienen la misma información. El diagrama de estados es más compacto y adecuado para aplicaciones simples, mientras que el diagrama ASM, que se asemeja a un diagrama de flujo, es más descriptivo y útil para aplicaciones con condiciones de transición y acciones complejas.

Diagrama de estados: Un diagrama de estados se compone de *nodos*, que representan estados y se dibujan como círculos, y *arcos de transición* anotados. Un solo nodo y sus arcos de transición se muestran en la Figura 3.2(a). Cada arco de transición está asociado con una expresión lógica expresada en términos de señales de entrada, que representa una condición específica. El arco se toma cuando la expresión correspondiente se evalúa como verdadera.

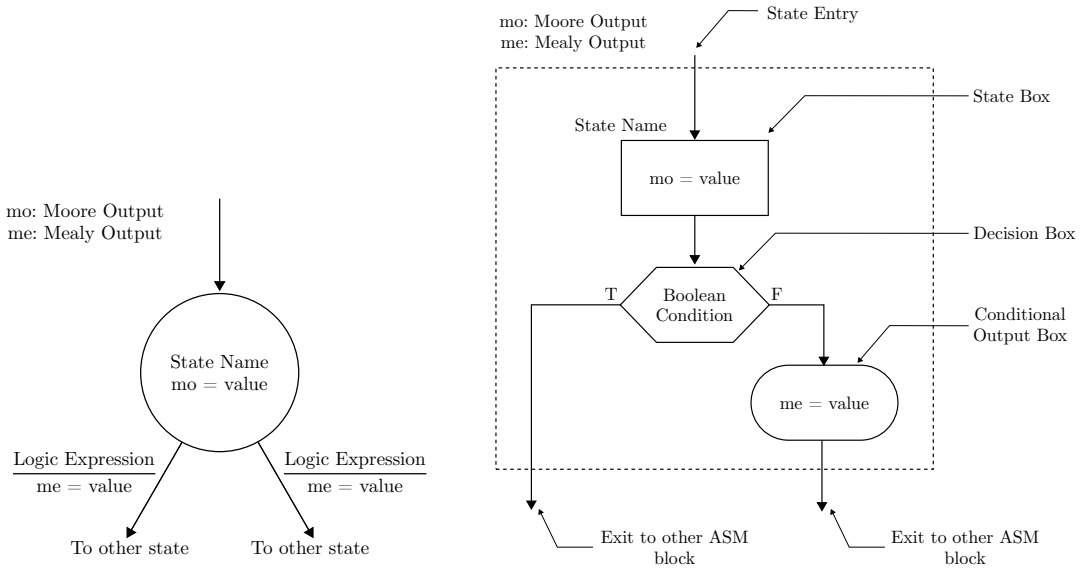
Los valores de salida Moore se ubican dentro del círculo, ya que dependen exclusivamente del estado actual. Los valores de salida Mealy, por su parte, se asocian con las condiciones de los arcos de transición, dado que dependen tanto del estado actual como de la entrada externa. Para mantener el diagrama más claro y ordenado, solo se muestran los valores de salida que están activos. En caso contrario, la señal de salida toma su valor predeterminado (es decir, inactivo).

En la Figura 3.3(a) se muestra un diagrama de estados representativo. La FSM tiene tres estados, dos señales de entrada externas (a y b), una señal de salida Moore (y_1) y una señal de salida Mealy (y_0). La señal y_1 se activa cuando la FSM se encuentra en los estados S_0 o S_1. Por otro lado, la señal y_0 se activa cuando la FSM está en el estado S_0 y las señales a y b están en 1 respectivamente.

Diagrama ASM: Un diagrama ASM está compuesto por una red de bloques ASM. Un *bloque ASM* incluye una *caja de estado* y, de manera opcional, una red de *cajas de decisión* y *cajas de salida condicional*. En la Figura 3.3(b) se muestra un bloque ASM representativo.

Una caja de estado representa un estado en una FSM, y dentro de ella se enumeran los valores de salida de Moore que están activos. Es importante destacar que esta caja tiene solo un camino de salida. Una caja de decisión evalúa la condición de entrada y determina cuál camino de salida seguir. Esta caja tiene dos caminos de salida, etiquetados como T y F, que corresponden a los valores verdadero y falso de la condición. Una caja de salida condicional enumera los valores de salida Mealy que están activos y generalmente se coloca después de una caja de decisión. Esta caja indica que la señal de salida listada solo puede activarse cuando se cumple la condición correspondiente en la caja de decisión.

Un diagrama de estados puede convertirse fácilmente en un diagrama ASM, y viceversa. El diagrama ASM correspondiente al diagrama de estado FSM anterior se muestra en la Figura 3.3(b).



(a) Nodo **(b)** Bloque ASM

Figura 3.2: Simbolo de un estado.

3.1.5. FSM con data path (FSMD)

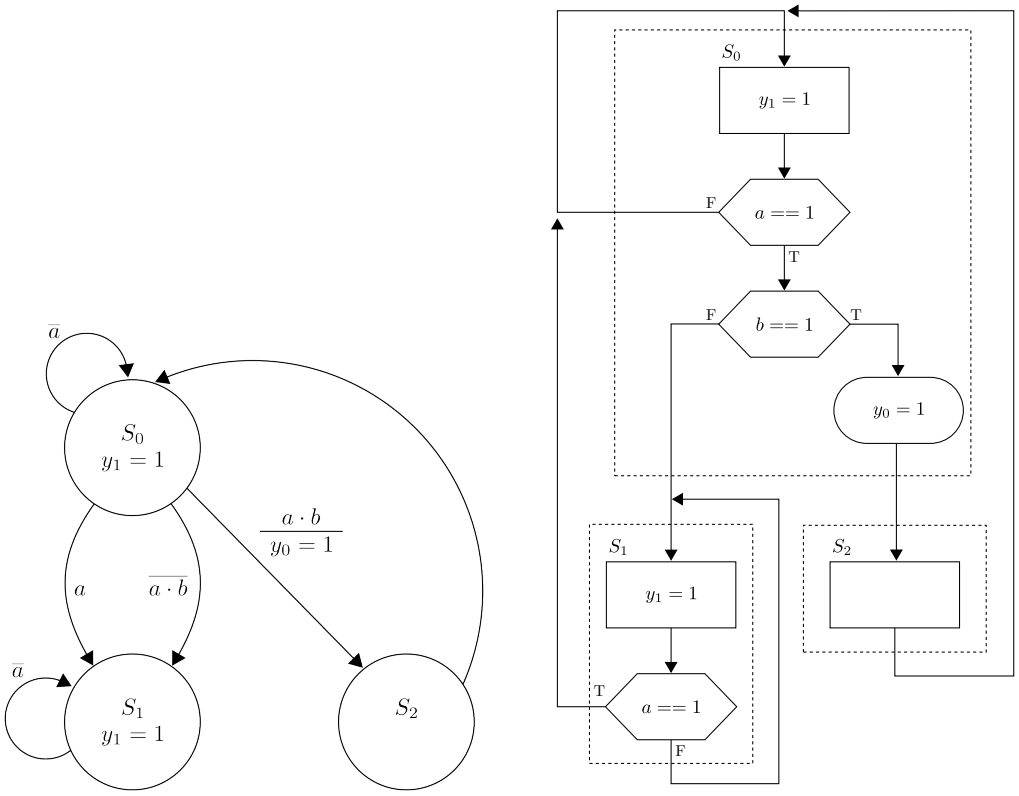
Una FSMD (máquina de estados finitos con data path) combina una FSM y circuitos secuenciales regulares. La FSM, que a veces se conoce como *control path*, examina los comandos externos y el estado, y genera señales de control para especificar la operación de los circuitos secuenciales regulares, que en conjunto se conocen como *data path*. La FSMD se utiliza para implementar sistemas descritos por la *metodología RT* (register transfer), en la cual las operaciones se especifican como manipulación y transferencia de datos entre un conjunto de registros.

3.1.6. Operación RT simple

Una operación RT especifica la manipulación y transferencia de datos para un único registro de destino. Se representa mediante la notación

$$r_{\text{dest}} \leftarrow f(r_{\text{src1}}, r_{\text{src2}}, \ldots, r_{\text{srcn}}) \tag{3.1}$$

donde r_{dest} es el registro de destino, r_{src1}, r_{src2} y r_{srcn} son los registros de origen, y $f(\cdot)$ especifica la operación a realizar. La notación indica que el contenido de los registros de origen se alimenta a la función $f(\cdot)$, que se realiza mediante un circuito combinacional,



(a) Diagrama de estados **(b)** Diagrama ASM

Figura 3.3: Ejemplo de una FSM.

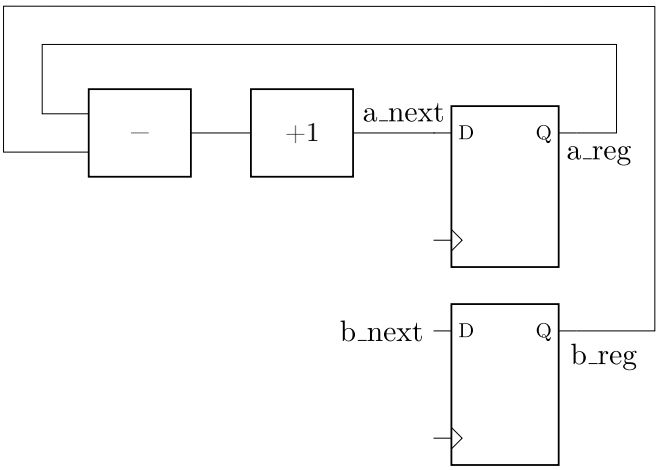
y el resultado se pasa a la entrada del registro de destino y se almacena en el registro de destino en el siguiente flanco ascendente del reloj.

A continuación se presentan varias operaciones RT representativas:

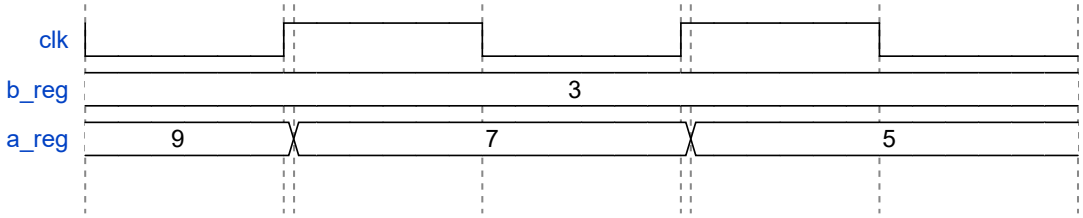
- $r1 \leftarrow 0$. Se almacena una constante 0 en el registro $r1$.
- $r1 \leftarrow r1$. El contenido del registro $r1$ se escribe de nuevo sobre sí mismo.
- $r2 \leftarrow r2\check{}3$. El registro $r2$ se desplaza tres posiciones a la derecha y luego se escribe de nuevo sobre sí mismo.
- $r2 \leftarrow r1$. El contenido del registro $r1$ se transfiere al registro $r2$.
- $i \leftarrow i + 1$. El contenido del registro i se incrementa en 1 y el resultado se escribe sobre sí mismo.
- $d \leftarrow s1 + s2 + s3$. La suma de los registros $s1$, $s2$ y $s3$ se escribe en el registro d.
- $y \leftarrow a \cdot a$. El cuadrado de a se escribe en el registro y.

Una operación RT simple puede implementarse construyendo un circuito combinacional para la función $f(\cdot)$ y conectando la entrada y la salida de los registros. Por

ejemplo, considere la operación $a \leftarrow a - b + 1$. La función $f(\cdot)$ implica un restador y un incrementador. El diagrama de bloques se muestra en la Figura 3.4(a). Para mayor claridad, utilizamos los sufijos "_next" y "_reg" para representar la entrada y la salida de un registro. Nótese que una operación RT está sincronizada por un reloj incorporado. El resultado de la función $f(\cdot)$ no se almacena en el registro de destino hasta el siguiente flanco ascendente del reloj. El diagrama de tiempo de la operación RT anterior se muestra en la Figura 3.4(b).



(a) Diagrama de bloques de ejemplo RT



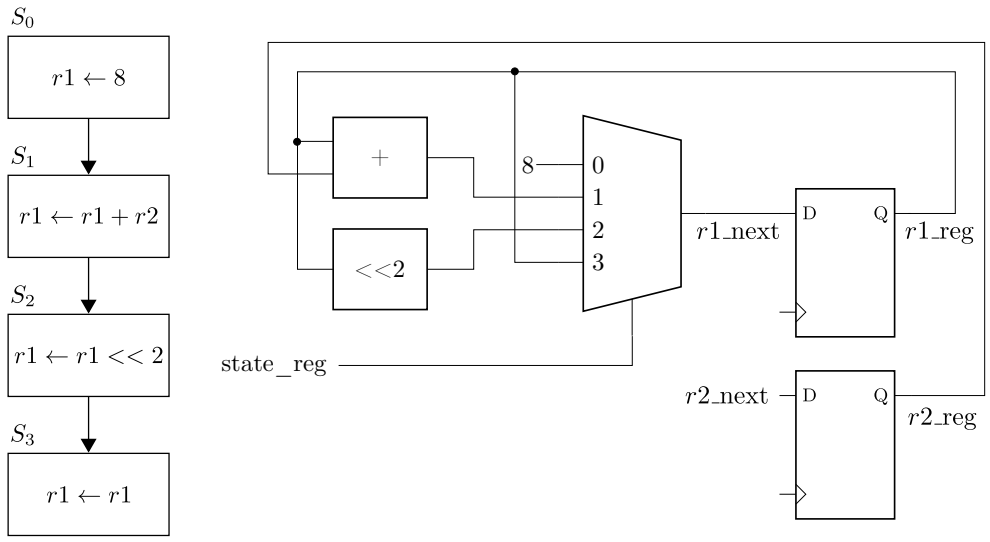
(b) Diagrama de tiempo de ejemplo RT

Figura 3.4: Diagramas de bloques y de tiempo de una operación RT.

3.1.7. Diagrama ASMD

Un circuito basado en la metodología de transferencia de registros (RT) especifica qué operaciones RT deben ejecutarse en cada paso. Dado que una operación RT se realiza en un ciclo de reloj, su temporización es similar a la transición de estado de una FSM. Por ello, una FSM es la opción natural para especificar la secuenciación de un algoritmo RT.

Se extiende el diagrama ASM para incorporar operaciones RT y se denomina *diagrama ASMD* (ASM con data path). Las operaciones RT se tratan como otro tipo de actividad y pueden ubicarse en los mismos lugares donde se utilizan las señales de salida.



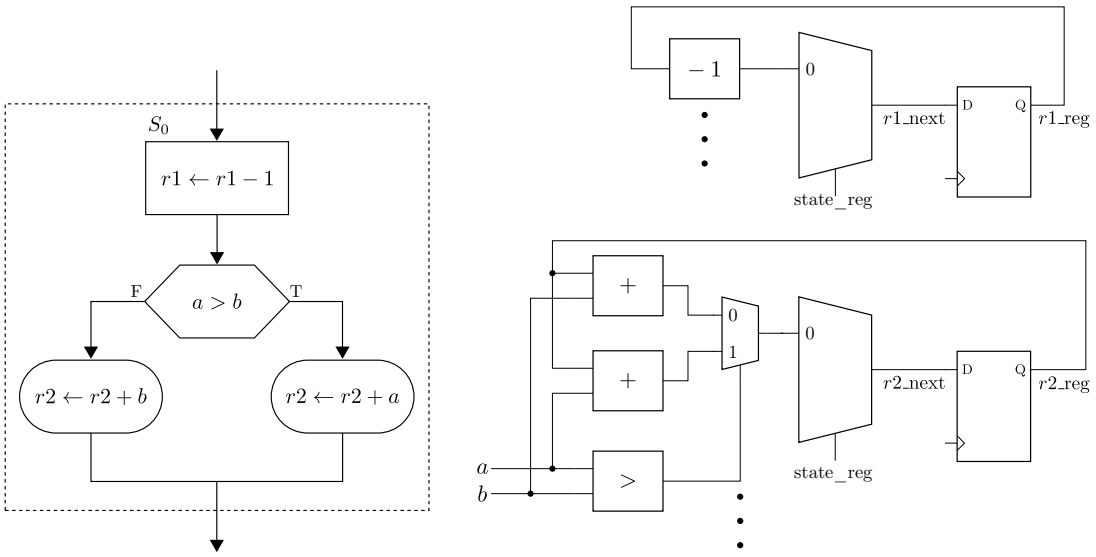
(a) Segmento ASMD

(b) Diagrama de bloques

Figura 3.5: Realización de segmento ASMD.

Un segmento de un diagrama ASMD se muestra en la Figura 3.5(a). Este segmento incluye un registro de destino, $r1$, que se inicializa con el valor 8, se suma con el contenido del registro $r2$, y luego se desplaza dos posiciones a la izquierda. Es importante señalar que el registro $r1$ debe especificarse en cada estado. Cuando $r1$ no se modifica, se debe usar la operación $r1 \leftarrow r1$ para mantener su contenido actual, como en el estado S_3. En algunos diagramas, se asume que $r1 \leftarrow r1$ es la operación RT predeterminada para el registro $r1$ y, por lo tanto, no se incluye en el diagrama ASMD. Implementar las operaciones RT de un diagrama ASMD requiere un circuito de multiplexación para dirigir el siguiente valor deseado al registro de destino. Por ejemplo, el segmento anterior puede implementarse mediante un multiplexor de 4 a 1, como se muestra en la Figura 3.5(b). El estado actual de la FSM (es decir, la salida del registro de estado) controla la señal de selección del multiplexor, seleccionando así el resultado de la operación RT deseada.

Una operación RT también puede especificarse en una caja de salida condicional, como se muestra con el registro $r2$ en la Figura 3.6(a). Dependiendo de la condición $a > b$, la FSMD ejecutará $r2 \leftarrow r2+a$ o $r2 \leftarrow r2+b$. Es importante señalar que todas las operaciones dentro de un bloque ASMD se realizan en paralelo. Para implementar esto, se necesitan realizar las operaciones $a > b$, $r2+a$ y $r2+b$, y utilizar un multiplexor para dirigir el valor deseado hacia $r2$. El diagrama de bloques correspondiente se muestra en la Figura 3.6(b).



(a) Diagrama de bloques **(b)** Bloque ASM

Figura 3.6: Realización de una operación RT en una caja de salida condicional.

3.1.8. Caja de decisión con registro

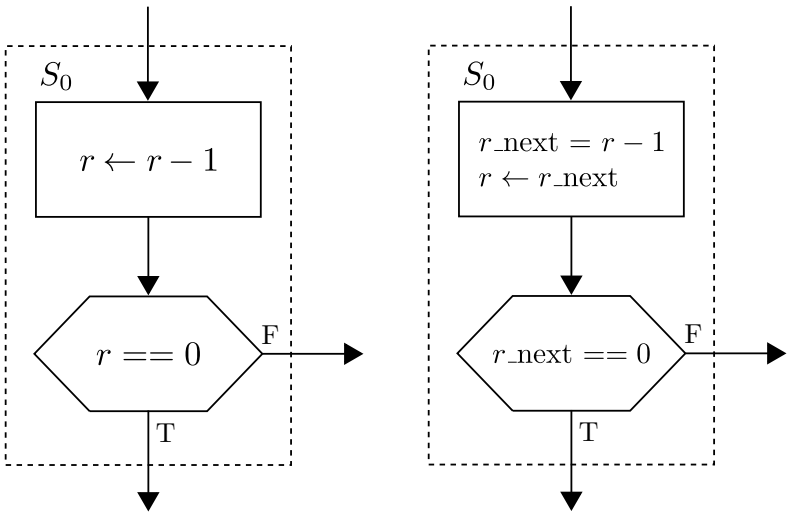
La apariencia de un diagrama ASMD es similar a la de un diagrama de flujo convencional. La principal diferencia radica en que la operación RT en un diagrama ASMD está controlada por una señal de reloj interna, y el registro de destino se actualiza cuando el FSMD sale del bloque ASMD actual, pero no dentro del bloque. La operación $r + r - 1$ en realidad significa que:

- `r_next = r_reg - 1`
- `r_reg = r_next` en el flanco ascendente del reloj (es decir, cuando el FSMD sale del bloque actual).

Este almacenamiento retrasado puede introducir errores sutiles cuando se utiliza un registro en una caja de decisión. Considere el segmento FSMD en la Figura 3.7(a). El registro r se decrementa en la caja de estado y se utiliza en la caja de decisión. Dado que el registro r no se actualiza hasta que el FSMD sale del bloque, el contenido antiguo de r se utiliza para la comparación en la caja de decisión. Si se desea utilizar el nuevo valor de r, se debe usar la salida de la lógica combinacional (es decir, $r_$next) en la caja de decisión (es decir, reemplazar la expresión $r == 0$ con $r_$next $== 0$), como se muestra en la Figura 3.7(b).

3.1.9. Diagrama de bloques de una FSMD

El diagrama de bloques conceptual de una FSMD se divide en *data path* y en *control path*, como se muestra en la Figura 3.8. El *data path* se encarga de ejecutar las



(a) Utilizar el valor antiguo de r **(b)** Utilizar el nuevo valor de r

Figura 3.7: Bloque ASM afectado por un almacenamiento retrasado.

operaciones RT necesarias. Consiste en:

- *Data registers*: almacenan los resultados intermedios de los cálculos.
- *Functional units*: realizan las funciones especificadas por las operaciones RT.
- *Routing network*: dirige los datos entre los registros de almacenamiento y las unidades funcionales.

El *data path* sigue la **señal de control** (Control Signal) para llevar a cabo las operaciones RT deseadas y genera la señal de **estado interno** (Internal Status).

El *control path* es una FSM. Al igual que cualquier FSM convencional, contiene un registro de estado, lógica de próximo estado y lógica de salida. Utiliza la señal de **comando** (Command) externa y la señal de **estado interno** (Internal Status) del *data path* como entradas, y genera la **señal de control** (Control Signal) que regula la operación del *data path*. Además, la FSM genera una señal de **estado externa** (External Status) para indicar el estado de la operación del FSMD.

Es importante destacar que, aunque un FSMD está compuesta por dos tipos de circuitos secuenciales, ambos circuitos son controlados por el mismo reloj, lo que mantiene al FSMD como un sistema síncrono.

3.2. Visión general de la comunicación serial

Con frecuencia, un sistema necesita comunicarse con otro que no reside en el mismo dispositivo. Para reducir el número de pines de I/O y el cableado externo, los dos

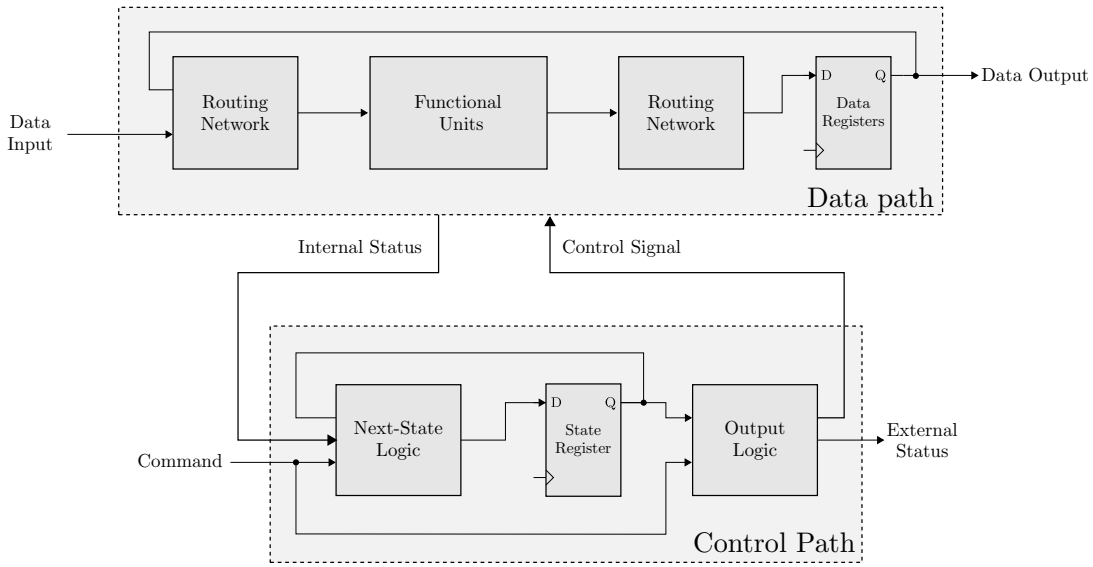


Figura 3.8: Diagrama de bloques de una FSMD.

sistemas pueden transferir datos a través de una única línea serial, bit a bit. El sistema transmisor realiza la conversión de paralelo a serie y, a continuación, envía los datos serie a través de una única línea. El sistema receptor realiza la conversión serie a paralelo y restaura los datos paralelos originales.

La comunicación serial puede utilizarse tanto para transferir datos a alta velocidad como a baja velocidad. En una interfaz de alta velocidad, como USB y gigabit Ethernet, la velocidad de transmisión de datos puede alcanzar varios cientos de millones de bits por segundo o más.

En una interfaz de baja velocidad, la velocidad de transmisión de datos oscila entre varios miles y varios cientos de miles de bits por segundo. Es adecuada para la mayoría de los periféricos de I/O generales y para tareas de adquisición y control de datos. Dado que la velocidad de datos es mucho más lenta que la velocidad de reloj de una FPGA, estos esquemas pueden ser realizados por los elementos lógicos genéricos de una FPGA. A continuación se discuten los esquemas UART y SPI.

3.3. Controlador UART

La comunicación serial utiliza una única línea de datos para intercambiar información entre dos sistemas. El sistema transmisor convierte los datos paralelos en un flujo serie y el sistema receptor vuelve a ensamblar los datos serie en su formato paralelo original. El esquema más utilizado es el UART, (Universal Asynchronous Receiver and Transmitter) o receptor y transmisor asíncrono universal en español.

3.3.1. Visión general

Un controlador UART básico incluye un transmisor y un receptor. El transmisor es un registro de corrimiento especial que carga datos en paralelo y luego los desplaza bit a bit a una velocidad específica. El receptor, por su parte, desplaza los datos bit a bit y los vuelve a ensamblar. La línea serial es 1 cuando está inactiva. La transmisión comienza con un bit de inicio, que es 0, seguido de bits de datos y un bit de paridad opcional, y termina con bits de parada, que son 1. El número de bits de datos puede ser 6, 7 u 8. El bit de paridad opcional se utiliza para la detección de errores. Para paridad impar, se pone a 0 cuando los bits de datos tienen un número impar de 1's. Para paridad par, se pone a 0 cuando los bits de datos tienen un número par de 1's. El número de bits de parada puede ser 1, 1,5 ó 2. En la Figura 3.9 se muestra la transmisión con ocho bits de datos, sin paridad y un bit de parada. Observe que el LSB de la palabra de datos se transmite primero.

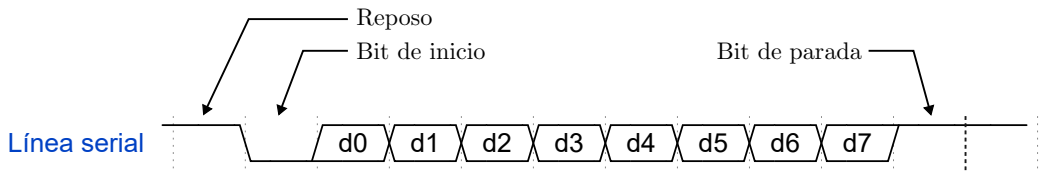


Figura 3.9: Diagrama de la transmisión de un byte serial con UART.

A través de la línea serial no se transmite información de reloj. Antes de iniciar la transmisión, el transmisor y el receptor deben acordar de antemano una serie de parámetros, que incluyen la velocidad en baudios (es decir, el número de bits por segundo), el número de bits de datos y bits de parada, y el uso del bit de paridad. Con los parámetros predeterminados, el receptor utiliza un esquema de sobremuestreo para recuperar los bits de datos. Las velocidades en baudios más utilizadas son 9,600 y 19,200 baudios.

3.3.2. Procedimiento de sobremuestreo

La velocidad de muestreo más utilizada es 16 veces la velocidad en baudios, lo que significa que cada bit serie se muestrea 16 veces. Para un enlace de comunicación con N bits de datos y M bits de parada, el esquema de sobremuestreo funciona de la siguiente manera:

1. Espera hasta que la señal entrante se convierta en 0, el comienzo del bit de inicio, y luego comienza a contar los pulsos de muestreo.

2. Cuando el contador llegue a 7, la señal entrante alcanzará el punto medio del bit de inicio. Limpia el contador a 0 y reinícialo.

3. Cuando el contador llegue a 15, la señal entrante habrá avanzado un bit y alcanzará el punto medio del primer bit de datos. Obtén su valor, desplázalo a un registro y reinicia el contador.

4. Repite el paso 3 N–1 veces más para obtener los bits de datos restantes.

5. Si se utiliza el bit de paridad opcional, repite el paso 3 una vez para obtener el bit de paridad.

6. Repite el paso 3 M veces más para obtener los bits de parada.

El esquema de sobremuestreo realiza básicamente la función de una señal de reloj. En lugar de utilizar el flanco ascendente para indicar cuándo es válida la señal de entrada, utiliza los pulsos de muestreo para estimar el punto medio de cada bit. Aunque el receptor no tiene información sobre el momento exacto de inicio del bit, la estimación puede estar desviada como máximo en 1/16. Las siguientes recuperaciones de bits de datos también se desvían como máximo de 1/16 del punto medio. Debido al sobremuestreo, la velocidad en baudios sólo puede ser una pequeña fracción de la velocidad de reloj del sistema, por lo que este esquema no es apropiado para una velocidad de datos alta.

3.3.3. Diseño conceptual

El diagrama de alto nivel de un sistema UART se muestra en la Figura 3.10. Consiste en tres componentes principales. El generador de la tasa de baudios genera una señal de sobremuestreo. El receptor realiza la conversión de serie a paralelo y el transmisor realiza la conversión de paralelo a serie. El código completo integrando los tres componentes se muestra en el Código A.8.

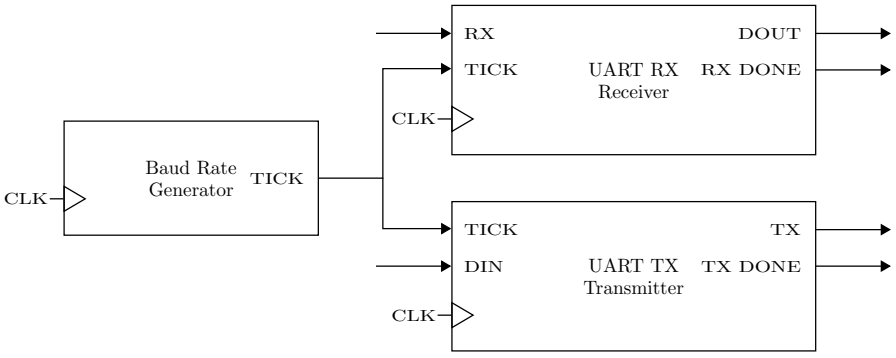


Figura 3.10: Diagrama de bloques de una controlador UART completo.

3.3.4. Generador de la tasa de baudios

El generador de la tasa de baudios genera una señal de muestreo cuya frecuencia es exactamente 16 veces la tasa de baudios designada para el UART. Para evitar la creación de un nuevo dominio de reloj y violar el principio de diseño síncrono, la señal de muestreo debe funcionar como pulsos de habilitación en lugar de la señal de reloj para el receptor UART.

El generador de la tasa de baudios es un contador programable y el código HDL se muestra en el Código A.5. El código es un contador mod$-m$ parametrizado que se reinicia cada vez que el contador llega a un valor determinado. Por ejemplo para una tasa de 19,200 baudios, la tasa de muestreo tiene que ser de 307,200 (es decir, 19,200 $\times$ 16) pulsos por segundo. Dada una frecuencia del reloj de sistema de 50 MHz, el generador de la tasa de baudios necesita un contador mod-163, $\left(\frac{50\times10^6}{19200\times16}\right)$, en el cual se genera un pulso de ciclo de reloj cada 163 ciclos de reloj del sistema.

3.3.5. Receptor UART

El receptor UART obtiene el byte de datos de la línea serial mediante sobremuestreo. Utiliza la señal del generador de la tasa de baudios para estimar los puntos medios de los bits transmitidos y, a continuación, los recupera en esos puntos de manera precisa.

La operación general de recepción se puede describir mediante un diagrama ASMD, como se muestra en la Figura 3.11. Para permitir futuras modificaciones, se utilizan dos parámetros en la descripción. El parámetro D_BIT indica el número de bits de datos, mientras que SB_TICK representa el número de "ticks"necesarios para los bits de parada, que son 16, 24 y 32 para 1, 1.5 y 2 bits de parada, respectivamente. En este diseño, a D_BIT y SB_TICK se les asignaron los valores 8 y 16.

El diagrama sigue los pasos explicados en la Sección 3.3.2 e incluye tres estados principales: **START**, **DATA** y **STOP**, que representan el procesamiento del bit de inicio, los bits de datos y el bit de parada. La señal s_tick es el tick de habilitación proveniente del generador de tasa de baudios, con 16 ticks en cada intervalo de bits. Es importante señalar que el FSMD permanece en el mismo estado a menos que se active la señal s_tick. Existen dos contadores, representados por los registros s y n. El registro s lleva la cuenta de los ticks de muestreo, contando hasta 7 en el estado de START, hasta 15 en el estado de DATA, y hasta SB_TICK en el estado de STOP. El registro n cuenta el número de bits de datos recibidos en el estado de DATA. Los bits recuperados se desplazan y se reensamblan en el registro b. Se incluye una señal de estado, rx_done_tick, que se activa durante un ciclo de reloj al finalizar el proceso de recepción. El código correspondiente se encuentra en el Código A.6.

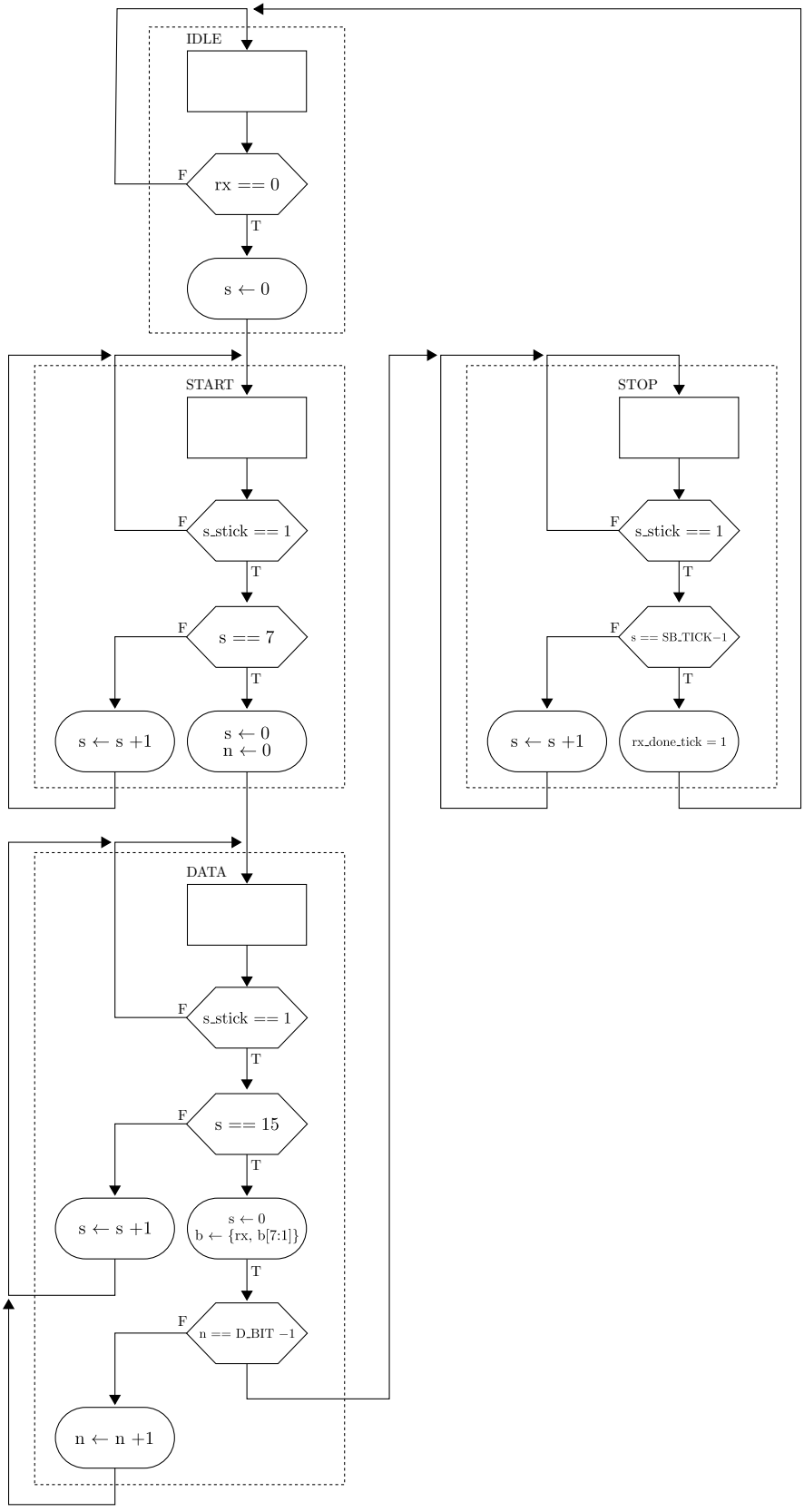


Figura 3.11: Diagrama ASMD de receptor UART.

3.3.6. Transmisor UART

La estructura de un subsistema de transmisión UART es similar a la del subsistema de recepción. Está compuesto por un transmisor UART y un generador de tasa de baudios.

El transmisor UART es, en esencia, un registro de desplazamiento que envía los bits de datos a una velocidad específica. Esta velocidad puede ser controlada mediante ticks de habilitación de un solo ciclo de reloj generados por el generador de tasa de baudios. Como no se utiliza sobremuestreo, la frecuencia de los ticks es 16 veces más lenta que la del receptor UART. En lugar de introducir un nuevo contador, el transmisor UART suele compartir el generador de tasa de baudios del receptor y utiliza un contador interno para llevar el control de los ticks de habilitación. Se desplaza un bit cada 16 ticks de habilitación.

El diagrama ASMD del transmisor UART es similar al del receptor UART. Tras la activación de la señal tx_start, el FSMD carga la palabra de datos y avanza progresivamente por los estados de **START**, **DATA** y **STOP** para enviar los bits correspondientes. Al completar el proceso, se indica mediante la señal tx_done_tick, que se activa durante un ciclo de reloj. Se emplea un búfer de 1 bit, tx_reg, para filtrar posibles fallos o errores. El código correspondiente se encuentra en el Código A.7.

3.4. Controlador SPI

El estándar SPI (Serial Peripheral Interface) o Interfaz Periférica Serial en español, es un protocolo de transferencia de datos en serie desarrollado originalmente por Motorola a principios de los 80. El bus SPI se compone de tres líneas, dos de ellas para transmitir y recibir datos en serie, y una para la señal de reloj. Se pueden conectar al bus un dispositivo maestro y varios dispositivos esclavos. El maestro genera la señal de reloj e inicia la transferencia de datos. El estándar SPI se utiliza ampliamente en sistemas embebidos para conectar módulos periféricos. [36], [37]

3.4.1. Visión general

El estándar SPI especifica el protocolo que intercambia datos entre dos dispositivos a través de líneas seriales. En lugar de utilizar el esquema de sobremuestreo de UART, la interfaz SPI incluye una tercera línea para controlar el desplazamiento y el muestreo de los datos seriales. Las actividades se realizan en el flanco de transición de esta señal. Su función es similar a la señal de reloj de un sistema síncrono, por lo que esta línea se denomina reloj SPI.

A diferencia de la configuración UART, en la que dos sistemas son simétricos y am-

bos pueden iniciar una transmisión, el estándar SPI utiliza una configuración maestro-esclavo. El maestro controla el funcionamiento general y genera la señal de reloj SPI. Sólo el maestro puede iniciar una transferencia de datos.

A pesar de su nombre, el reloj SPI no es un reloj de sistema real y no debe utilizarse para controlar ningún registro directamente. La velocidad del reloj del sistema del controlador SPI es mucho más rápida que la velocidad del reloj SPI. Desde el punto de vista del controlador SPI, el reloj SPI es sólo otra señal de control.

3.4.2. Arquitectura del controlador

El diagrama conceptual de un bus SPI con dos dispositivos se muestra en la Figura 3.12. Tanto el maestro como el esclavo tienen un registro de desplazamiento (shift register) en su interior. Los dos registros de desplazamiento están conectados como un anillo a través de las líneas MOSI (para Master-Out-Slave-In) y MISO (para Master-In-Slave-Out) y su funcionamiento está coordinado por la misma señal de reloj SPI, SCLK.

Las señales MOSI y MISO son algo así como la señal de transmisión (TX) y la señal de recepción (RX) del UART. Suponemos que ambos registros tienen ocho bits de ancho y que la transferencia de datos se realiza byte a byte. Al principio de la operación, tanto el maestro como el esclavo cargan datos en los registros. Durante la transferencia de datos, los datos de ambos registros se desplazan un bit a la derecha en cada ciclo SCLK. Tras ocho ciclos SCLK, se han desplazado ocho bits de datos y el maestro y el esclavo han intercambiado los valores de los registros. A continuación, el maestro y el esclavo pueden procesar los datos recibidos. Esta operación puede interpretarse como que el maestro escribe datos en el esclavo y los lee simultáneamente, lo que se conoce como operación full-duplex.

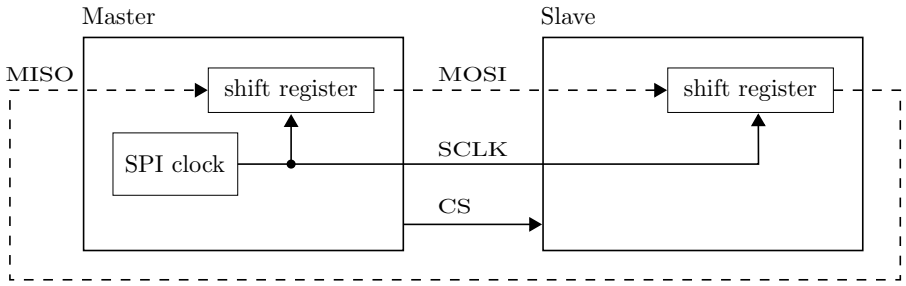


Figura 3.12: Diagrama conceptual del bus SPI.

Además de las líneas MOSI, MISO y SCLK, un dispositivo esclavo también puede tener una entrada de selección de chip activa en bajo, SS (para Slave Select). En otros casos también puede tener el nombre de CS (para Chip-Select). Se puede utilizar para que el maestro seleccione el dispositivo esclavo deseado si hay varios dispositivos esclavos

en el bus. Muchos dispositivos SPI también utilizan CS para ciertas funcionalidades de control y no se puede omitir, incluso en una configuración de un solo esclavo.

3.4.3. Configuración de múltiples dispositivos

El estándar SPI admite una configuración múltiple-esclavo, en la que un dispositivo maestro puede controlar más de un dispositivo esclavo. Existen dos esquemas básicos, que son la configuración en paralelo y la configuración en cadena.

La configuración en paralelo utiliza una línea CS dedicada para cada dispositivo esclavo, como se muestra en la Figura 3.13. Una línea CS funciona como señal de selección de chip y el maestro puede seleccionar el dispositivo deseado activando la línea correspondiente. Esta configuración puede acomodar un maestro y dispositivos esclavos independientes. Dado que las líneas MISO de los esclavos están unidas, la línea MISO debe ser controlada por un buffer de tres estados y su salida debe estar en un estado de alta impedancia cuando el dispositivo esclavo no está seleccionado.

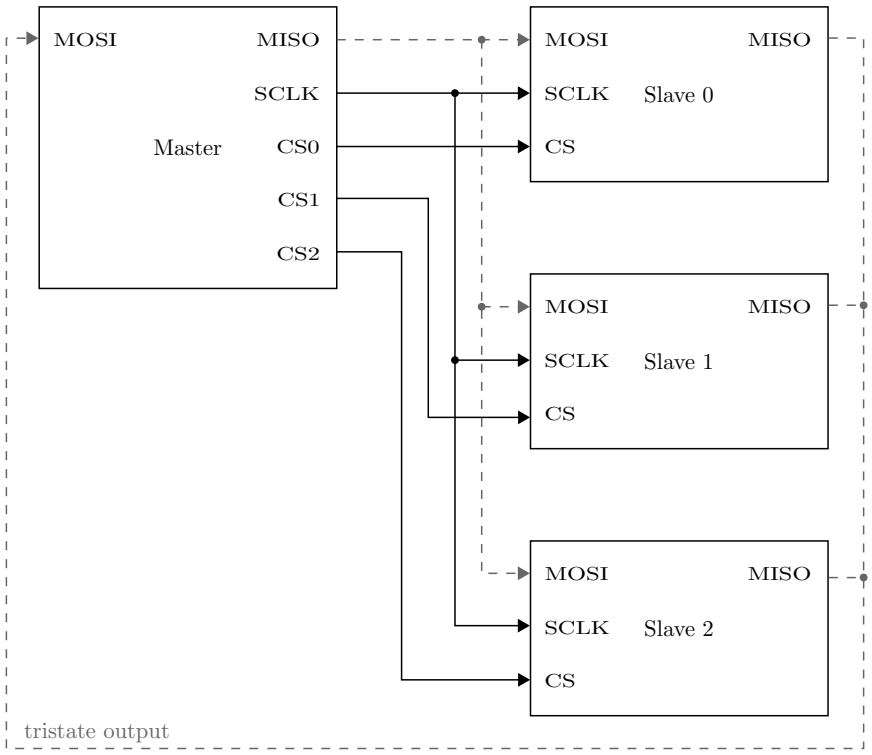


Figura 3.13: Configuración en paralelo de bus SPI.

La configuración en cadena conecta las líneas MOSI y MISO en una cadena en cascada, como se muestra en la Figura 3.14. Se utiliza una única línea CS para controlar todos los dispositivos esclavos. Conceptualmente, la cadena forma un gran registro de desplazamiento y los datos se transfieren en serie de dispositivo a dispositivo. Los

dispositivos de esta configuración deben ser cooperativos y seguir el mismo protocolo para transmitir, insertar y extraer bytes de datos.

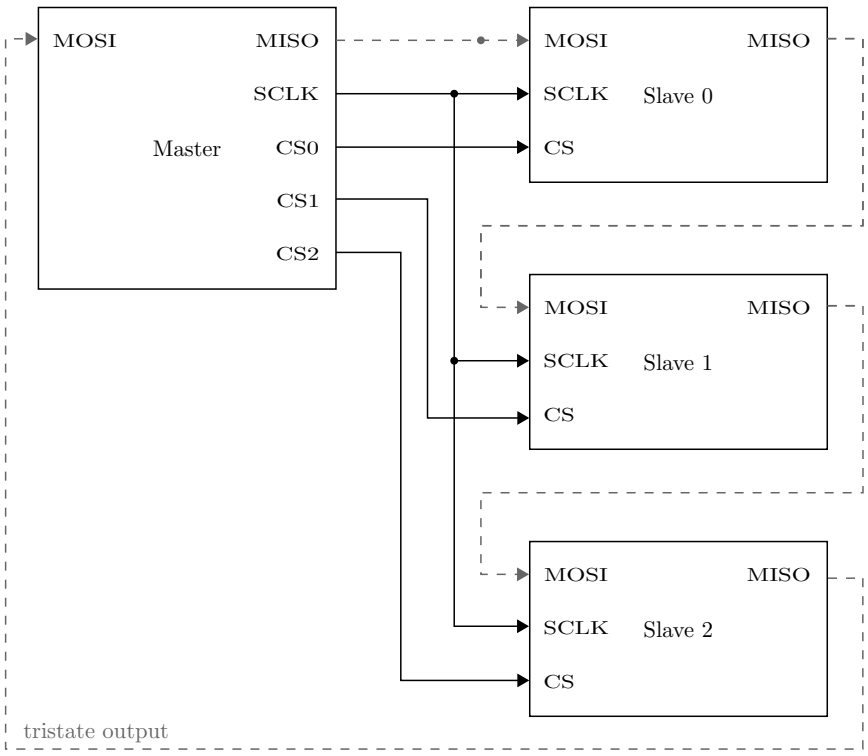


Figura 3.14: Configuración en cadena de bus SPI.

3.4.4. Sincronización del controlador

El bus SPI utiliza los flancos del reloj SPI (SCLK) para controlar y sincronizar la transferencia de bits de los datos. Para aclarar la explicación, se definen dos actividades durante una transferencia de bits: enviar (es decir, desplazar) un nuevo bit a la línea de datos y muestrear (es decir, capturar) un bit de la línea de datos. El envío y el muestreo se completan en el mismo ciclo de reloj SPI, pero ocurren en flancos de reloj opuestos. En la Figura 3.15 se muestra un diagrama de tiempos representativo. Inicialmente, el bus está inactivo y la línea SCLK está en 0. En t_0, el maestro activa CS y el esclavo designado coloca el primer bit de datos (bit 7) en la línea MISO. En t_1, el maestro inicia el reloj SPI y envía el bit b7 en la línea MOSI. Dado que la primera mitad del periodo del reloj SPI es 0, el valor en SCLK permanece sin cambios. El tiempo que transcurre entre t_0 y t_1 puede ser muy pequeño o incluso cero. En t_2, el maestro sube el reloj SPI y avanza a la segunda mitad del periodo del reloj. En el flanco de transición de 0 a 1, el maestro muestrea los datos en MISO y el esclavo los datos en MOSI. En t_3, se completa el primer ciclo de reloj SPI. El maestro comienza el segundo periodo de reloj SPI y baja SCLK. Tanto el maestro como el esclavo envían nuevos bits a la línea de datos.

En t_4, el maestro vuelve a subir SCLK y tanto el maestro como el esclavo muestrean los nuevos bits de datos. Las actividades de transmisión y muestreo se repiten hasta que se transfieren ocho bits de datos. Los últimos datos se muestrean en t_5 y SCLK vuelve a 0 en t_6. En t_7, el maestro desactiva CS. Cabe destacar que todos los muestreos se realizan en los flancos ascendentes de SCLK y todas las transmisiones (excepto la inicial) se realizan en los flancos descendentes de SCLK.

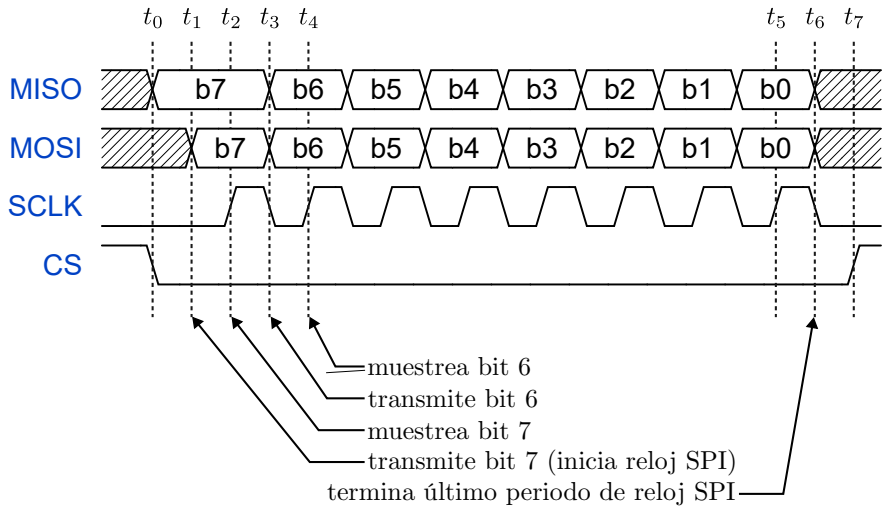


Figura 3.15: Diagrama de tiempos representativo de una transferencia de datos SPI.

3.4.5. Modos de operación

El modo de operación del SPI define las relaciones entre los flancos del reloj SPI y las actividades de envío y muestreo en las líneas de datos. Hay cuatro modos. Los modos dependen de dos parámetros, que son la polaridad del reloj (abreviado como CPOL, Clock Polarity) y la fase del reloj (abreviado como CPHA, Clock Phase). La polaridad del reloj se define como el valor de SCLK cuando está en reposo, que puede ser 0 o 1. La fase del reloj es más difícil de definir. Una interpretación es si un flanco del reloj se utiliza para enviar el primer bit de datos. Si CPHA es 1, el maestro conduce el bit en el primer flanco de transición. Si CPHA es 0, el maestro envía el bit en el flanco de transición cero, inmediatamente cuando CS se activa (lo que significa que no hay flanco o no en el primer flanco).

Basándose en los dos parámetros, los modos de operación del SPI se definen como sigue:

Observe que el diagrama de tiempos de la Figura 3.15 corresponde al modo 0, ya que SCLK es 0 cuando está inactivo y el primer bit no es transmitido por el primer flanco de transición.

El diagrama de tiempo de los cuatro modos se muestra en la Figura 3.16. El modo

Tabla 3.1: Modos de operación del SPI.

Modos SPI	Clock polarity (CPOL)	Clock phase (CPHA)	Los datos de desplazan en	Los datos se muestrean en
0	0	0	bajada SCLK, y cuando CS se activa	subida SCLK
1	0	1	subida SCLK	bajada SCLK
2	1	0	subida SCLK, y cuando CS se activa	bajada SCLK
3	1	1	bajada SCLK	subida SCLK

0 es el modo más comúnmente utilizado. En este modo, el valor en reposo es 0 y el ciclo del reloj comienza con 0. Dado que el valor en reposo y el valor inicial del reloj son los mismos, el primer bit de datos se transmite antes del primer flanco de transición. En el modo 1, el valor en reposo también es 0, pero el ciclo del reloj comienza con 1. El valor inicial de 1 lleva a una transición de 0 a 1, por lo tanto, el primer bit se transmite en el primer flanco. Es importante notar que en estos dos modos, el período del reloj y el tiempo de inicio son los mismos, pero sus valores están fuera de fase.

El valor en reposo de SCLK en los modos 2 y 3 es 1. La forma de onda de SCLK en el modo 2 es exactamente opuesta a la del modo 0, y la forma de onda en el modo 3 es exactamente opuesta a la del modo 1. Nuevamente, es importante destacar que el período del reloj y el tiempo de inicio son los mismos para todos los modos.

3.4.6. Aspectos indefinidos

La interfaz SPI fue desarrollada por Motorola y se ha convertido en un estándar de facto. No existe un organismo rector u organización que supervise este estándar. Varios aspectos importantes no están definidos en el estándar.

El primer aspecto es el uso de la señal CS. La señal CS actúa principalmente como una señal de habilitación o selección de chip. Un dispositivo esclavo se desactiva si su señal CS no está activa. En muchos dispositivos, la señal CS también funciona como una señal de control. El intercambio de datos se realiza transacción por transacción:

- El maestro activa CS.
- El maestro y el esclavo seleccionado transfieren bits de datos.
- El maestro desactiva CS.

Una transacción se muestra en la Figura 3.17. Las transiciones causadas por la activación y desactivación de CS se utilizan para activar ciertas acciones, como enviar un bit o capturar datos en paralelo, en el dispositivo esclavo. Esto implica que CS debe estar conectado al maestro, incluso si solo hay un dispositivo esclavo; es decir, simplemente conectarlo a 0 no funcionará. El estándar SPI no define explícitamente el papel de la señal CS ni el protocolo en la transacción. Además, los requisitos de

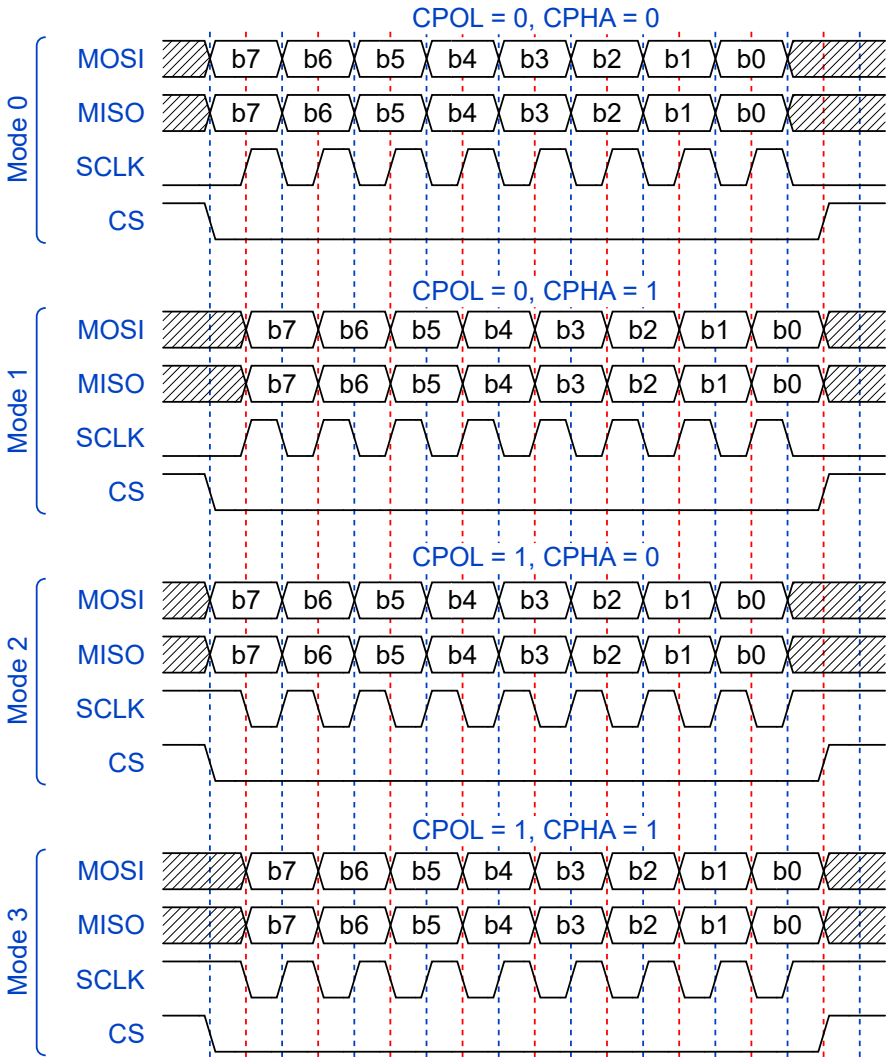


Figura 3.16: Diagrama de tiempos de los diferentes modos SPI.

temporización como el tiempo de preparación de CS (t_{SS_SETUP}), que es el intervalo entre la activación de CS y el inicio del reloj, el tiempo de retención de CS (t_{SS_HOLD}), que es el intervalo entre la desactivación de CS y la terminación del reloj, y el tiempo de cambio entre dos transacciones (t_{SS_TURN}) no esta especificado.

El segundo aspecto indefinido es el número de bits en un intercambio de datos. En la Figura 3.15 se transfieren ocho bits. Sin embargo, el estándar SPI no especifica el número de bits transferidos en una transacción.

Finalmente, el estándar SPI no especifica el orden de bits de la transmisión, es decir, si se transfiere primero el bit mas significactivo (MSB) o el bit menos significativo (LSB) de un byte de datos o de una palabra de datos. El MSB se utiliza comúnmente, pero no está garantizado.

Debido a estos aspectos indefinidos, debemos consultar la hoja de datos del dispositivo y adaptar el acceso para cada dispositivo. Esto suele hacerlo el controlador de

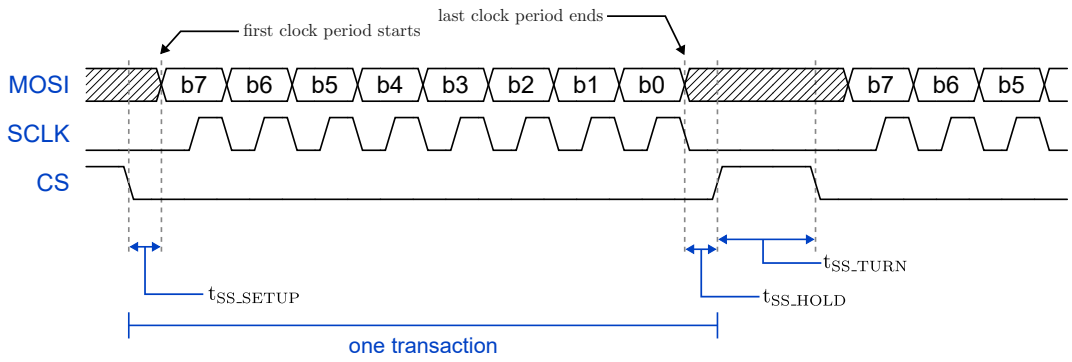


Figura 3.17: Diagrama de tiempos con aspectos indefinidos, SPI modo 0.

software y el programa de aplicación.

3.5. Convertidores DAC y ADC

3.5.1. Visión general

Los convertidores analógico-digitales (ADC) traducen las magnitudes analógicas, características de la mayoría de los fenómenos del mundo real, a lenguaje digital, utilizado en el procesamiento de la información, la informática, la transmisión de datos y los sistemas de control. Los convertidores de digital-analógico (DAC) se utilizan para transformar los datos transmitidos o almacenados, o los resultados del procesamiento digital, de nuevo en variables del mundo real para su control, visualización de información o procesamiento analógico posterior [38].

Las variables de entrada analógicas, independientemente de su su origen, suelen ser convertidas mediante transductores en voltajes o corrientes. Estas magnitudes eléctricas pueden manifestarse de las siguientes formas: (1) como mediciones directas y continuas de corriente continua (DC) que pueden ser rápidas o lentas, representando un fenómeno en el dominio del tiempo, (2) como formas de onda de corriente alterna (AC) moduladas mediante diversas técnicas de modulación, (3) o en una combinación de ambas. Ejemplos del primer tipo incluyen las salidas de termopares, potenciómetros en referencias de corriente continua (DC); del segundo tipo, mediciones ópticas moduladas, puentes de corriente alterna (AC), y señales digitales sumergidas en ruido.

Las variables analógicas a tratar son aquellas que involucran voltajes o corrientes que representan fenómenos analógicos reales. Estas pueden ser de banda ancha o banda estrecha, y pueden ser escaladas a partir de la medición directa o sometidas a algún tipo de preprocesamiento analógico, como la linealización, combinación, demodulación, filtrado, retención de muestras, entre otros.

Como parte del proceso, los voltajes y corrientes se normalizan a rangos compatibles

con los rangos de entrada asignados al ADC. Las salidas analógicas de voltaje o corriente de los DACs se generan de forma directa y en formato normalizado, aunque pueden ser procesadas posteriormente (por ejemplo, escaladas, filtradas, amplificadas, etc.)

La información en forma digital se representa normalmente mediante niveles de voltaje arbitrariamente fijos referidos a tierra, ya sea en las salidas de las compuertas lógicas o aplicados a sus entradas. Los números digitales utilizados son básicamente binarios; es decir, cada bit o unidad de información tiene uno de dos posibles estados. Estos estados son apagado, falso, o 0, y encendido, verdadero, o 1. También es posible representar los dos estados lógicos mediante dos niveles diferentes de corriente; sin embargo, esto es mucho menos común que el uso de voltajes. Tampoco es necesario que los voltajes estén referidos a tierra, como en el caso de la lógica acoplada por emisor (ECL), la lógica acoplada por emisor positiva (PECL) o la lógica de señalización diferencial de bajo voltaje (LVDS).

3.5.2. Códificación

Una palabra o *Word* en inglés, es grupo de niveles que representan un número digital. Estos niveles pueden presentarse de forma simultánea en *paralelo*, en un bus o en grupos de entradas o salidas de compuertas, de forma *serial* (es decir, en una secuencia temporal) en una única línea, o como una secuencia de bytes paralelos (serial por bytes) o nibbles (porciones más pequeñas de bytes). Por ejemplo, una palabra de 16 bits puede ocupar los 16 bits de un bus de 16 bits, dividirse en dos bytes secuenciales para un bus de 8 bits, o en cuatro nibbles de 4 bits para un bus de 4 bits.

A cada nivel analógico cuantificado (es decir, que representa una porción única del rango analógico) se le asigna una agrupación, ya sea paralelo o en serie única de niveles digitales, o un *número* o *código*. Un código digital típico sería el siguiente arreglo:

$$a_7\ a_6\ a_4\ a_3\ a_2\ a_1\ a_0 = 1\ 0\ 1\ 1\ 1\ 0\ 0\ 1 \tag{3.2}$$

Está compuesto por ocho bits. El "1" en el extremo izquierdo se denomina "bit más significativo" (MSB, o Bit 1), mientras que el del extremo derecho se llama "bit menos significativo" (LSB, o Bit N: 8 en este caso). El significado del código, ya sea como un número, un carácter o una representación de una variable analógica, no se conoce hasta que se defina tanto el *código* como su *relación de conversión*. Es importante no confundir la designación de un bit específico (como Bit 1, Bit 2, etc.) con los subíndices asociados al arreglo "a". Estos subíndices corresponden a la potencia de 2 vinculada al peso de cada bit dentro de la secuencia.

El código más conocido, aparte del sistema decimal (base 10), es el binario natural o directo (base 2). Los códigos binarios son más utilizados para representar números enteros. En un código binario natural de N bits, el LSB tiene un peso de 2^0 (es decir,

1), el siguiente bit tiene un peso de 2^1 (o 2), y así sucesivamente hasta llegar al MSB, cuyo peso es de 2^{N-1} (o $2^N/2$). El valor de un número binario se obtiene sumando los pesos de todos los bits no nulos. Cuando estos bits ponderados se suman, forman un número único que puede variar entre 0 y $2^N - 1$. Cada bit adicional en cero al final, si existe, esencialmente duplica el tamaño del número.

En la tecnología de convertidores, el valor de escala completa (abreviado como FS) es independiente del número de bits de resolución, N. Un sistema de codificación más práctico es el binario fraccionario, que siempre se normaliza a la escala completa. El binario entero puede interpretarse como binario fraccionario dividiendo todos los valores enteros por 2^N. Por ejemplo, el MSB tiene un peso de 1/2 (es decir, $2^{(N-1)}/2^N = 2^{-1}$), el siguiente bit tiene un peso de 1/4 (o 2^{-2}), y así sucesivamente hasta el LSB, que tiene un peso de $1/2^N$ (o 2^{-N}). Al sumar los bits ponderados, se obtiene un número con cualquiera de los 2^N valores, que van desde 0 hasta $(1-2^{-N})$ de la escala completa. Los bits adicionales simplemente añaden mayor detalle sin modificar el rango de la escala completa. La relación entre los números en base 10 y los números binarios (base 2) se ilustra en las siguientes ecuaciones:

$$\text{Numbero entero}_{10} = \underbrace{a_{N-1}2^{N-1}}_{\text{MSB}} + a_{N-2}2^{N-2} + \ldots + a_1 2^1 + \underbrace{a_0 2^0}_{\text{LSB}} \tag{3.3}$$

$$\text{Numbero fraccionario}_{10} = \underbrace{a_{N-1}2^{-1}}_{\text{MSB}} + a_{N-2}2^{-2} + \ldots + a_1 2^{-(N-1)} + \underbrace{a_0 2^{-N}}_{\text{LSB}} \tag{3.4}$$

En los sistemas de conversión de datos, el método de codificación debe estar relacionado con el rango de entrada analógica (o intervalo) de un ADC, o con el rango de salida analógica de un DAC. El caso más sencillo ocurre cuando la entrada del ADC o la salida del DAC es una tensión unipolar positiva (aunque las salidas de corriente son comunes en los DACs, no lo son tanto en las entradas de ADCs). El código más utilizado para este tipo de señal es el binario directo, como se muestra en la Tabla 3.2 para un convertidor de 4 bits. Como se puede observar, existen 16 niveles distintos posibles, desde el código 0000 (todos ceros) hasta el código 1111 (todos unos). Es importante destacar que el valor analógico representado por el código de todos unos no corresponde a la escala completa (FS), sino a FS menos 1 LSB. Esta es una convención común en la notación de conversión de datos, aplicable tanto a ADCs como a DACs. La Tabla 3.2 muestra el número equivalente en base 10, el valor del código binario en base 2 en relación a la escala completa (FS), y el nivel de voltaje correspondiente para cada código (suponiendo un convertidor de escala completa de 10 V). También se muestra el código Gray equivalente, que será discutido más adelante

Tabla 3.2: Códigos binarios unipolares, convertidor de 4 bits.

Número en base 10	Escala	+10V FS	Binario	Gray
+15	+FS – 1LSB = +15/16 FS	9.375	1111	1000
+14	+7/8 FS	8.750	1110	1001
+13	+13/16 FS	8.125	1101	1011
+12	+3/4 FS	7.500	1100	1010
+11	+11/16 FS	6.875	1011	1110
+10	+5/8 FS	6.250	1010	1111
+9	+9/16 FS	5.625	1001	1101
+8	+1/2 FS	5.000	1000	1100
+7	+7/16 FS	4.375	0111	0100
+6	+3/8 FS	3.750	0110	0101
+5	+5/16 FS	3.125	0101	0111
+4	+1/4 FS	2.500	0100	0110
+3	+3/16 FS	1.875	0011	0010
+2	+1/8 FS	1.250	0010	0011
+1	1LSB = +1/16 FS	0.625	0001	0001
0	0	0.000	0000	0000

3.5.3. Cuantización

La Figura 3.18 muestra la función de transferencia de un DAC ideal de 3 bits con codificación binaria directa en la entrada. Se puede observar que la salida analógica es cero cuando el código de entrada es todo ceros. A medida que el código de entrada digital aumenta, la salida analógica incrementa 1 LSB (equivalente a 1/8 de la escala en este caso) por cada cambio de código. El voltaje de salida más alto es de 7/8 de la escala completa (FS), lo que corresponde a un valor de FS - 1 LSB. La salida a mitad de escala, equivalente a 1/2 FS, se produce cuando el código de entrada digital es 100.

La función de transferencia de un ADC ideal de 3 bits se ilustra en la Figura 3.18. En este caso, existe un rango de voltaje de entrada analógico dentro del cual el ADC genera un código de salida específico; este rango, conocido como *incertidumbre de cuantización*, es equivalente a 1 LSB. Es importante destacar que, en un ADC ideal, la amplitud de las regiones de transición entre códigos adyacentes es cero. No obstante, en la práctica, siempre hay ruido de transición asociado a estos niveles, lo que provoca que la amplitud no sea nula. Por convención, la entrada analógica correspondiente a un código determinado se define por el *centro del código*, ubicado a medio camino entre dos regiones de transición adyacentes (representado por los puntos negros en el diagrama). Esto implica que la primera región de transición se produce a 1/2 LSB. El voltaje de entrada analógico de escala completa se establece en 7/8 FS (FS - 1 LSB).

Otro código que merece ser mencionado es el código Gray (o binario reflejado). El equivalente del código binario directo de 4 bits en código Gray también se ilustra en la

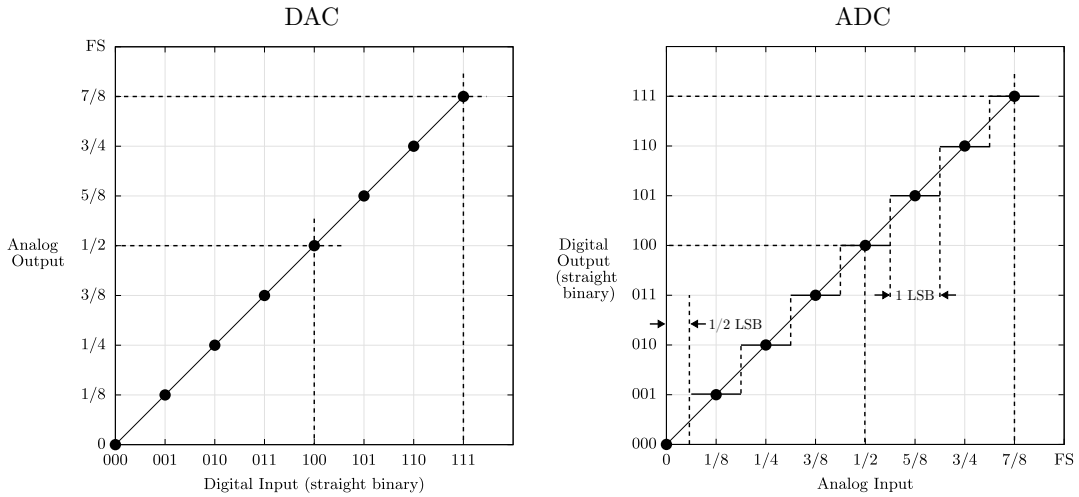


Figura 3.18: Funciones de transferencia para un DAC y ADC unipolares de 3 bits.

Tabla 3.2. Aunque su uso en aritmética de computadoras es poco común, el código Gray posee propiedades que lo hacen útil en la conversión A/D. Una característica destacada es que, en el código Gray, cuando el valor del número cambia, las transiciones de un código al siguiente afectan solo a un bit a la vez. Esto es en contraste con el código binario, donde todos los bits cambian en la transición entre 0111 y 1000. Algunos ADCs lo emplean de manera interna y luego lo convierten a código binario para su uso externo.

3.5.4. Códigos binarios

En muchos sistemas, es deseable representar cantidades analógicas tanto positivas como negativas mediante códigos binarios. Existen varios métodos para lograr esto, como el binario con desplazamiento, el complemento a dos, el complemento a uno o la magnitud con signo, aunque los más comunes son el binario con desplazamiento y el complemento a dos. Las relaciones entre estos códigos para un sistema de 4 bits se ilustran en la Tabla 3.3. Es importante destacar que los valores están ajustados para un rango de voltaje de entrada/salida de ±5 V en escala completa.

En el sistema binario con desplazamiento, el valor de señal cero se asigna al código 1000. La secuencia de códigos es idéntica a la del binario directo, siendo la única diferencia el desplazamiento de media escala asociado a la señal analógica. El valor más negativo (–FS + 1 LSB) se representa con el código 0001, mientras que el valor más positivo (+FS – 1 LSB) se representa con el código 1111. Es importante señalar que, para mantener una simetría perfecta alrededor de la mitad de la escala, el código de todos ceros (0000), que representa la escala completa negativa (–FS), generalmente no se utiliza en los cálculos. Puede emplearse para indicar una condición de fuera de rango negativa o asignarse simplemente el valor de 0001 (–FS + 1 LSB).

La relación entre el código binario con desplazamiento y el rango de salida analógica

Tabla 3.3: Códigos bipolares, convertidor de 4 bits.

Número en base 10	Escala	±5V FS	Offset binary	Twos comp	Ones comp	Sign Mag
+7	+FS - 1LSB = +7/8 FS	+4.375	1111	0111	0111	0111
+6	+3/4 FS	+3.750	1110	0110	0110	0110
+5	+5/8 FS	+3.125	1101	0101	0101	0101
+4	+1/2 FS	+2.500	1100	0100	0100	0100
+3	+3/8 FS	+1.875	1011	0011	0011	0011
+2	+1/4 FS	+1.250	1010	0010	0010	0010
+1	+1/8 FS	+0.625	1001	0001	0001	0001
0	0	0.000	1000	0000	*0000	*1000
-1	-1/8 FS	-0.625	0111	1111	1110	1001
-2	-1/4 FS	-1.250	0110	1110	1101	1010
-3	-3/8 FS	-1.875	0101	1101	1100	1011
-4	-1/2 FS	-2.500	0100	1100	1011	1100
-5	-5/8 FS	-3.125	0011	1011	1010	1101
-6	-3/4 FS	-3.750	0010	1010	1001	1110
-7	-FS + 1LSB = -7/8 FS	-4.375	0001	1001	1000	1111
-8	-FS	-5.000	0000	1000		

de un DAC bipolar de 3 bits se muestra en la Figura 3.19. La salida analógica del DAC es cero cuando el código de entrada es 100. El voltaje de salida más negativo suele estar definido por el código 001 (–FS + 1 LSB), mientras que el más positivo lo define el código 111 (+FS – 1 LSB). El voltaje de salida correspondiente al código de entrada 000 se puede utilizar si se desea, pero esto provoca que la salida no sea simétrica respecto al cero y complica los cálculos.

El código de salida binario con desplazamiento para un ADC bipolar de 3 bits, en función de su entrada analógica, se muestra en la Figura 3.19. Es importante señalar que la entrada analógica cero define el centro del código de mitad de escala, que es 100. Al igual que en los DACs bipolares, el voltaje de entrada más negativo está definido por el código 001 (–FS + 1 LSB), mientras que el más positivo lo define el código 111 (+FS – 1 LSB). Como se mencionó anteriormente, el código de salida 000 está disponible para su uso si se desea, pero esto provoca que la salida no sea simétrica respecto al cero y complica los cálculos matemáticos

El complemento a dos es idéntico al binario con desplazamiento, con la diferencia de que el bit más significativo (MSB) se invierte. Esto es muy fácil de implementar en un conversor de datos mediante un inversor simple o utilizando la salida complementaria de un flip-flop tipo D. La popularidad del complemento a dos radica en la facilidad con la que se pueden realizar operaciones matemáticas en computadoras y procesadores DSP. En términos de conversión, el complemento a dos se compone de un código binario para magnitudes positivas (bit de signo 0) y el complemento a dos de cada número positivo para representar su equivalente negativo. El complemento a dos se forma aritméticamente invirtiendo el número y sumando 1 LSB. Por ejemplo, para obtener –3/8 de FS, se toma el complemento a dos de +3/8 de FS. Primero se invierte +3/8 de FS, es decir, 0011, obteniendo 1100. Luego, al sumar 1 LSB, obtenemos 1101.

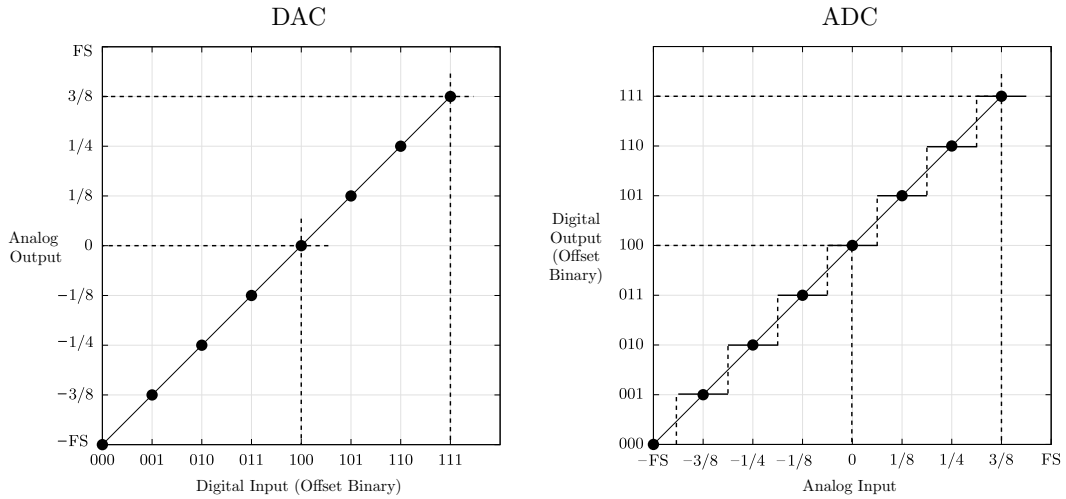


Figura 3.19: Funciones de transferencia para un DAC y ADC bipolares ideales de 3 bits.

El complemento a dos simplifica las operaciones de resta. Por ejemplo, para restar 3/8 de FS de 4/8 de FS, se suma 4/8 a –3/8, es decir, se suman 0100 y 1101. El resultado es 0001, que equivale a 1/8, ignorando el acarreo adicional.

El complemento a uno también puede emplearse para representar números negativos, aunque es mucho menos común que el complemento a dos y su uso es raro en la actualidad. El complemento a uno se obtiene simplemente invirtiendo todos los dígitos de un número positivo. Por ejemplo, el complemento a uno de 3/8 FS (0011) es 1100. Un código con complemento a uno se forma invirtiendo cada valor positivo para generar su valor negativo correspondiente. Esto incluye el cero, que puede representarse con dos códigos: 0000 (conocido como 0+) o 1111 (conocido como 0–). Esta ambigüedad debe resolverse matemáticamente, lo que presenta problemas obvios para los ADCs y DACs, donde solo existe un código para representar el valor cero.

La representación de magnitud con signo podría parecer la forma más sencilla de expresar digitalmente cantidades analógicas con signo. Consiste simplemente en determinar el código adecuado para la magnitud y añadir un bit de polaridad. El sistema BCD de magnitud con signo es común en voltímetros digitales bipolares, pero presenta el problema de tener dos códigos posibles para representar el valor cero. Por este motivo, no es muy popular en la mayoría de las aplicaciones que involucran ADCs o DACs.

3.5.5. Resolución

Lo más importante a tener en cuenta sobre los DACs y ADCs es que, ya sea la entrada o la salida, es digital, lo que significa que la señal está cuantizada. En otras palabras, una palabra de N bits representa uno de los 2^N estados posibles. Por lo tanto,

un DAC de N bits (con una referencia fija) solo puede generar 2^N salidas analógicas posibles, y un ADC de N bits solo puede producir 2^N salidas digitales. Como se mencionó anteriormente, las señales analógicas generalmente serán voltajes o corrientes.

La resolución de los convertidores de datos puede expresarse de diversas maneras: en términos del peso del bit menos significativo (LSB), en partes por millón de la escala completa (ppm FS), en milivoltios (mV), entre otros. Dado que los dispositivos, incluso los provenientes del mismo fabricante, pueden tener especificaciones diferentes, los usuarios de convertidores deben aprender a interpretar y convertir entre estos distintos tipos de especificaciones para poder comparar dispositivos con éxito.

Antes de analizar las distintas arquitecturas utilizadas en los convertidores de datos, es necesario evaluar el rendimiento que se puede esperar y las especificaciones que resultan importantes. En las siguientes secciones se abordará la definición de los errores y las especificaciones asociadas a los convertidores de datos. Comprender esto es fundamental para identificar las fortalezas y debilidades de las distintas arquitecturas de ADC/DAC.

La Figura 3.20 muestra las características de transferencia ideales de un DAC unipolar de 3 bits y un ADC unipolar de 3 bits. En un DAC, tanto la entrada como la salida están cuantizadas, por lo que el gráfico consta de ocho puntos. Si bien es posible discutir la línea que conecta estos puntos, es fundamental recordar que la característica de transferencia real no es una línea continua, sino un conjunto de puntos discretos.

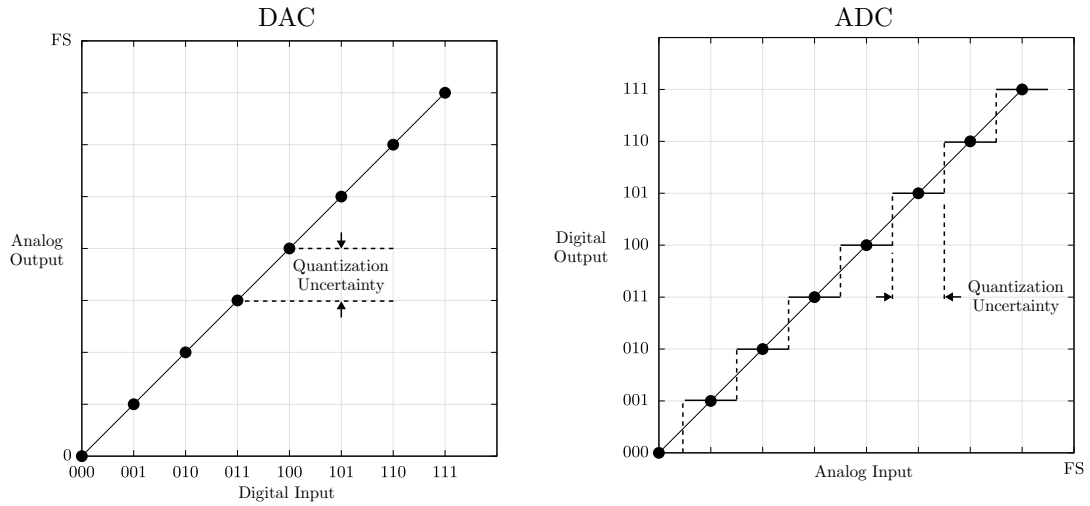


Figura 3.20: Funciones de transferencia para DAC y ADC ideales de 3 bits.

La entrada de un ADC es analógica y no está cuantizada, mientras que su salida sí lo está. Por ello, la característica de transferencia se representa mediante ocho escalones horizontales. Para analizar el desplazamiento, la ganancia y la linealidad de un ADC, se considera la línea que une los puntos medios de estos escalones, conocidos habitualmente como los *centros de código*.

3.5.6. Errores en los convertidores

En los DACs y ADCs, la escala completa digital (todos los bits en 1) corresponde a 1 LSB por debajo de la escala completa analógica (FS). Las transiciones ideales en un ADC ocurren a 1/2 LSB por encima de cero y, a partir de ahí, en cada LSB hasta llegar a $1\frac{1}{2}$ LSB por debajo de la escala completa analógica. Como la entrada analógica de un ADC puede tomar cualquier valor, mientras que la salida digital está cuantizada, puede haber una diferencia de hasta 1/2 LSB entre la entrada analógica real y el valor exacto de la salida digital. Este fenómeno se conoce como *error de cuantización* o *incertidumbre de cuantización*, como se muestra en la Figura 3.20. En aplicaciones de señal alterna (muestreo), este error de cuantización genera ruido de cuantización.

Los cuatro errores de corriente continua en un convertidor de datos son: *error de offset*, *error de ganancia* y dos tipos de *errores de linealidad* (*diferencial* e *integral*). Los errores de offset y ganancia son similares a los que se presentan en amplificadores, para un rango de entrada bipolar. (Cabe señalar que, aunque los errores de offset y de cero son idénticos en amplificadores y convertidores de datos unipolares, no lo son en convertidores bipolares, por lo que deben diferenciarse con cuidado).

Las características de transferencia tanto de los DACs como de los ADCs pueden representarse como una línea recta expresada por D = K + GA, donde D es el código digital, A es la señal analógica, y K y G son constantes. En un convertidor unipolar, el valor ideal de K es cero; en un convertidor bipolar con offset, es –1 LSB. El error de offset es la diferencia entre el valor real de K y su valor ideal.

El error de ganancia es la diferencia entre G y su valor ideal, y generalmente se expresa como un porcentaje de esa diferencia, aunque también puede definirse como la contribución del error de ganancia (en mV o LSB) al error total en escala completa. Estos errores suelen poder corregirse por el usuario del convertidor de datos. Sin embargo, es importante señalar que en los amplificadores, el ajuste de desfase se realiza con una entrada de cero, mientras que la ganancia se ajusta cerca de la escala completa. El algoritmo de ajuste para un convertidor de datos bipolar es más complejo.

El error de linealidad integral de un convertidor es similar al error de linealidad de un amplificador, y se define como la desviación máxima de la característica de transferencia real del convertidor respecto a una línea recta, generalmente expresada como un porcentaje de la escala completa (aunque también puede expresarse en LSBs). En el caso de un ADC, la convención más común es trazar una línea recta a través de los puntos medios de los códigos, o centros de código. Hay dos métodos comunes para determinar esta línea recta: la línea basada en los puntos extremos y la mejor línea recta.

El otro tipo de no linealidad en un convertidor es la no linealidad diferencial (DNL), que se refiere a la linealidad de las transiciones de código en el convertidor. Idealmente,

un cambio de 1 LSB en el código digital debería corresponder a un cambio exacto de 1 LSB en la señal analógica. En un DAC, esto significa que un cambio de 1 LSB en el código digital genera exactamente 1 LSB de cambio en la salida analógica, mientras que en un ADC se requiere un cambio de 1 LSB en la entrada analógica para pasar de una transición digital a la siguiente. El error de linealidad diferencial se define como la desviación máxima de cualquier cambio cuántico (o de LSB) en toda la función de transferencia con respecto a su tamaño ideal de 1 LSB.

Cuando el cambio en la señal analógica correspondiente a un cambio de 1 LSB en el código digital es mayor o menor que 1 LSB, se considera que existe un error de DNL. El error de DNL en un convertidor se define generalmente como el valor máximo de DNL que puede encontrarse en cualquier transición dentro del rango del convertidor. La Figura 3.21 muestra las funciones de transferencia no ideales de un DAC y un ADC, mostrando los efectos del error de DNL.

Si el DNL de un DAC es inferior a –1 LSB en alguna transición, el DAC se considera no monótono, lo que significa que su característica de transferencia presenta uno o más máximos o mínimos localizados. Un DNL superior a +1 LSB no provoca no monotonía, pero sigue siendo indeseable

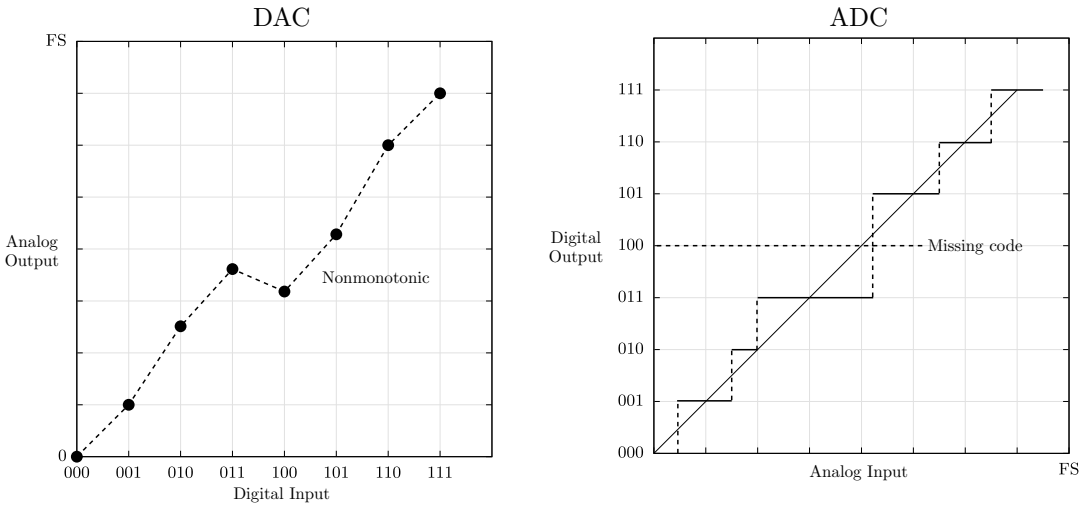


Figura 3.21: Funciones de transferencia para DAC y ADC no ideales de 3 bits.

La monotonía de un DAC a menudo se especifica de manera explícita en las hojas de datos, aunque si se garantiza que el DNL es menor a 1 LSB (es decir, $|\text{DNL}| \leq 1$ LSB), el dispositivo será monótono, incluso sin una garantía explícita.

Los ADCs pueden presentar no monotonía, pero un resultado más común del exceso de DNL en los ADCs es la aparición de códigos faltantes. La ausencia de códigos en un ADC es tan problemática como la no monotonía en un DAC. Esto ocurre, al igual que en los DACs, cuando el DNL es inferior a –1 LSB.

En un DAC, no puede haber códigos faltantes, ya que cada palabra de entrada

digital producirá una salida analógica correspondiente. Sin embargo, los DACs pueden ser no monótonos, como se mencionó anteriormente. En un DAC binario directo, el punto más probable para que ocurra una condición de no monotonía es en la mitad de la escala, entre los códigos 011...11 y 100...00. Si se produce una condición de no monotonía en este punto, generalmente se debe a que el DAC no está calibrado o ajustado correctamente. Un ADC de aproximación sucesiva con un DAC interno no monótono normalmente generará códigos faltantes, pero seguirá siendo monótono. Sin embargo, es posible que un ADC sea no monótono, dependiendo de la arquitectura de conversión específica.

3.5.7. Convertidores Analógico-Digitales

Las arquitecturas de Conversores Analógico-Digital (ADC) juegan un papel fundamental en la digitalización de señales analógicas, permitiendo la conversión de señales continuas en una representación discreta para su posterior procesamiento digital. En esta sección, se presentan las principales arquitecturas de ADC, incluyendo sus ventajas, desventajas y aplicaciones más comunes. Posteriormente, se realizará una comparación detallada entre estas arquitecturas.

3.5.7.1. Convertidor Flash

El convertidor Flash es la arquitectura de ADC más rápida disponible en la actualidad, ya que realiza la conversión en un solo ciclo. Este tipo de convertidor utiliza $2^N - 1$ comparadores para realizar la conversión directa de una señal analógica en un código digital de N bits. Cada comparador está polarizado con un valor de referencia que representa una fracción del rango completo de la señal analógica, típicamente espaciada en pasos de un bit menos significativo (LSB). En el caso de un convertidor Flash de 4 bits, la señal analógica se compara simultáneamente con 15 valores de referencia mediante 15 comparadores.

Una de las principales ventajas de los ADC Flash es su alta velocidad, ya que la conversión se lleva a cabo en un solo ciclo. Esto los convierte en la elección preferida para aplicaciones de alta velocidad, como en sistemas de comunicaciones y en aplicaciones de captura de imágenes donde el tiempo de conversión es crucial. Sin embargo, la desventaja más significativa de los ADC Flash es la cantidad exponencial de comparadores necesarios para aumentar la resolución. Por ejemplo, un ADC de 8 bits requiere 255 comparadores, mientras que uno de 16 bits necesitaría 65,535 comparadores, lo que hace que esta arquitectura sea poco práctica para resoluciones más altas debido a la complejidad del diseño, el alto consumo de energía y el coste.

3.5.7.2. Convertidor Pipeline

El convertidor Pipeline (o de etapas en cascada) resuelve algunas de las limitaciones del ADC Flash, dividiendo el proceso de conversión en varias etapas secuenciales. Cada una de estas etapas incluye un circuito de retención y muestreo, un ADC de m bits (generalmente de tipo Flash), un DAC de m bits. Primero, el circuito de muestreo y retención de la primera etapa adquiere la señal. Luego, el convertidor flash de m bits convierte la señal muestreada en datos digitales. El resultado de la conversión forma los bits más significativos de la salida digital. Esta misma salida digital se introduce en un convertidor digital-analógico de m bits, y su salida se resta de la señal original muestreada. La señal analógica residual se amplifica y se envía a la siguiente etapa de la cadena para ser muestreada y convertida de manera similar a la primera etapa. Este proceso se repite en tantas etapas como sean necesarias para lograr la resolución deseada. En principio, un convertidor en pipeline con p etapas, cada una con un convertidor flash de m bits, puede producir un ADC de alta velocidad con una resolución de $R = p \times m$ bits, utilizando $p \times (2^m - 1)$ comparadores.

Una ventaja clave de los ADCs Pipeline es que permiten obtener resoluciones más altas que los ADCs Flash sin requerir un número exponencial de comparadores. Por ejemplo, un ADC de Pipeline con dos etapas de 8 bits requiere solo 30 comparadores en lugar de los 255 que necesitaría un ADC Flash de 8 bits. Además, los convertidores Pipeline pueden alcanzar tasas de conversión comparables a los ADCs Flash, ya que las etapas posteriores pueden procesarse de manera simultánea con diferentes señales, maximizando el rendimiento. Sin embargo, una desventaja de los ADC Pipeline es la latencia igual a p ciclos introducida por las múltiples etapas, lo que los hace menos adecuados para aplicaciones que requieren una conversión inmediata.

3.5.7.3. Convertidor de Aproximaciones Sucesivas (SAR)

El convertidor de Aproximaciones Sucesivas (SAR) sigue un principio de funcionamiento diferente al del ADC Flash. En lugar de realizar todas las comparaciones de manera simultánea, el ADC SAR emplea un único comparador que realiza una serie de comparaciones secuenciales. El proceso comienza comparando la señal analógica con la mitad de la escala completa (correspondiente al bit más significativo) y, en función del resultado, retiene o descarta ese valor. El proceso se repite para los siguientes bits de menor peso, reduciendo progresivamente el rango de comparación hasta alcanzar la resolución deseada.

La arquitectura SAR es eficiente en términos de hardware, ya que requiere solo un comparador y un DAC de alta precisión para generar los niveles de referencia. Sin embargo, debido a que el ADC SAR necesita N ciclos de comparación para resolver una conversión de N bits, su velocidad es inferior a la de los ADCs Flash y Pipeline. A pesar

de ello, el ADC SAR es ideal para aplicaciones de alta resolución y baja velocidad, como la adquisición de señales no periódicas o señales que requieren alta precisión, como en la instrumentación médica. Además, el ADC SAR es altamente eficiente en términos de consumo de energía y es una opción común en aplicaciones de bajo consumo.

3.5.7.4. Convertidor Sigma-Delta

El convertidor Sigma-Delta ($\Sigma\Delta$) emplea un enfoque completamente diferente al de las arquitecturas anteriores. En lugar de realizar una conversión directa de la señal analógica en un número binario, el ADC Sigma-Delta sobreamuestrea la señal analógica y la convierte en una secuencia de bits a través de un proceso de integración y comparación. Este flujo de bits se procesa mediante un filtro digital y se decima para obtener una salida digital de alta resolución.

El Sigma-Delta es la arquitectura preferida para aplicaciones que requieren alta resolución y bajo ancho de banda, como la medición de precisión en instrumentos científicos. Una de las principales ventajas del ADC Sigma-Delta es su capacidad para mitigar el ruido de baja frecuencia mediante el modelado de ruido, desplazando gran parte de ese ruido a frecuencias más altas, fuera de la banda de interés. Además, debido al alto índice de sobremuestreo, la necesidad de filtros anti-alias externos se reduce significativamente. Sin embargo, una de las limitaciones del ADC Sigma-Delta es su alta latencia, lo que lo hace inadecuado para aplicaciones de respuesta rápida o señales multiplexadas.

3.5.7.5. Comparación entre las Arquitecturas de ADC

En la Tabla 3.4 se presenta una comparación detallada entre las principales arquitecturas de ADC en función de parámetros clave como la velocidad, resolución, latencia, complejidad de diseño y adecuación para aplicaciones específicas.

Tabla 3.4: Comparación entre arquitecturas de ADC.

Arquitectura	Velocidad	Resolución	Latencia	Complejidad	Aplicación Ideal
Flash	Muy alta	Baja	Muy baja	Muy alta	Sistemas de alta velocidad (telecomunicaciones)
Pipeline	Alta	Media-alta	Media	Alta	Alta resolución con alta velocidad (imagen, radar)
SAR	Media	Alta	Baja	Baja	Instrumentación de precisión, adquisición de datos
Sigma-Delta	Baja	Muy alta	Alta	Media-alta	Medición de precisión, instrumentos científicos

La Tabla 3.5 resume y clasifica (de manera general) las ventajas relativas de las arquitecturas flash, pipeline, SAR y sigma-delta. Una clasificación de 1 en una categoría de rendimiento indica que la arquitectura es inherentemente superior a las demás en esa categoría. Un * indica que la arquitectura posee la capacidad o característica mencionada.

Tabla 3.5: Resumen de tipos de ADC.

Características	**Flash**	**Pipelined**	**SAR**	**Sigma-Delta**
Rendimiento	1	2	3	4
Resolución	4	3	2	1
Latencia	1	3	2	4
Adecuación para convertir múltiples señales por ADC	1	2	1	3
Capacidad para convertir señales multiplexadas no periódicas	1	2	1	3
Antialiasing simplificado				⋆
Puede realizar submuestreo	⋆	⋆	⋆	
Puede aumentar la resolución mediante promediado (con ruido de trémulo)	⋆	⋆	⋆	

La elección de la arquitectura adecuada depende en gran medida de los requisitos específicos de la aplicación. Por ejemplo, en aplicaciones donde se requiere una alta tasa de muestreo, como en comunicaciones o adquisición de imágenes, los convertidores Flash o Pipeline son ideales debido a su alta velocidad de conversión. Por otro lado, si se necesita alta resolución con señales de bajo ancho de banda, como en la instrumentación de precisión, los convertidores Sigma-Delta son la opción más adecuada. Los convertidores SAR, por su parte, son ideales para aplicaciones que requieren alta precisión con un coste relativamente bajo y bajo consumo de energía.

Capítulo 4

Diseño de firmware

En este capítulo se describen las etapas realizadas para implementar un sistema de adquisición de datos capaz de generar una imagen. Estas etapas abarcan la caracterización de resistencias, multiplexores y una fotorresistencia, además del diseño de una PCB para una matriz de fototransistores. Finalmente, se presenta el diseño digital implementado para adquirir una imagen. Todos los diseños fueron descritos en Verilog e implementados en la tarjeta Basys 3 de Digilent, que cuenta con una FPGA Artix-7 (XC7A35T-1CPG236C). La adquisición de datos se realizó en MATLAB.

4.1. Caracterización de resistencias

En esta etapa, se realizó la caracterización de cuatro resistencias utilizando un circuito divisor de voltaje (Figura 4.1). En este circuito, R_{test} simula la resistencia de un pixel de un arreglo de microbolómetros, mientras que R_{ref} emula un circuito de lectura. Trabajar con resistencias de valores conocidos permite verificar con mayor precisión si los resultados experimentales coinciden con los teóricos.

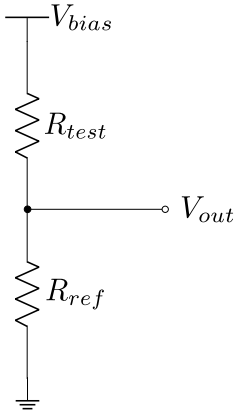


Figura 4.1: Divisor de voltaje como microbolómetro y circuito de lectura

Para determinar cuál es el valor más adecuado de R_{ref} para cualquier R_{test} y el voltaje de polarización del circuito, se llevó a cabo una simulación de dos circuitos en LTSpice. En ambos circuitos, se realizó un barrido de R_{test} desde $1k\Omega$ hasta $10k\Omega$, con incrementos de $1k\Omega$, con el fin de imitar un microbolómetro bajo la influencia de luz y un barrido de la fuente de voltaje V_1 de $0V$ a $3.3V$ en incrementos de $0.1V$. La única diferencia entre los circuitos fue el valor de R_{ref}: en uno de ellos, R_{ref} se fijó en $1k\Omega$, mientras que en el otro se estableció en $10k\Omega$. Esta comparación permitió evaluar cómo afectaba el valor de R_{ref} en la respuesta del sistema ante distintos valores de R_{test}, facilitando la selección del valor más adecuado para optimizar la lectura del circuito.

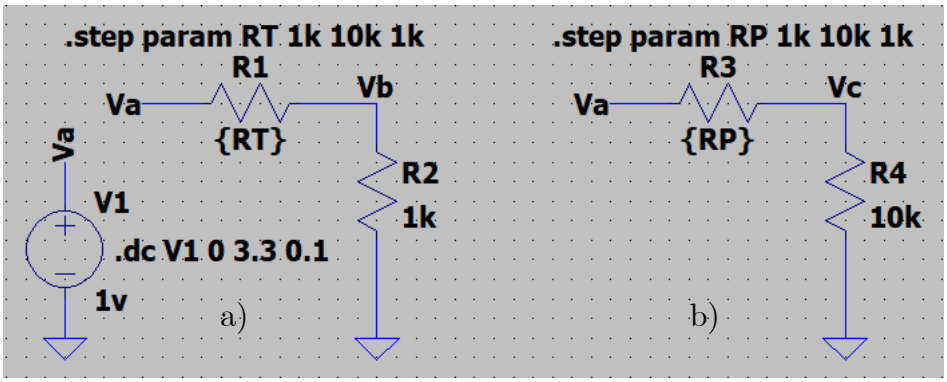


Figura 4.2: Divisores de voltaje para la elección de R_{ref}

En las Figuras 4.3 y 4.4 se presentan los resultados de los barridos de R_{test}.

Supongase que estos resultados son de un microbolómetro cuya resistencia no se puede medir directamente, salvo a través del circuito de lectura, R_{ref}, y despejando R_{test} de la ecuación del divisor de voltaje. Al polarizar el circuito divisor con voltajes muy bajos, resulta imposible identificar con precisión la resistencia del detector. No obstante, al polarizar a partir de $2.7V$, se vuelve más fácil determinar el valor de R_{test}.

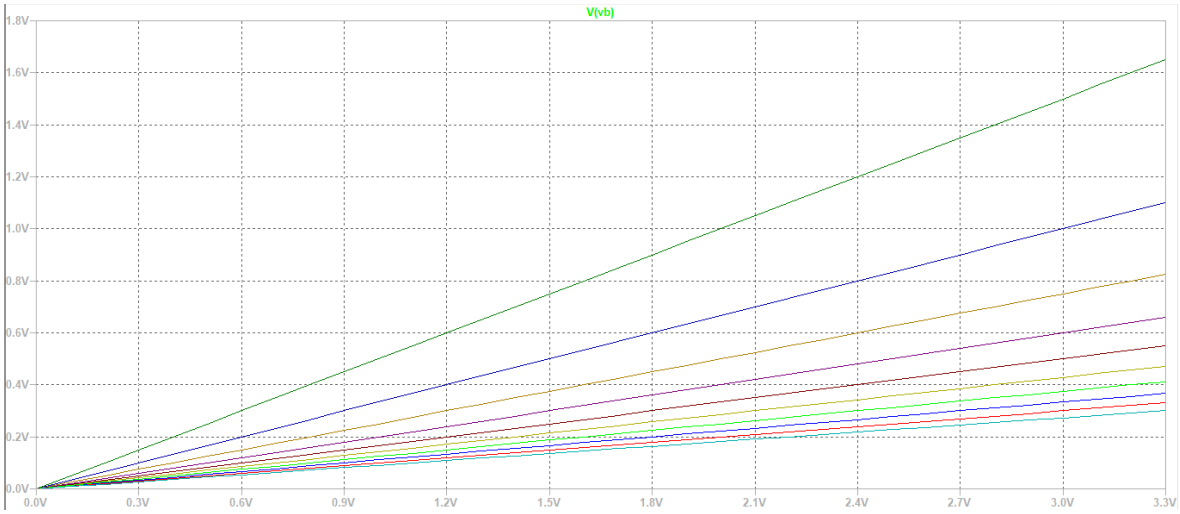


Figura 4.3: Gráficas de voltaje de $R_{ref} = 1k\Omega$ con barrido en R_{test}

Sin embargo, es importante señalar que la influencia de R_{ref} también juega un papel crucial. En la Figura 4.3, con $R_{ref} = 1k\Omega$, se observa que, para valores altos de R_{test}, resulta difícil conocer con precisión la resistencia del detector, siendo únicamente posible identificarla para valores menores a $6k\Omega$. En contraste, en la Figura 4.4, donde $R_{ref} = 10k\Omega$, las curvas para cualquier R_{test} no se empalman y es más sencillo identificar la resistencia del detector, esto llevó a optar por un valor de $R_{ref} = 10k\Omega$ para la caracterización de resistencias.

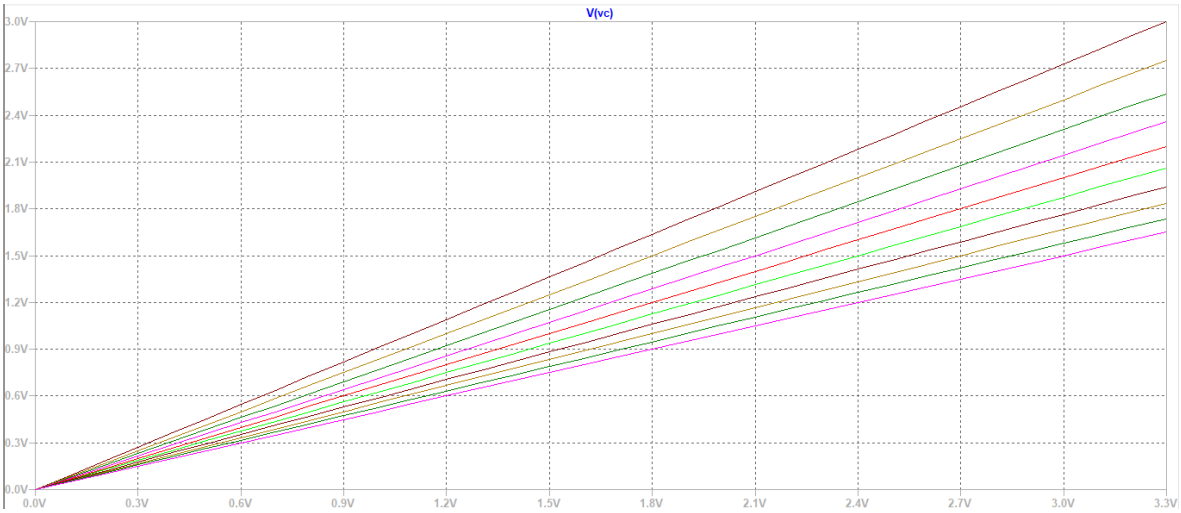


Figura 4.4: Gráficas de voltaje de $R_{ref} = 10k\Omega$ con barrido en R_{test}

Finalmente, se seleccionaron como R_{test} las resistencias de $1k\Omega$, $3.3k\Omega$, $5.6k\Omega$ y $10k\Omega$ con el objetivo de simular el comportamiento de un microbolómetro en función de la cantidad de luz recibida y para proporcionar una mayor variedad en los resultados.

Una vez realizadas las simulaciones y seleccionados los valores de R_{test} y R_{ref}, se procedió al diseño digital en FPGA para la caracterización de resistencias. Este diseño incluyó la implementación de un barrido de voltaje, realizado mediante una memoria ROM y un convertidor D/A, cuyos voltajes se encargan de polarizar el circuito divisor de voltaje. La lectura y conversión de la caída de voltaje en R_{ref} fue llevado a cabo empleando un convertidor A/D, estas conversiones fueron transmitidas a MATLAB a través del protocolo RS232 para su posterior procesamiento.

El diseño digital para la caracterización de resistencias se compone de varios módulos interconectados que permiten la adquisición de datos. Estos módulos son:

- Memoria ROM (rom volts): Contiene valores en binario correspondientes a voltajes de $0V$ a $3V$, con incrementos de $0.1V$. La salida de la memoria ROM está conectada al módulo de escritura del convertidor D/A.

- Escritura del DAC (spi write dac): Convierte y transmite los voltajes guardados en la memoria ROM al circuito divisor de voltaje, permitiendo la polarización del circuito.

- Escritura y lectura del ADC (spi write read adc): Lee y convierte la caída de voltaje en R_{ref}. Los valores obtenidos son almacenados en una memoria RAM.

- Memoria RAM (ram volts): Almacena los voltajes convertidos por el ADC, y su función principal es conservar el valor de la resistencia del microbolómetro ante posibles cambios en la iluminación.

  La palabra de 12 bits obtenida de la RAM se divide en dos partes: una de 8 bits (bits menos significativos) y otra de 4 bits (bits más significativos). La parte de 4 bits se concatena con cuatro ceros, formando así dos palabras de 8 bits, que luego se conectan a un multiplexor.

- Multiplexor (mux ch): Selecciona cuál de las dos palabras de 8 bits almacenadas en la RAM será transmitida a través del módulo UART.

- Transmisión (uart tx): Envía los datos guardados en la RAM y organizados por el multiplexor hacia MATLAB para su procesamiento.

- Contadores: Se utilizan dos contadores con selector, uno para la memoria ROM (counter volts) y otro para la memoria RAM (counter address). Ambos son empleados para direccionar los datos de manera adecuada.

- Máquina de estados (fsm dac adc tx): Es el módulo más importante, compuesto por 17 estados, se encarga de habilitar y coordinar el funcionamiento de todos los módulos anteriormente mencionados. A continuación, se explica la función de cada uno de los estados.

  1. s0: La máquina de estados espera una señal de activación, la cual proviene de un push button.
  2. s1: Una vez activada, se habilita la escritura del DAC para iniciar el proceso de conversión de voltaje.
  3. s2: El DAC se deshabilita, y el sistema permanece en este estado mientras la bandera *end of DAC* (eodac) sea cero. Cuando la bandera cambia a 1, indica que el DAC ha completado su conversión.
  4. s3 y s4: Estados de guarda que aseguran que el circuito divisor de voltaje esté correctamente polarizado.
  5. s5: Se activa el ADC para iniciar la lectura del voltaje de R_{ref}.
  6. s6: El ADC se desactiva, y el sistema permanece en este estado mientras la bandera *end of ADC* (eoadc) sea cero. Cuando cambia a 1, significa que el ADC ha terminado su conversión.
  7. s7: Se habilita la RAM y se almacena la conversión realizada por el ADC.

8. s8: La RAM se deshabilita, y se incrementan los contadores de la ROM y la RAM.
9. s9: El sistema mantiene el valor de los contadores y verifica si el contador de la RAM ha llegado a 31. Si es cierto, pasa al estado s10; si no, vuelve al estado s1.
10. s10: Se reinician los contadores de la ROM y la RAM.
11. s11: Se habilita el módulo de transmisión para comenzar a enviar datos.
12. s12: La transmisión se deshabilita, y el sistema permanece en este estado hasta que la bandera *end of transmission* (eotx) indique que se han enviado los 8 bits más significativos de la salida de la RAM.
13. s13: Se habilita nuevamente la transmisión, y el selector del multiplexor se pone a 1 para enviar los 8 bits menos significativos de la salida de la RAM.
14. s14: La transmisión se desactiva, y el sistema espera hasta que se haya terminado de transmitir los bits menos significativos.
15. s15: El contador de la RAM se incrementa para enviar el siguiente conjunto de datos.
16. s16: El sistema mantiene el valor del contador de la RAM y verifica si ha llegado a 31. Si es cierto, se regresa al estado s0 para repetir el proceso; de lo contrario, se vuelve al estado s11 para continuar la transmisión.

Los códigos de los módulos descritos anteriormente se encuentran en el **Apéndice A**, sección **A.2.3.5**.

4.2. Caracterización de multiplexores

Cuando se quiere trabajar con una matriz de $m \times n$ detectores, se deben emplear dos multiplexores: uno para direccionar las columnas y otro para las filas de la matriz. En este trabajo se utilizaron los multiplexores 74HC4051, cada uno con 8 entradas/salidas (Yn), una entrada/salida común (Z), tres entradas digitales de selección (S0-S2) y un pin de habilitación (E), activo en bajo. Estos multiplexores pueden alimentarse con un voltaje en el rango de $-5V$ a $+5V$.

Al utilizar una matriz de detectores, es esencial comprender cómo los multiplexores afectan el circuito, ya que, al estar conectados a la matriz, influyen en el valor del voltaje que debe caer sobre el circuito de lectura. Para evaluar este impacto, se emplearon las cuatro resistencias seleccionadas como R_{test}, organizándolas en forma de matriz. Los multiplexores fueron alimentados con $3.3V$ mediante la tarjeta Basys 3, y se utilizaron los mismos códigos que en la caracterización de resistencias. De igual forma, se utilizaron los mismos códigos de MATLAB de la sección anterior. Tampoco fue necesario

desarrollar código adicional para controlar los multiplexores, ya que estos se operaron manualmente durante las pruebas.

A continuación, se presenta la tabla de verdad del 74HC4051, que muestra cómo debe ser controlado.

Tabla 4.1: Tabla de verdad de multiplexor 74HC4051.

Inputs				
$\overline{E}$	S2	S1	S0	Channel ON
L	L	L	L	Y0 to Z
L	L	L	H	Y1 to Z
L	L	H	L	Y2 to Z
L	L	H	H	Y3 to Z
L	H	L	L	Y4 to Z
L	H	L	H	Y5 to Z
L	H	H	L	Y6 to Z
L	H	H	H	Y7 to Z
H	X	X	X	switches off

4.3. Caracterización de fotorresistencia

Se utilizó una fotorresistencia para tener una referencia clara de cómo sería trabajar con un microbolómetro real, debido a que ambos detectores comparten un comportamiento similar: en ausencia de luz, su resistencia aumenta, mientras que bajo iluminación, disminuye. En este caso, se empleó la fotorresistencia GL5537, que presenta una resistencia de $2M\Omega$ en oscuridad y un rango de $20K\Omega$ a $30K\Omega$ bajo luz, con un tiempo de respuesta de 20 ms.

Primero, la fotorresistencia fue caracterizada bajo la iluminación de las lámparas de un salón. Posteriormente, se realizaron pruebas en condiciones controladas dentro de una caja oscura, midiendo su resistencia en ausencia de luz. Para simular diferentes niveles de iluminación, se incorporó un LED de 3W, cuya intensidad fue regulada mediante un PWM con cuatro ciclos de trabajo: 25 %, 50 %, 75 % y 100 %.

El diseño digital empleado para esta caracterización es similar al utilizado en la caracterización de resistencias, con la diferencia de que se agregó un módulo PWM para controlar la iluminación y observar la respuesta de la fotorresistencia.

4.4. Sistema de adquisición de datos

En esta etapa final, se desarrolló una PCB para una matriz de 8×8 fototransistores, diseñada en Altium Designer, con el propósito de generar una imagen. En esta sección,

se explicará primero el proceso de diseño de la matriz y, posteriormente, el diseño digital. Varios de los módulos implementados en etapas anteriores fueron reutilizados, junto con la incorporación de nuevos módulos para la obtención de imágenes.

4.4.1. Diseño de PCB

Antes de proceder con el diseño de la PCB, se realizó una búsqueda exhaustiva de detectores que pudieran ser adecuados para generar una imagen. Tras analizar varias opciones, se determinó que el fototransistor ALS-PT19-315C, un detector de luz visible, era el más apropiado, debido a su tiempo de respuesta. Además este sensor ha sido utilizado exitosamente en proyectos similares, lo que garantiza su fiabilidad en aplicaciones de imágenes. Otros factores que influyeron en su selección fueron su precio accesible y su amplia disponibilidad en stock. Adicionalmente, se decidió implementar una matriz de 8×8 fototransistores, ya que este número coincide con las entradas/salidas disponibles en los multiplexores seleccionados. Con esta información se procedió al desarrollo de la PCB.

La creación de la librería de esquemáticos fue el primer paso en el diseño de la PCB. En este caso, los únicos componentes que se incorporaron fueron los headers de 1×8 y el fototransistor ALS-PT19-315C (Figura 4.5), dado que estos elementos se emplean repetidamente en la matriz de 8×8, no fue necesario agregar más componentes a la librería.

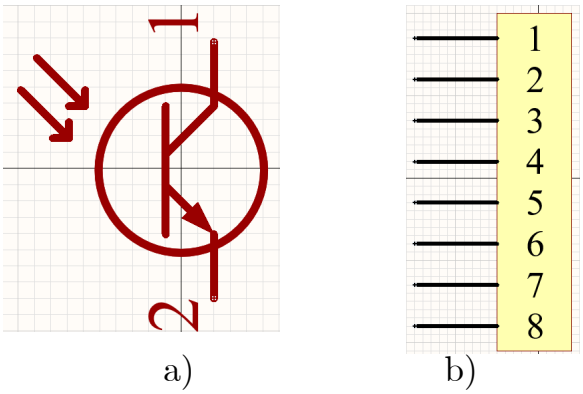


Figura 4.5: Componentes de librería de esquemáticos: a) fototransistor, b) pines de 1×8.

El segundo paso fue realizar el esquemático de la matriz, el cual se muestra en la Figura 4.6. En esta etapa, se realizaron las conexiones entre los componentes, asegurando que cada fototransistor y header estuvieran correctamente vinculados según el diseño de la matriz. Además, en este paso también se asignaron los nombres a las pistas de la PCB, lo que facilita la organización y el enrutamiento de las señales. Este proceso es crucial para asegurar que todos los componentes interactúen correctamente.

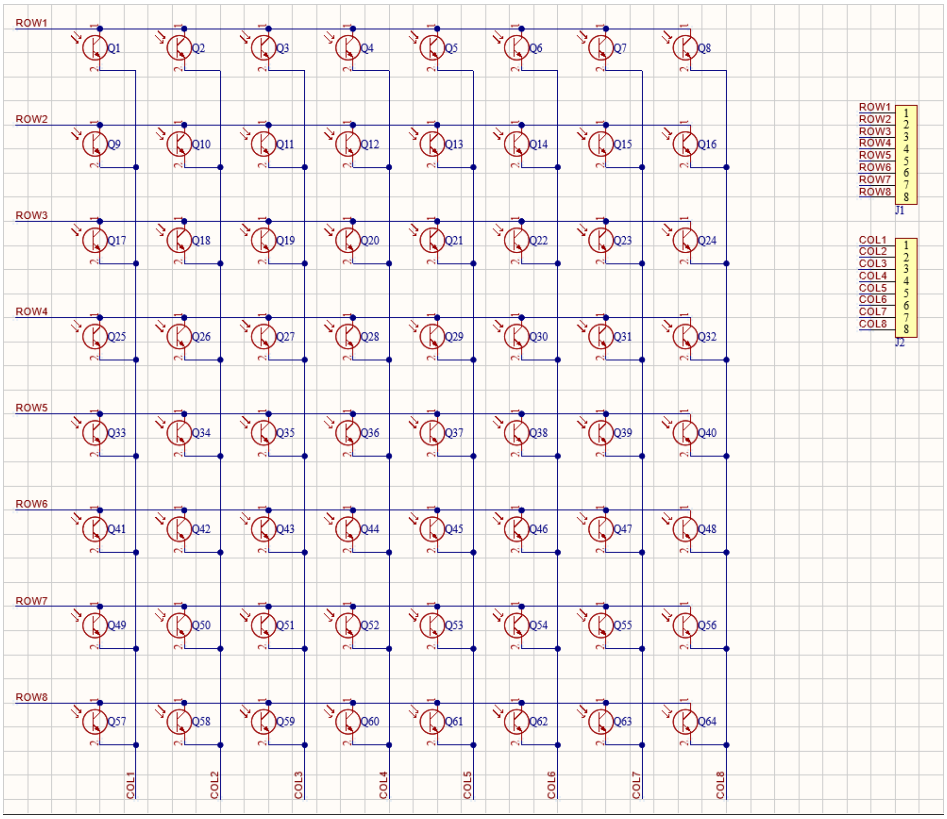


Figura 4.6: Esquemático de matriz de fototransistores.

El siguiente paso fue la generación de los pads de los componentes. Los pads en una PCB son áreas de contacto metálico diseñadas para permitir la conexión eléctrica y mecánica entre los componentes electrónicos y la propia placa. Para garantizar que los pads fueran precisos y compatibles con los componentes utilizados, se revisó el datasheet de cada uno. En este caso, los fototransistores utilizados son de tipo SMD (Surface Mounting Device), lo que requiere pads específicos para montaje superficial, mientras que los headers son THT (Through-Hole Technology), lo que implica la necesidad de pads con perforaciones para su montaje.

Los pads del fototransistor se muestran en la siguiente imagen.

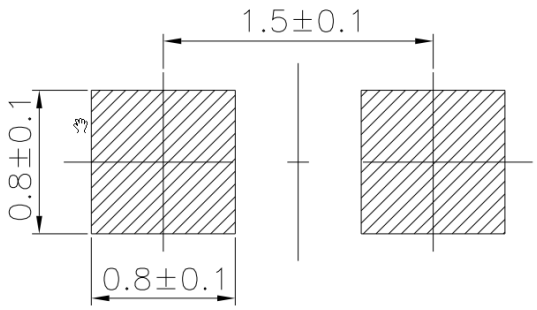


Figura 4.7: Pads de fototransistor ALS-PT19-315C.

Las dimensiones de los pads, de acuerdo con el datasheet del fototransistor, están

especificados en milímetros. En Altium Designer, se definió un tamaño de pad de 0.9 × 0.9 mm con un pitch (la distancia entre los centros de los pads) de 1.6 mm (Figura 4.8). Es importante optar por la mayor dimensión indicada en el datasheet, tanto por seguridad ante posibles variaciones en el tamaño del componente como para asegurar una correcta distribución de la soldadura durante el montaje del componente.

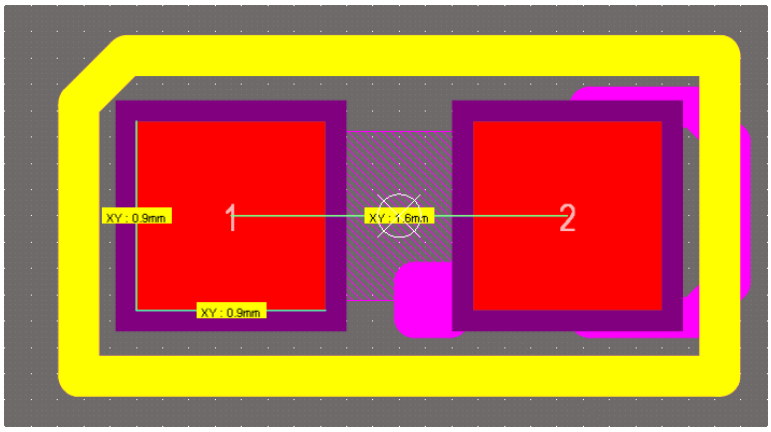


Figura 4.8: Pads de fototransistor ALS-PT19-315C realizados en Altium Designer.

Los pads para los headers de 1 × 8 siguen dimensiones estandarizadas, lo que facilita su integración en diversos diseños de PCBs. El tamaño de los pads es de 1.2 mm, con un pitch de 2.54 mm (Figura 4.9), estándar para la mayoría de los conectores de este tipo. Además, los headers 1 × 8 tienen una longitud aproximada de 20 mm y un ancho de 3 mm.

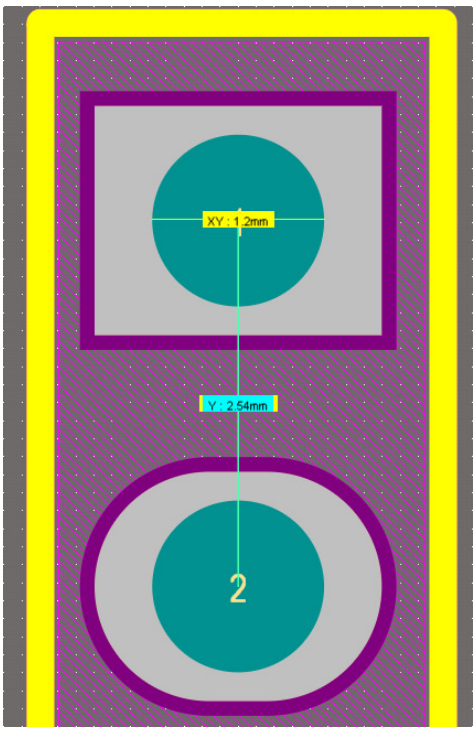


Figura 4.9: Pads de header 1 × 8 realizados en Altium Designer.

Finalmente, se procedió a dar forma a la PCB, organizando los componentes según las necesidades del diseño. Se optó por colocar cada fototransistor en un ángulo de 45°, formando una matriz cuadrada que optimiza el espacio y garantiza una disposición uniforme de los detectores. Para el diseño de la tarjeta, se establecieron varias reglas clave, definidas por el fabricante: el clearance (espacio entre pistas) fue de 2.54 mm, el ancho de las pistas de 0.3 mm, los agujeros con un diámetro mínimo de 0.254 mm y un máximo de 5 mm, y las vías con un diámetro externo de 0.6 mm y un diámetro interno de 0.3 mm.

Tras cotizar con varios proveedores, JLCPCB resultó ser la opción más conveniente en términos de precio y calidad. La PCB diseñada fue de 2 capas, con dimensiones de 40x40 mm, un grosor de 1.6 mm y fabricada en material FR-4.

En la Figura 4.10, se puede apreciar el diseño final de la PCB. Nótese que hay dos colores distintos en las pistas: las pistas de color rojo corresponden a las que pasan por la parte superior de la tarjeta, mientras que las de color azul representan las que pasan por la parte inferior. Esta distribución de las pistas en ambas caras de la tarjeta es la razón por la cual la PCB tiene 2 capas.

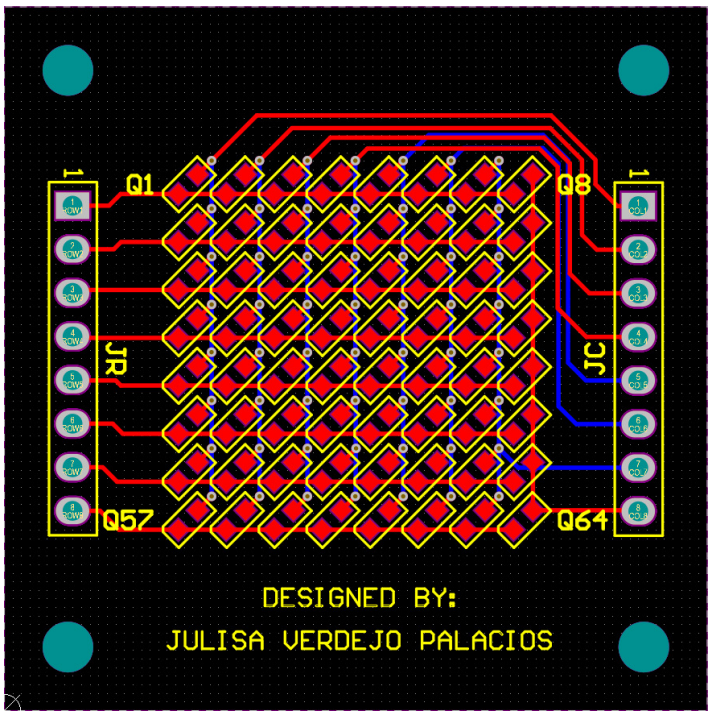


Figura 4.10: Diseño de PCB para una matriz de fototransistores.

4.4.2. Diseño digital

El fototransistor ALS-PT19-315C es un sensor diseñado para detectar luz visible. Su funcionamiento es sencillo: a mayor incidencia de luz, mayor es la corriente generada, y a menor luz, la corriente disminuye.

Este detector puede ser alimentado con un voltaje en un rango entre $2.5V$ y $5V$. Presenta una corriente en oscuridad de 0.1μA. Además, cuenta con un rise time de 0.11 ms y un fall time de 0.22 ms, lo que le permite responder rápidamente a los cambios en la intensidad de la luz. En la siguiente tabla se muestra un resumen de los parámetros mencionados.

Tabla 4.2: Parámetros del fototransistor ALS-PT19-315C

Parameter	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
Supply voltage	V_{CC}	2.5	—	5	V
Dark current	I_{CEO}	—	—	0.1	μA
Rise time	t_r	—	0.11	—	ms
Fall time	t_f	—	0.22	—	ms

En el diseño digital, se reutilizaron varios módulos previamente implementados, como los módulos de escritura y lectura del SPI, la transmisión por UART, la memoria RAM, un multiplexor, el debouncer, y un contador para direccionar la RAM. Además, se incorporaron dos contadores de carrera libre de 3 bits: uno para direccionar las filas y otro para las columnas de la matriz de fototransistores, junto con un contador descendente que permite estabilizar el circuito de lectura. Finalmente, se diseñó una máquina de estados específica para coordinar y controlar el proceso de obtención de imágenes.

La máquina de estados diseñada consta de 18 estados, a continuación, se explicará la función de cada uno de ellos.

- s0: Espera una señal de habilitación, la cual proviene del módulo debouncer para iniciar la máquina de estados.
- s1: Habilita el contador ascendente y espera a que este llegue a cero.
- s2 y s3: Estados de guarda.
- s4: Habilitación del módulo SPI para controlar al ADC.
- s5: Deshabilita el módulo SPI y permanece en este estado hasta que la conversión finalice.
- s6: Habilita la memoria RAM para almacenar la caída de voltaje del circuito de lectura, leída y convertida por el ADC.
- s7: Deshabilita la memoria RAM y habilita el contador de las columnas.

- s8: Verifica si el contador de las columnas ha llegado a cero. Si es así, pasa al estado s9; de lo contrario, regresa a s1 y repite el ciclo hasta que se cumpla la condición.
- s9: Activa el contador de las filas.
- s10: Deshabilita el contador de las filas y verifica si ha llegado a cero. Si es así, pasa al estado s11; de lo contrario, regresa a s1 y repite el ciclo hasta que se cumpla la condición.
- s11: Activa la transmisión.
- s12: Desactiva la transmisión y espera a que esta termine.
- s13: Cambia el selector del multiplexor a 1.
- s14: Habilita nuevamente la transmisión.
- s15: Deshabilita la transmisión y, cuando la bandera eotx sea igual a 1 pasa al estado s16.
- s16: Cambia el selector del multiplexor a 0 y habilita el contador que direcciona a la memoria RAM para transmitir la siguiente conversión del ADC.
- s17: Deshabilita el contador de direcciones y verifica si ha llegado a cero. Si es así, regresa al estado s0; de lo contrario, vuelve al estado s11 para realizar una nueva transmisión de datos.

Dadas las especificaciones anteriores, se optó por polarizar el fototransistor con los 3.3V proporcionados por la tarjeta Basys 3, lo que hizo innecesario el uso del DAC en el proceso de adquisición de imágenes. El circuito de lectura utilizado consistió en una resistencia de $10K\Omega$ y un capacitor de 10nF. Una vez implementado el diseño en la FPGA y conectados los módulos correspondientes, la recepción de los datos se realizó en MATLAB para su posterior procesamiento y generación de la imagen.

Capítulo 5

Mediciones y resultados

En este capítulo se presentan los resultados del diseño del protocolo SPI utilizado para controlar los convertidores D/A y A/D, así como los resultados obtenidos de la caracterización de resistencias, multiplexores y fotorresistencia. Además, se muestran las imágenes generadas por la matriz de fototransistores, evidenciando el correcto funcionamiento del sistema de adquisición de datos implementado y de la PCB diseñada.

5.1. Resultados de protocolo SPI

Para probar el funcionamiento del diseño del protocolo SPI, se utilizó un potenciómetro al que se varió la resistencia, conectado a un canal del convertidor A/D. Para evaluar el convertidor D/A, se asignó un valor de voltaje en formato binario. Las señales generadas por los módulos implementados en la FPGA, junto con las conversiones realizadas por el ADC y DAC, fueron visualizadas mediante un osciloscopio y un multímetro. En esta sección se explicarán los resultados obtenidos de ambos convertidores.

5.1.1. ADC

En el ADC se utilizó el protocolo de lectura y escritura del SPI. De acuerdo al datasheet, una conversión completa requiere 25 ciclos de dclk: durante los primeros 8 ciclos se configura el ADC, luego se deja pasar un ciclo, y en los siguientes 16 ciclos se realiza la conversión. El ADC realiza la conversión en grupos de 8 bits.

En las capturas del osciloscopio (Figura 5.2 y Figura 5.3) se visualizan cuatro señales numeradas del 1 al 4. La señal 1 corresponde al comando de configuración del ADC, la señal 2 al reloj dclk, la señal 3 a la conversión realizada por el ADC, y la señal 4 al chip select (CS). Es importante destacar que las señales 1, 2, y 4 son generadas por los módulos diseñados e implementados en la FPGA, mientras que la señal 3 proviene directamente del ADC, mostrando el resultado de la conversión.

Se realizaron dos pruebas para comprobar el funcionamiento del SPI. En la primera se conectó un potenciómetro al canal 0 del ADC y se configuró hasta obtener un voltaje de 2V. En la siguiente figura se puede observar la conexión.

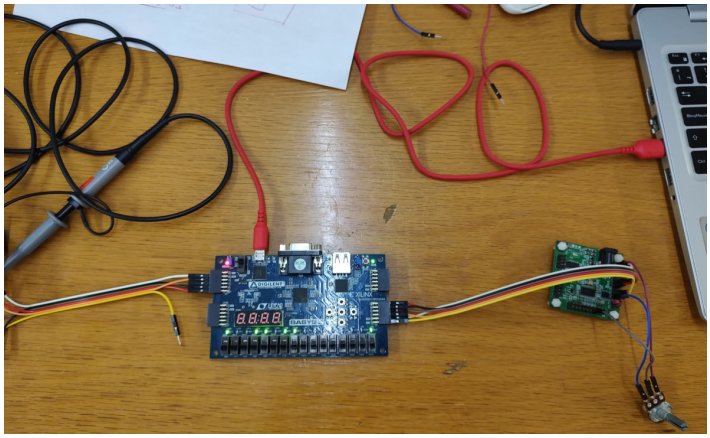


Figura 5.1: Arreglo de potenciómetro, ADC y tarjeta Basys 3.

En la Figura 5.2 se muestra la conversión realizada por el dispositivo, evidenciando el funcionamiento del diseño del protocolo SPI y del ADC al realizar la conversión de la señal analógica a digital.

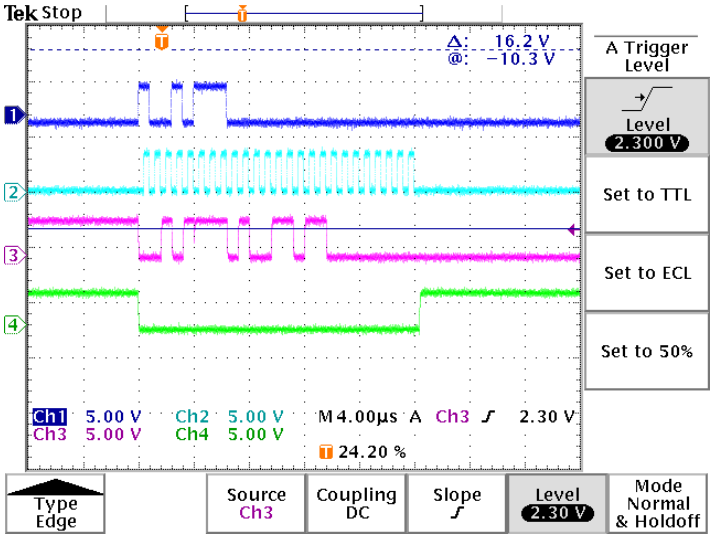


Figura 5.2: Captura de osciloscopio de la conversión de 2V realizada por el ADC.

La conversión en el ADC comienza a partir del décimo pulso del dclk. En este caso, la conversión resultante fue 1001 1011 0000, lo que equivale a 2480 en decimal. Realizando una regla de tres, se obtiene que este valor corresponde a un voltaje de 1.9985V.

La segunda prueba consistió en aplicar 3.3V al canal 0 del ADC para evaluar su respuesta ante un voltaje mayor. En la siguiente figura se muestra la conversión resultante

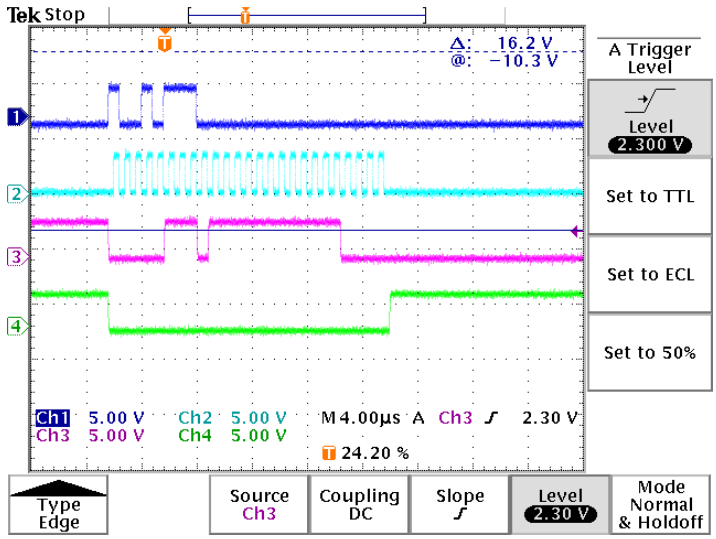


Figura 5.3: Captura de osciloscopio de la conversión de 3.3V realizada por el ADC.

El resultado de la conversión fue 1111 1111 1111, que corresponde a 4095 en decimal. Este valor equivale a un voltaje de 3.3V, confirmando que el ADC realizó correctamente la conversión de la señal analógica al valor digital máximo posible para este canal.

5.1.2. DAC

Para controlar el DAC, se utilizó únicamente la escritura a través del protocolo SPI. Basándonos en el datasheet del DAC, la escritura se realiza en 16 ciclos de sck. Durante los primeros 4 ciclos, se envía el comando de configuración, en el cual se especifica en qué canal se tendrá el resultado de la conversión. En los 12 ciclos restantes, se transmite un valor de voltaje en formato binario de 12 bits al dispositivo para su conversión a señal analógica.

En las Figuras 5.4 y 5.6 se muestran las capturas del osciloscopio, donde se pueden observar tres señales numeradas del 1 al 3. La señal número 1 corresponde a los pulsos de reloj sck, la señal número 2 muestra el comando y el valor del voltaje en formato binario, y la señal número 3 representa el chip select del dispositivo. Todas estas señales fueron generadas por los módulos de diseño del protocolo SPI implementado.

Para verificar que el DAC realizara correctamente la conversión, se asignaron dos valores de voltaje en formato binario al MOSI: 1001 1011 0010, equivalente a 2V, y 1111 1111 1111, equivalente a 3.3V. Posteriormente, se midió el voltaje en el canal asignado utilizando un multímetro, confirmando que las conversiones fueron precisas.

En las siguientes imágenes se muestran las capturas del osciloscopio correspondientes a la escritura del DAC, así como el resultado en el multímetro de la conversión de 2V y 3.3V, respectivamente.

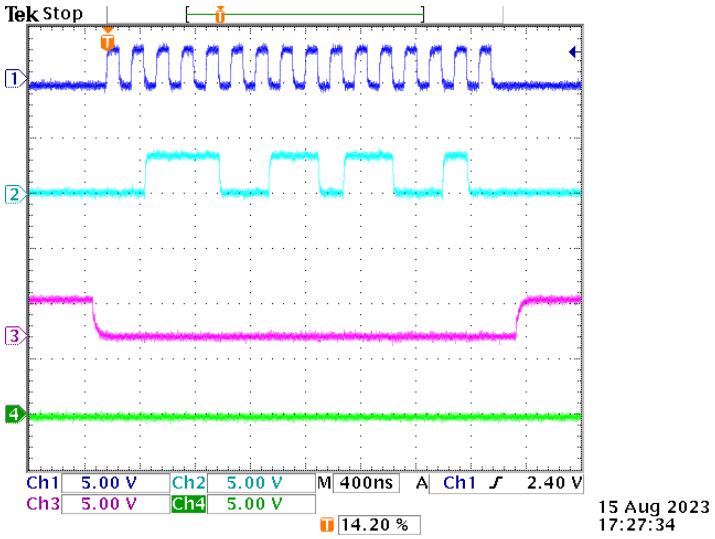


Figura 5.4: Captura de osciloscopio de la escritura de 2V en binario en el DAC.

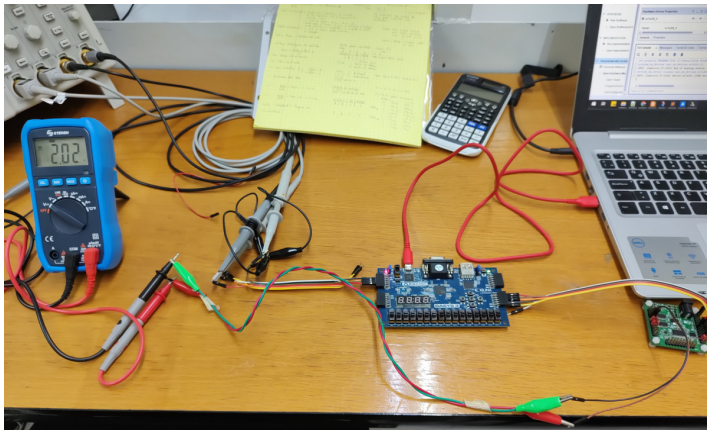


Figura 5.5: Resultado de la conversión de 2V realizada por el DAC.

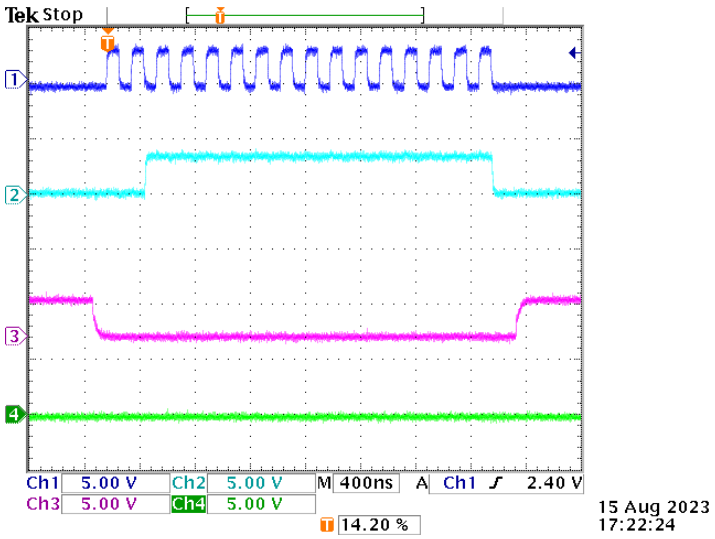


Figura 5.6: Captura de osciloscopio de la escritura de 3.3V en binario en el DAC

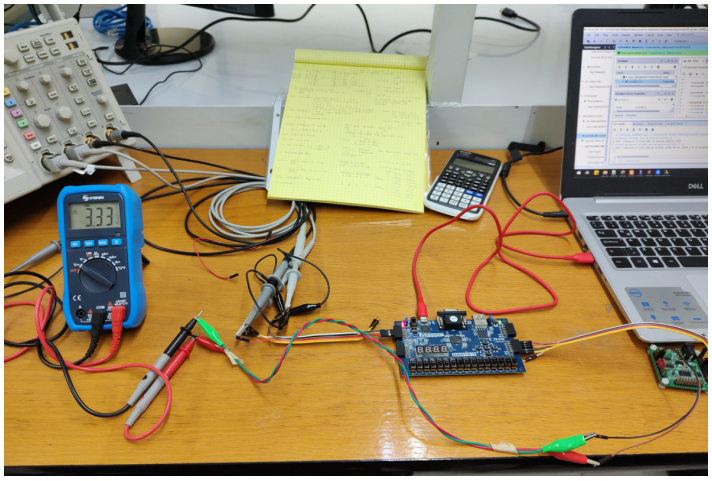


Figura 5.7: Resultado de la conversión de 3.3V realizada por el DAC.

5.2. Resultados de caracterización de resistencias y multiplexores

En esta sección se presentan los resultados combinados de la caracterización de las resistencias y de los multiplexores.

Para la caracterización de las resistencias, cada una de las R_{test} seleccionadas se colocó junto con la resistencia R_{ref} en un protoboard. A través de un código en MATLAB, se adquirieron 10 veces los valores de las caídas de voltaje en R_{ref} correspondientes a cada uno de los voltajes de polarización, posteriormente estos fueron promediados.

En la siguiente imagen se muestra el arreglo descrito.

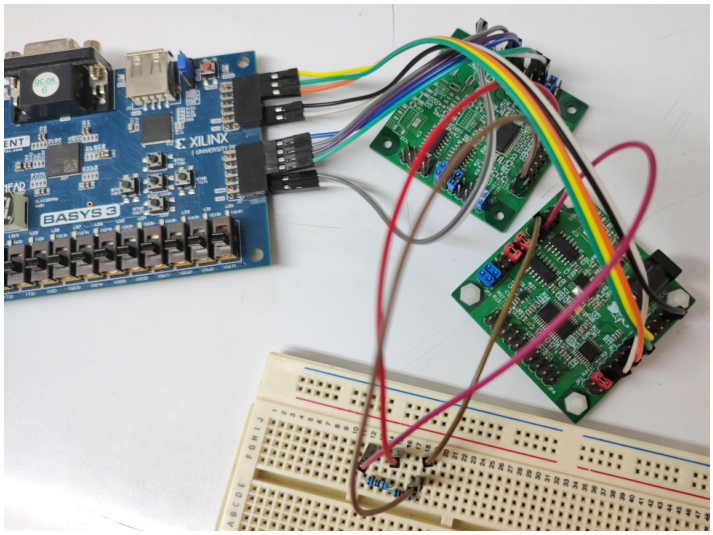


Figura 5.8: Arreglo de elementos para la caracterización de una resistencia.

Para la caracterización de los multiplexores, se colocaron las cuatro R_{test} en forma de matriz en el protoboard, la resistencia de $1K\Omega$ se ubicó en la posición (1,1), la de

$3.3K\Omega$ en la (1,2), la de $5.6K\Omega$ en la (2,1) y la de $10K\Omega$ en la (2,2). Los multiplexores se conectaron para controlar tanto las filas como las columnas de la matriz, y fueron operados de manera manual. La adquisición de los valores de las caídas de voltaje se realizó de manera idéntica a la utilizada en la caracterización de una resistencia, asegurando consistencia en el proceso de medición.

El arreglo de la matriz de resistencias y multiplexores se muestra en la siguiente imagen.

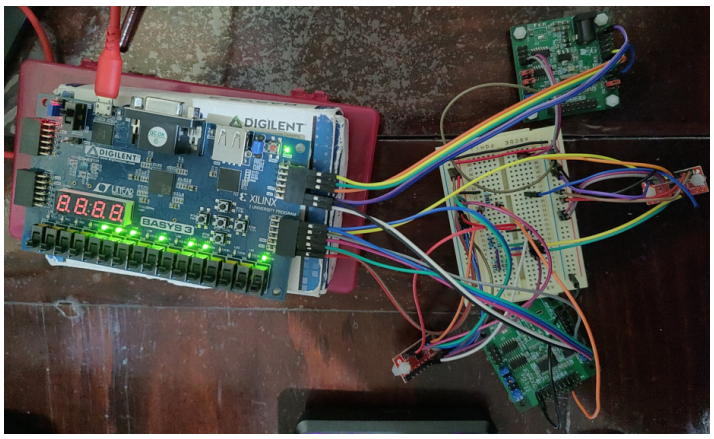


Figura 5.9: Arreglo de elementos para la caracterización de multiplexores.

En las siguientes figuras se presentan las comparaciones de las curvas de los voltajes teóricos con los voltajes recibidos en MATLAB para cada una de las R_{test}, tanto con como sin multiplexores. Es notable que los multiplexores influyen en el resultado final debido a su resistencia interna, ya que cuando no se utilizan, el voltaje teórico y el voltaje recibido son muy similares. Para analizar con mayor detalle el efecto de los multiplexores, nos enfocaremos en los resultados obtenidos al polarizar el circuito con un voltaje de 3V.

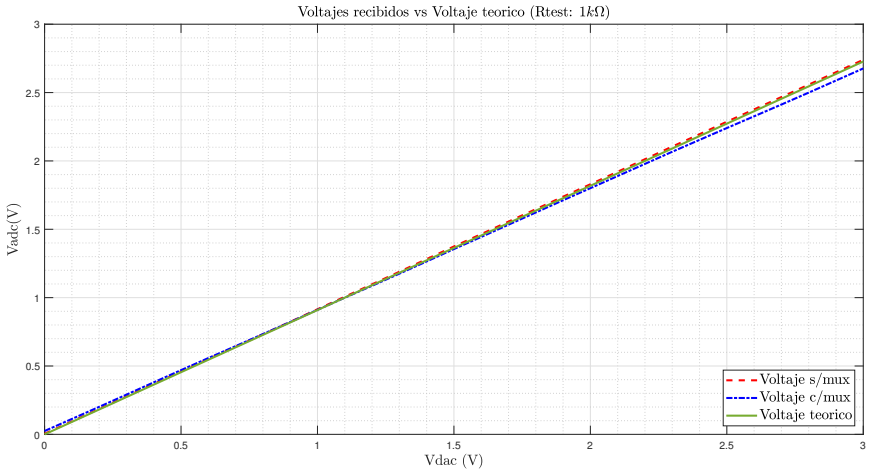


Figura 5.10: Voltajes teóricos vs Voltajes recibidos para $R_{test} = 1K\Omega$.

En la imagen anterior, el voltaje teórico y el voltaje recibido sin multiplexor es de 2.72V, mientras que el voltaje agregando los multiplexores también es de 2.6952V.

Para una resistencia de $3.3K\Omega$, se tiene que el valor de los voltajes teórico y sin multiplexores es de 2.2556V, y el voltaje con multiplexor igual a 2.2375V.

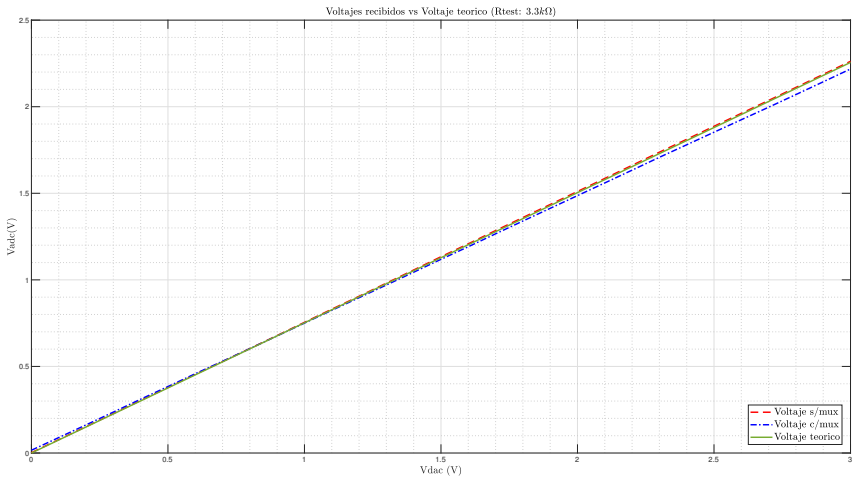


Figura 5.11: Voltajes teóricos vs Voltajes recibidos para $R_{test} = 3.3K\Omega$.

Se tiene que los voltajes teórico y el de una resistencia de $5.6K\Omega$ es 1.92V, cuando se conectan los multiplexores, se registró un voltaje de 1.90V.

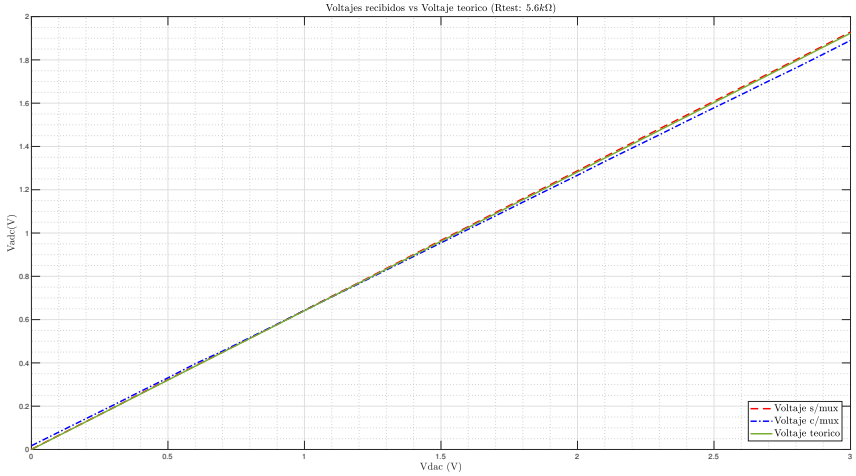


Figura 5.12: Voltajes teóricos vs Voltajes recibidos para $R_{test} = 5.6K\Omega$.

Finalmente, para una resistencia de $10K\Omega$, el voltaje teórico y el voltaje recibido es de 1.5V, y el voltaje con multiplexor de 1.49V.

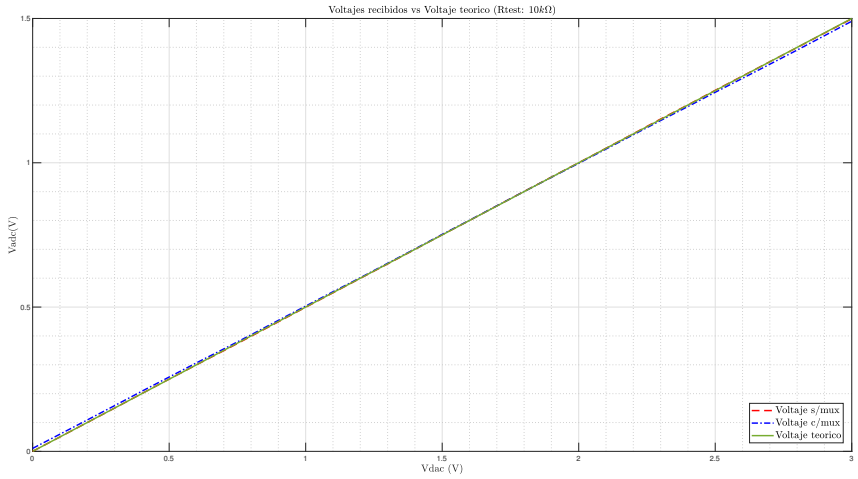


Figura 5.13: Voltajes teóricos vs Voltajes recibidos para $R_{test} = 10K\Omega$.

La resistencia interna de los multiplexores, fue medida con un multímetro, manteniendo el voltaje de polarización del circuito fijo en 3V. En la siguiente tabla se muestran los valores obtenidos de las resistencias internas de los multiplexores.

Tabla 5.1: Resistencia de multiplexores.

R_{test}	$V_{muxrows}$	$V_{muxcols}$	i	$R_{mux_{row}}$	$R_{mux_{col}}$
1KΩ	3.86e-2 V	3.5e-2 V	2.65e-4 A	145.66 Ω	132Ω
3.3KΩ	3.43e-2 V	3.1e-2 V	2.21e-4 A	155.2 Ω	140.27Ω
5.6KΩ	2.83e-2 V	2.48e-2 V	1.89e-4 A	149.73 Ω	131.21Ω
10KΩ	2.25e-2 V	2.11e-2 V	1.49e-4 A	151 Ω	141.6Ω

De acuerdo con el datasheet, las resistencias de cada una de las entradas/salidas Yn varían. Según los datos presentados en la tabla, la resistencia interna de los multiplexores se encuentra en un rango de 130 a 160 Ω.

5.3. Resultados de caracterización de fotorresistencia

Para caracterizar la fotorresistencia, inicialmente se expuso a la iluminación de una lámpara de un salón, pero el valor de la resistencia no lograba estabilizarse. Posteriormente, se intentó medir la fotoresistencia en oscuridad, pero esta superaba los $5M\Omega$, lo que impedía que el multímetro detectara su valor y que esta pudiera ser caracterizada,

debido a que, al ser una resistencia muy grande, la caída de voltaje en el circuito de lectura sería tan pequeña que el ADC no podría convertir. Por lo tanto, se optó por colocar la fotorresistencia en condiciones ideales y controladas, dentro de una caja oscura, y variando la intensidad de un LED de 3W mediante un PWM con distintos duty cycles. De esta forma, se obtuvieron valores de resistencia adecuados para su caracterización.

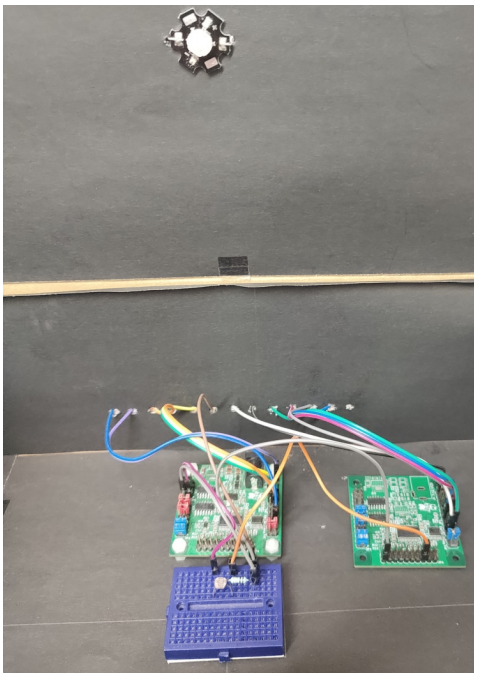


Figura 5.14: Arreglo de fotoresistencia para su caracterización.

Antes de caracterizar la fotorresistencia bajo condiciones ideales, fue necesario definir el voltaje adecuado para alimentar el LED de 3W, ya que el brillo del LED varía en función del voltaje aplicado. Inicialmente, el LED se alimentó con 3V, variando el duty cycle del PWM y midiendo la resistencia de la fotorresistencia en cada caso. Posteriormente, el proceso se repitió alimentando el LED con 3.3V y, finalmente, con 3.5V. Estas mediciones permitieron observar cómo el voltaje de alimentación del LED influye en la respuesta de la fotorresistencia.

En la siguiente tabla se muestran los valores de la fotorresistencias bajo distintos voltajes de polarización y duty cycles.

Tabla 5.2: Valor de fotorresistencias con diferentes duty cycles.

V_{Fuente}	**Duty cycle (%)**	$R_{med}(K\Omega)$
3 V	25	210 - 213.7
	50	104 - 105.7
	75	70 -71
	100	53 - 54
3.3 V	25	24 - 31
	50	15 - 18
	75	10 - 12
	100	8 - 9
3.5 V	25	6 - 7
	50	3.6
	75	2.6
	100	2

Dado que con un voltaje de polarización de 3.3V se obtuvieron valores de resistencia más adecuados para la caracterización de la fotorresistencia, y que esta presentó diferentes valores a distintos duty cycles, se optó por alimentar el LED con ese voltaje. Esto permitió una mejor evaluación del comportamiento de la fotorresistencia bajo condiciones controladas de iluminación.

El circuito de lectura utilizado para la caracterización de la fotorresistencia fue nuevamente una resistencia de $10K\Omega$. El diseño digital empleado para esta caracterización es el mismo que se utilizó en las etapas anteriores.

En la siguiente imagen se muestran las curvas de voltaje de la fotorresistencia a diferentes duty cycles.

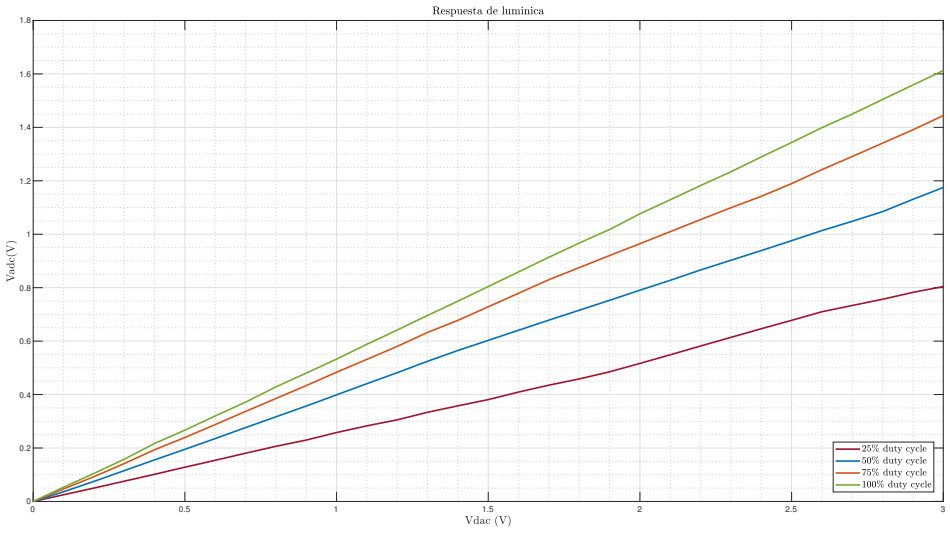


Figura 5.15: Respuesta lumínica.

Usando la ecuación del divisor de voltaje, se determinó que los valores obtenidos de la fotorresistencia coinciden con los presentados en la Tabla 5.2. Además, en la imagen se puede observar que, a mayor iluminación, la fotorresistencia presenta un comportamiento más lineal.

5.4. Resultados de las imágenes obtenidas

De acuerdo al datasheet de la fotorresistencia, su tiempo de respuesta es de 20 ms después de haber sido expuesta a la luz durante 2 horas. Debido a este comportamiento, no se pudo generar una imagen con estos elementos, ya que el tiempo necesario para obtenerla sería demasiado extenso. Por este motivo, se optó por utilizar fototransistores, que ofrecen un tiempo de respuesta mucho más rápido y adecuado para la captura de imágenes en menos tiempo.

Para obtener imágenes utilizando la matriz de 8x8 fototransistores, se propusieron las máscaras mostradas en la Figura 5.16.

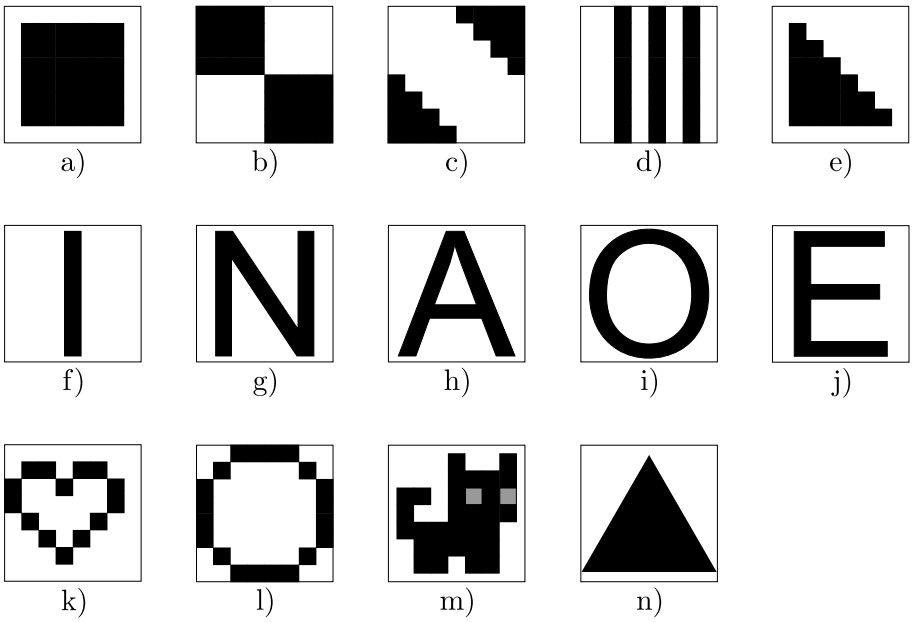


Figura 5.16: Máscaras propuestas.

Cada una de estas máscaras fue colocada encima de la matriz, que fue iluminada con el LED de un celular situado a 20 cm de distancia. Las variaciones de voltaje del circuito de lectura fueron adquiridas y procesadas en MATLAB.

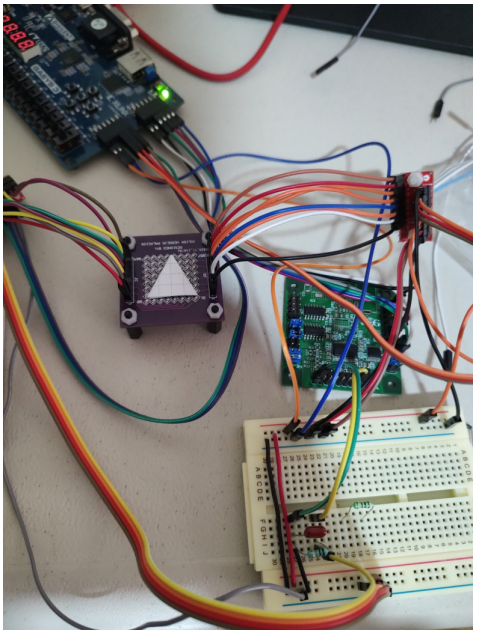


Figura 5.17: Máscara triangular colocada encima de la matriz de fototransistores.

En la Figura 5.18 se muestran las imágenes obtenidas al colocar cada una de las máscaras sobre la matriz de 8×8 fototransistores.

El diseño del sistema de adquisición de datos y la PCB funcionaron con éxito, permitiendo generar imágenes que son muy similares a las formas de las máscaras utilizadas. Esto demuestra la efectividad del sistema para capturar y reproducir las variaciones de luz.

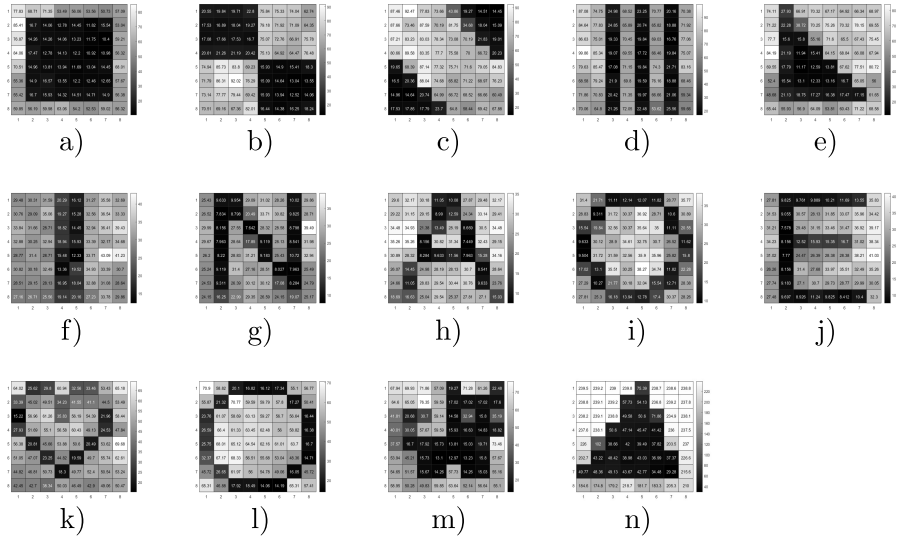


Figura 5.18: Resultados de las máscaras aplicadas a los fototransistores.

Capítulo 6

Conclusiones

- Uno de los principales logros de este trabajo fue el diseño de códigos robustos, modulares y escalables para la implementación del protocolo SPI en Verilog, específicamente para el control de los convertidores analógico-digital (A/D) y digital-analógico (D/A), ADS7841 y MCP4922, respectivamente. La robustez del SPI fue comprobada a través de simulaciones, mediciones con osciloscopio y multímetro, así como por la recepción de datos mediante MATLAB. Estos códigos no solo cumplen con los requisitos del proyecto, sino que además tienen la flexibilidad de ser reutilizados y adaptados a futuras necesidades. Su diseño modular permite modificaciones fáciles y rápidas.

- Se logró diseñar una PCB funcional con una matriz de 8x8 fototransistores, capaz de detectar las variaciones de luz. La PCB no solo demostró ser eficiente en su propósito, sino que también cumplió con todos los requisitos de diseño y fabricación, lo que permitió que fuera producida y ensamblada exitosamente. Este resultado valida tanto el diseño como su implementación práctica, asegurando su utilidad en aplicaciones futuras que requieran la captura de imágenes a través de variaciones lumínicas.

- El módulo de caracterización desarrollado no solo demostró ser eficaz para las pruebas realizadas, sino que también ofrece la flexibilidad necesaria para ser modificado y adaptado para la caracterización de un microbolómetro real. Este enfoque modular permite que el sistema se ajuste a diferentes requisitos y aplicaciones.

- Aunque en este trabajo el uso de una resistencia de referencia y un capacitor fueron suficientes para emplearse como circuito de lectura, no significa que estos sirvan para cualquier tipo de sensor. Es necesario contar con un circuito de lectura específico adaptado a la señal de salida generada por el detector en uso. Esto es esencial para asegurar una captura precisa y eficiente de los datos.

- Para obtener lecturas adecuadas de una matriz de detectores, no es suficiente contar únicamente con convertidores A/D y D/A de alta resolución. Es igualmente crucial utilizar multiplexores que no introduzcan una resistencia excesiva en el circuito. La resistencia añadida por los multiplexores altera las mediciones, afectando la precisión del sistema de adquisición de datos. Por lo tanto, es esencial seleccionar cuidadosamente los componentes para minimizar estas interferencias y obtener lecturas precisas y confiables.

Apéndice A

Códigos

A.1. Códigos en MATLAB

Código A.1: Adquisición de datos

```
%% RS232 Conection
clear; close all; clc;

port = "COM4";
baudrate = 115200;
fpga = serialport(port,baudrate,"Parity","none","Timeout", 30);
flush(fpga);
filename = "voltajes_" + 1 + ".txt";
file = fopen(filename, "w");

n_lecturas = 4;
data = zeros(n_lecturas,2);

for i = 1:n_lecturas
    data(i,:) = read(fpga,2,"uint8");
    volt = (data(i,1)*256 + data(i,2))*(3.3/4095);
    fprintf(file,'%3u %3u %8.6f\n', data(i,:), volt);
end


fclose(file);

clear fpga;

% Default Configurations
    % "Parity","none"
    % "DataBits",8
    % "StopBits",1
    % "FlowControl","none"
    % "ByteOrder","little-endian"
    % "Timeout", 10
```

Código A.2: Adquisición de datos para matriz de fototransistores

```
%% RS232 Conection
clear; close all; clc;

port = "COM4";
baudrate = 115200;
fpga = serialport(port,baudrate,"Parity","none","Timeout", 30);
flush(fpga);
filename = "final" + ".txt";
file = fopen(filename, "w");

n_lecturas = 64;
data = zeros(n_lecturas,2);

for i = 1:n_lecturas
    data(i,:) = read(fpga,2,"uint8");
    volt = (data(i,1)*256 + data(i,2))*(3.3/4095);
    fprintf(file,'%3u %3u %8.6f\n', data(i,:), volt);
end


fclose(file);
```

```
clear fpga;

% Default Configurations
    % "Parity","none"
    % "DataBits",8
    % "StopBits",1
    % "FlowControl","none"
    % "ByteOrder","little-endian"
    % "Timeout", 10
```

Código A.3: Generación de imagen

```
clear; close all; clc;

data = load("final.txt");
volts = data(:,3);
matrix = reshape(volts, [8, 8])';
min_val = 0;
max_val = 3.2;
normalized_matrix = 255 * (matrix - min_val) / (max_val - min_val);
h = heatmap(normalized_matrix);
colormap(gray);
```

A.2. Códigos de Verilog

A.2.1. Debouncer

Código A.4: Debouncer IP.

```
////////////////////////////////////////////////////////////////////////////////
// [Filename]       debouncer.sv
// [Project]        debouncer_ip
// [Author]         Julisa Verdejo
// [Language]       Verilog 2005 [IEEE Std. 1364-2005]
// [Created]        2024.06.22
// [Description]    Debouncer circuit
// [Notes]          Tick output is useful to test FSMs
//                  Level output emulates a Schmitt trigger
//                  Asynchronous active high reset signal
//                  ClkRate: is the FPGA frequency
//                  Baud:    is the number of bits per second
//                  Example:
//                    ClkRate = 100_000_000    ->   100 MHz
//                    Baud    =  10_000_000    ->    10 Mbps
//                    Time    = 1 / Baud       ->   100 ns
//                  The debounce time is:
//                    db_time = (ClkRate / Baud) * (1 / ClkRate) + (1 / ClkRate)
//                            = 110 ns
// [Status]         Stable
////////////////////////////////////////////////////////////////////////////////

module debouncer_ip #(
    parameter ClkRate = 100_000_000,
    parameter Baud    =  10_000_000
) (
    input  clk_i,
    input  rst_i,
    input  sw_i,
    output db_level_o,
    output db_tick_o
);

  // Internal variables
  reg ff1, ff2, ff3, ff4;
  wire ena_cnt, clear_cnt;

  // Run the button through two flip-flops to avoid metastability issues
  always @(posedge clk_i, posedge rst_i) begin
    if (rst_i) begin
      ff1 <= 'd0;
      ff2 <= 'd0;
    end else begin
      ff1 <= sw_i;
      ff2 <= ff1;
    end
  end

  assign clear_cnt = ff1 ^ ff2;

  localparam BaudCounterMax = ClkRate / Baud;
  localparam BaudCounterSize = $clog2(BaudCounterMax);
  reg [BaudCounterSize-1:0] cnt;

  // Counter logic
  always @(posedge clk_i, posedge rst_i) begin
    if (rst_i) begin
      cnt <= 0;
    end else begin
      if (clear_cnt) begin
        cnt <= 0;
      end else if (~ena_cnt) begin
        cnt <= cnt + 1;
      end
    end
  end

  assign ena_cnt = (cnt == BaudCounterMax - 1) ? 1'b1 : 1'b0;

  // Output debounce level
  always @(posedge clk_i, posedge rst_i) begin
    if (rst_i) begin
      ff3 <= 0;
    end else if (ena_cnt) begin
      ff3 <= ff2;
    end
  end

  assign db_level_o = ff3;

  // Output single tick with edge detector
  always @(posedge clk_i, posedge rst_i) begin
    if (rst_i) begin
      ff4 <= 0;
    end else if (ena_cnt) begin
```

```
      ff4 <= ~ff3 & ff2;
    end
  end

  assign db_tick_o = ff4;

endmodule
```

A.2.2. Controlador UART

Código A.5: Generador de la tasa de baudios.

```
////////////////////////////////////////////////////////////////////////////////
// [Filename]       baud_gen.sv
// [Project]        uart_ip
// [Author]         Julisa Verdejo
// [Language]       Verilog 2005 [IEEE Std. 1364-2005]
// [Created]        2024.06.22
// [Description]    Module for generating baud rate for UART communication.
//                  Asynchronous active high reset signal
//                  Equation:
//                           dvsr_i = f_fpga/(baud_rate*16)
// [Notes]          This module supports configurable baud rates, see C++ code.
// [Status]         Stable
////////////////////////////////////////////////////////////////////////////////

module baud_gen (
    input         clk_i,
    input         rst_i,
    input  [10:0] dvsr_i,
    output        tick_o
);

  reg  [10:0] counter_q;
  wire [10:0] counter_d;
  wire counter_done;

  always @(posedge clk_i, posedge rst_i) begin
    if (rst_i) begin
      counter_q <= 11'd0;
    end else begin
      counter_q <= counter_d;
    end
  end

  assign counter_done = (counter_q == dvsr_i - 11'd1) ? 1'b1 : 1'b0;
  assign counter_d = (counter_done) ? 11'd0 : counter_q + 11'd1;

  assign tick_o = counter_done;

endmodule
```

Código A.6: UART RX.

```
////////////////////////////////////////////////////////////////////////////////
// [Filename]       uart_rx.sv
// [Project]        uart_ip
// [Author]         Julisa Verdejo
// [Language]       Verilog 2005 [IEEE Std. 1364-2005]
// [Created]        2024.06.22
// [Description]    UART Receiver Module.
// [Notes]          This code uses an oversamplig of 16.
//                  Asynchronous active high reset signal
//                  The number of stop bits can be set to 1, 1.5, or 2.
//                  This code does not consider the parity bit.
// [Status]         Stable
////////////////////////////////////////////////////////////////////////////////

module uart_rx #(
    parameter WordLength   = 8,
    parameter StopBitTicks = 16
) (
    input            clk_i,
    input            rst_i,
    input            rx_i,
    input            sample_tick_i,
    output reg       rx_done_tick_o,
    output     [7:0] dout_o
);

  localparam [1:0]
    IDLE  = 2'b00,
    START = 2'b01,
    DATA  = 2'b10,
    STOP  = 2'b11;

  reg [1:0] state_reg, state_next;
```

```
  reg [3:0] sample_tick_counter_q, sample_tick_counter_d;
  reg [2:0] data_bit_counter_q,    data_bit_counter_d;
  reg [7:0] data_shift_buffer_q,   data_shift_buffer_d;

  always @(posedge clk_i, posedge rst_i) begin
    if (rst_i) begin
      state_reg            <= IDLE;
      sample_tick_counter_q <= 0;
      data_bit_counter_q    <= 0;
      data_shift_buffer_q   <= 0;
    end else begin
      state_reg <= state_next;
      sample_tick_counter_q <= sample_tick_counter_d;
      data_bit_counter_q    <= data_bit_counter_d;
      data_shift_buffer_q   <= data_shift_buffer_d;
    end
  end

  always @(*) begin
    state_next            = state_reg;
    rx_done_tick_o        = 1'b0;
    sample_tick_counter_d = sample_tick_counter_q;
    data_bit_counter_d    = data_bit_counter_q;
    data_shift_buffer_d   = data_shift_buffer_q;
    case (state_reg)
      IDLE: begin
        if (~rx_i) begin
          state_next            = START;
          sample_tick_counter_d = 0;
        end
      end
      START: begin
        if (sample_tick_i) begin
          if (sample_tick_counter_q == 7) begin
            state_next            = DATA;
            sample_tick_counter_d = 0;
            data_bit_counter_d    = 0;
          end else begin
            sample_tick_counter_d = sample_tick_counter_q + 1;
          end
        end
      end
      DATA: begin
        if (sample_tick_i) begin
          if (sample_tick_counter_q == 15) begin
            sample_tick_counter_d = 0;
            data_shift_buffer_d   = {rx_i, data_shift_buffer_q[7:1]};
            if (data_bit_counter_q == (WordLength - 1)) begin
              state_next = STOP;
            end else begin
              data_bit_counter_d = data_bit_counter_q + 1;
            end
          end else begin
            sample_tick_counter_d = sample_tick_counter_q + 1;
          end
        end
      end
      STOP: begin
        if (sample_tick_i) begin
          if (sample_tick_counter_q == (StopBitTicks - 1)) begin
            state_next = IDLE;
            rx_done_tick_o = 1'b1;
          end else begin
            sample_tick_counter_d = sample_tick_counter_q + 1;
          end
        end
      end
    endcase
  end

  assign dout_o = data_shift_buffer_q;

endmodule
```

Código A.7: UART TX.

```
////////////////////////////////////////////////////////////////////////////////
// [Filename]       uart_tx.sv
// [Project]        uart_ip
// [Author]         Julisa Verdejo
// [Language]       Verilog 2005 [IEEE Std. 1364-2005]
// [Created]        2024.06.22
// [Description]    UART Transmitter Module.
// [Notes]          This code uses an oversamplig of 16.
//                  Asynchronous active high reset signal
//                  The number of stop bits can be set to 1, 1.5, or 2.
//                  This code does not consider the parity bit.
// [Status]         Stable
////////////////////////////////////////////////////////////////////////////////

module uart_tx #(
    parameter WordLength   = 8,
    parameter StopBitTicks = 16
) (
    input            clk_i,
    input            rst_i,
```

```
    input            start_tx_i,
    input            sample_tick_i,
    input      [7:0] din_i,
    output           tx_o,
    output reg       tx_done_tick_o
);

  localparam [1:0]
    IDLE  = 2'b00,
    START = 2'b01,
    DATA  = 2'b10,
    STOP  = 2'b11;

  reg [1:0] state_reg, state_next;
  reg [3:0] sample_tick_counter_q, sample_tick_counter_d;
  reg [2:0] data_bit_counter_q,    data_bit_counter_d;
  reg [7:0] data_shift_buffer_q,   data_shift_buffer_d;
  reg tx_q, tx_d;

  always @(posedge clk_i, posedge rst_i) begin
    if (rst_i) begin
      state_reg            <= IDLE;
      sample_tick_counter_q <= 0;
      data_bit_counter_q    <= 0;
      data_shift_buffer_q   <= 0;
      tx_q                  <= 1'b1;
    end else begin
      state_reg            <= state_next;
      sample_tick_counter_q <= sample_tick_counter_d;
      data_bit_counter_q    <= data_bit_counter_d;
      data_shift_buffer_q   <= data_shift_buffer_d;
      tx_q                  <= tx_d;
    end
  end

  always @(*) begin
    state_next           = state_reg;
    tx_done_tick_o       = 1'b0;
    sample_tick_counter_d = sample_tick_counter_q;
    data_bit_counter_d    = data_bit_counter_q;
    data_shift_buffer_d   = data_shift_buffer_q;
    tx_d                  = tx_q;
    case (state_reg)
      IDLE: begin
        tx_d = 1'b1;
        if (start_tx_i) begin
          state_next            = START;
          sample_tick_counter_d = 0;
          data_shift_buffer_d   = din_i;
        end
      end
      START: begin
        tx_d = 1'b0;
        if (sample_tick_i) begin
          if (sample_tick_counter_q == 15) begin
            state_next            = DATA;
            sample_tick_counter_d = 0;
            data_bit_counter_d    = 0;
          end else begin
            sample_tick_counter_d = sample_tick_counter_q + 1;
          end
        end
      end
      DATA: begin
        tx_d = data_shift_buffer_q[0];
        if (sample_tick_i) begin
          if (sample_tick_counter_q == 15) begin
            sample_tick_counter_d = 0;
            data_shift_buffer_d   = data_shift_buffer_q >> 1;
            if (data_bit_counter_q == (WordLength - 1)) begin
              state_next = STOP;
            end else begin
              data_bit_counter_d = data_bit_counter_q + 1;
            end
          end else begin
            sample_tick_counter_d = sample_tick_counter_q + 1;
          end
        end
      end
      STOP: begin
        tx_d = 1'b1;
        if (sample_tick_i) begin
          if (sample_tick_counter_q == (StopBitTicks - 1)) begin
            state_next     = IDLE;
            tx_done_tick_o = 1'b1;
          end else begin
            sample_tick_counter_d = sample_tick_counter_q + 1;
          end
        end
      end
    endcase
  end

  assign tx_o = tx_q;

endmodule
```

Código A.8: UART IP.

```
////////////////////////////////////////////////////////////////////////////////
// [Filename]       uart_tx.sv
// [Project]        uart_ip
// [Author]         Julisa Verdejo
// [Language]       Verilog 2005 [IEEE Std. 1364-2005]
// [Created]        2024.06.22
// [Description]    UART IP module.
// [Notes]          This code uses an oversamplig of 16.
//                  Asynchronous active high reset signal
//                  The number of stop bits can be set to 1, 1.5, or 2.
//                  This code does not consider the parity bit.
// [Status]         Stable
////////////////////////////////////////////////////////////////////////////////

module uart_ip #(
    parameter WordLength   = 8,
    parameter StopBitTicks = 16
) (
    input         clk_i,
    input         rst_i,
    input  [10:0] dvsr_i,
    input  [7:0]  din_i,
    input         rx_i,
    input         start_tx_i,
    output [7:0]  dout_o,
    output        tx_o,
    output        rx_done_tick_o,
    output        tx_done_tick_o
);

  wire tick;

  baud_gen baud_gen_inst(
    .clk_i(clk_i),
    .rst_i(rst_i),
    .dvsr_i(dvsr_i),
    .tick_o(tick)
  );

  uart_rx #(
    .WordLength  (WordLength),
    .StopBitTicks(StopBitTicks)
  ) uart_rx_inst(
    .clk_i(clk_i),
    .rst_i(rst_i),
    .rx_i(rx_i),
    .sample_tick_i(tick),
    .rx_done_tick_o(rx_done_tick_o),
    .dout_o(dout_o)
  );

  uart_tx #(
    .WordLength  (WordLength),
    .StopBitTicks(StopBitTicks)
  ) uart_tx_inst (
    .clk_i(clk_i),
    .rst_i(rst_i),
    .start_tx_i(start_tx_i),
    .sample_tick_i(tick),
    .din_i(din_i),
    .tx_o(tx_o),
    .tx_done_tick_o(tx_done_tick_o)
  );

endmodule
```

A.2.3. Controlador SPI

A.2.3.1. SPI write

Código A.9: Divisor de frecuencia.

```
// Author: Julisa Verdejo Palacios
// Name: clk_div.v
//
// Description: Contador descendente que genera un timer de 400us

module clk_div #(
  parameter Width = 8
) (
  input             rst_i,
  input             clk_i,
  input             h_i,
  input [Width-1:0] kmax_i,
  output            slow_clk_o
);

  wire [Width-1:0] mux1;
  wire [Width-1:0] mux2_d;
  wire comp;
  reg [Width-1:0] reg_q;

  assign mux1 = (h_i) ? reg_q - 1 : reg_q;
  assign mux2_d = (comp) ? kmax_i : mux1;
  assign comp = ( reg_q == {Width{1'b0}} ) ? 1'b1 : 1'b0;

  always @(posedge clk_i, posedge rst_i) begin
    if (rst_i)
      reg_q <= 0;
    else
      reg_q <= mux2_d;
  end

  assign slow_clk_o = comp;

endmodule
```

Código A.10: Contador de bits.

```
// Author: Julisa Verdejo Palacios
// Name: counter_w.v
//
// Description: Contador descendente utilizado para contar los 25 ciclos de dclk.

module counter_w #(
  parameter Width = 5
) (
  input              rst_i,
  input              clk_i,
  input        [1:0] opc_i,
  output             flag_o
);

  reg [Width-1:0] mux_d, reg_q;

  always @(opc_i, reg_q) begin
    case (opc_i)
      2'b00   : mux_d = 0;
      2'b01   : mux_d = reg_q;
      2'b10   : mux_d = reg_q + 1;
      2'b11   : mux_d = 0;
      default : mux_d = reg_q;
    endcase
  end

  always @(posedge clk_i, posedge rst_i) begin
    if (rst_i)
      reg_q <= 0;
    else
      reg_q <= mux_d;
  end

  assign flag_o = (reg_q == 5'd25) ? 1'b1 : 1'b0;

endmodule
```

Código A.11: Máquina de estados de spi write.

```
// Author: Julisa Verdejo Palacios
// Name: fsm_spiw.v
//
// Description: Maquina de estados utilizada para controlar (escribir) el ADC.

module fsm_spi_write (
  input            rst_i,
  input            clk_i,
  input            strw_i,
  input            slow_clk_i,
  input            flag_i,
  output reg [1:0] opc1_o,
  output reg [1:0] opc2_o,
  output reg       cs_o,
  output reg       dclk_o,
  output reg       hab_o,
  output reg       eow_o
);

  localparam [2:0] s0 = 3'b000, // Reset piso_reg, Reset counter_w, Stop clk_div, Wait strw_i,
                   s1 = 3'b001, // Dummy state
                   s2 = 3'b010, // Load piso_reg, Enable clk_div
                   s3 = 3'b011, // Hold piso_reg, Hold counter_w, dclk up
                   s4 = 3'b100, // Shift piso_reg, Increase counter_w, dclk down
                   s5 = 3'b101, // Hold piso_reg, hold counter_w, dclk down
                   s6 = 3'b110; // Chip select time requirement


  reg [2:0] present_state, next_state;

  always @(strw_i, slow_clk_i, flag_i, present_state) begin
    next_state = present_state;
    opc1_o = 2'b11; opc2_o = 2'b11; cs_o = 1'b1; dclk_o = 1'b0; hab_o = 1'b0; eow_o = 1'b1;
    case (present_state)
      s0 : begin
             opc1_o = 2'b11; opc2_o = 2'b11; cs_o = 1'b1; dclk_o = 1'b0; hab_o = 1'b0; eow_o = 1'b1;
             if (strw_i)
               next_state = s1;
           end

      s1 : begin
             opc1_o = 2'b00; opc2_o = 2'b01; cs_o = 1'b0; dclk_o = 1'b0; hab_o = 1'b0; eow_o = 1'b0;
             next_state = s2;
           end

      s2 : begin
             opc1_o = 2'b01; opc2_o = 2'b01; cs_o = 1'b0; dclk_o = 1'b0; hab_o = 1'b1; eow_o = 1'b0;
             if (slow_clk_i)
               next_state = s3;
           end

      s3 : begin
             opc1_o = 2'b00; opc2_o = 2'b01; cs_o = 1'b0; dclk_o = 1'b1; hab_o = 1'b1; eow_o = 1'b0;
             if (slow_clk_i)
               next_state = s4;
           end

      s4 : begin
             opc1_o = 2'b10; opc2_o = 2'b10; cs_o = 1'b0; dclk_o = 1'b0; hab_o = 1'b1; eow_o = 1'b0;
             next_state = s5;
           end

      s5 : begin
             opc1_o = 2'b00; opc2_o = 2'b01; cs_o = 1'b0; dclk_o = 1'b0; hab_o = 1'b1; eow_o = 1'b0;
             if (slow_clk_i) begin
                if (flag_i) begin
                  next_state = s6;
                end else begin
                  next_state = s3;
                end
             end
           end

      s6 : begin
             opc1_o = 2'b00; opc2_o = 2'b01; cs_o = 1'b1; dclk_o = 1'b0; hab_o = 1'b1; eow_o = 1'b0;
               if(slow_clk_i)
                 next_state = s0;
           end

      default: begin
                 next_state = s0;
               end
    endcase
  end

  always @(posedge clk_i, posedge rst_i) begin
    if (rst_i)
      present_state <= s0;
    else
      present_state <= next_state;
  end

endmodule
```

Código A.12: PISO shift register.

```
// Author: Julisa Verdejo Palacios
// Name: piso_reg.v
//
// Description: Registro con entrada en paralelo y salida en serie con desplazamiento hacia la izquierda.

module piso_reg #(
  parameter Width = 8
) (
  input             rst_i,
  input             clk_i,
  input [Width-1:0] din_i,
  input       [1:0] op_i,
  output            dout_o
);

  reg [Width-1:0] mux_d, reg_q;

  always @(din_i, op_i, reg_q) begin
    case (op_i)
      2'b00   : mux_d = reg_q;
      2'b01   : mux_d = din_i;
      2'b10   : mux_d = {reg_q[Width-2:0],1'b0};
      2'b11   : mux_d = 0;
      default : mux_d = 0;
    endcase
  end

  always @(posedge clk_i, posedge rst_i) begin
    if (rst_i)
      reg_q <= 0;
    else
      reg_q <= mux_d;
  end

  assign dout_o = reg_q[Width-1];

endmodule
```

Código A.13: SPI READ IP.

```
// Author: Julisa Verdejo Palacios
// Name: spi_write.v
//
// Description: Instanciacion de todos los modulos utilizados para la escritura.

module spi_write_ip (
  input        clk_i,
  input        rst_i,
  input        strw_i,
  input  [7:0] cmd_i,
  input  [7:0] kmax_i,
  output       mosi_o,
  output       dclk_o,
  output       cs_o,
  output       eow_o,
  output       slow_clk_o
);

  wire [1:0] opc1, opc2;
  wire slow_clk, flag, hab;

  //localparam  [7:0] cmd_i = 8'b10010111;    // Canal 0
  //localparam  [7:0] kmax_i = 8'd39;         // Periodo dclk = 1600ns

  piso_reg   #(.Width(8))  mod_piso    (.rst_i(rst_i), .clk_i(clk_i), .din_i(cmd_i), .op_i(opc1), .dout_o(mosi_o));
  counter_w  #(.Width(5))  mod_cnt_w   (.rst_i(rst_i), .clk_i(clk_i), .opc_i(opc2), .flag_o(flag));
  clk_div    #(.Width(8))  mod_clkdiv  (.rst_i(rst_i), .clk_i(clk_i), .h_i(hab), .kmax_i(kmax_i), .slow_clk_o(slow_clk));

  fsm_spi_write  mod_fsm_write (
    .rst_i(rst_i),
    .clk_i(clk_i),
    .strw_i(strw_i),
    .slow_clk_i(slow_clk),
    .flag_i(flag),
    .opc1_o(opc1),
    .opc2_o(opc2),
    .cs_o(cs_o),
    .dclk_o(dclk_o),
    .hab_o(hab),
    .eow_o(eow_o)
  );

  assign slow_clk_o = slow_clk;

endmodule
```

A.2.3.2. SPI read

Código A.14: Contador de flancos de reloj de SPI.

```
// Author: Julisa Verdejo Palacios
// Name: counter_r.v
//
// Description: Contador descendente utilizado para contar los pulsos de dclk

module counter_r #(
  parameter Width = 6
) (
  input               rst_i,
  input               clk_i,
  input        [1:0]  opc_i,
  output [Width-1:0]  cnt_o
);

  reg [Width-1:0] mux_d, reg_q;

  always @(opc_i, reg_q) begin
    case (opc_i)
      2'b00 : mux_d = 0;
      2'b01 : mux_d = reg_q;
      2'b10 : mux_d = reg_q + 1;
      2'b11 : mux_d = 0;
      default : mux_d = 0;
    endcase
  end

  always @(posedge clk_i, posedge rst_i) begin
    if (rst_i)
      reg_q <= 0;
    else
      reg_q <= mux_d;
  end

  assign cnt_o = reg_q;

endmodule
```

Código A.15: Máquina de estados de spi read.

```
// Author: Julisa Verdejo Palacios
// Name: fsm_spir.v
//
// Description: Maquina de estados utilizada para guardar los datos recibidos del ADC (conversion).

module fsm_spi_read (
  input             rst_i,
  input             clk_i,
  input             strr_i,
  input             slow_clk_i,
  input      [5:0]  cnt_i,
  output reg [1:0]  opc1_o,
  output reg [1:0]  opc2_o,
  output reg        eor_o,
  output reg        hab_o
);

  localparam [3:0] s0 = 4'b0000, // Wait strr_i, Reset counter_r, Reset sipo_reg
                   s1 = 4'b0001, //
                   s2 = 4'b0010, //
                   s3 = 4'b0011, //
                   s4 = 4'b0100, //
                   s5 = 4'b0101, //
                   s6 = 4'b0110, //
                   s7 = 4'b0111; //

  reg [3:0] next_state, present_state;

  always @(slow_clk_i, cnt_i, strr_i, present_state) begin
    opc1_o = 2'b00; opc2_o = 2'b00; eor_o = 1'b1; hab_o = 1'b0;
    next_state = present_state;
    case (present_state)
      s0 : begin
             opc1_o = 2'b00; opc2_o = 2'b00; eor_o = 1'b1; hab_o = 1'b0;
             if (strr_i)
               next_state = s1;
           end

      s1 : begin
             opc1_o = 2'b01; opc2_o = 2'b01; eor_o = 1'b0; hab_o = 1'b0;
             next_state = s2;
           end

      s2 : begin
             opc1_o = 2'b01; opc2_o = 2'b01; eor_o = 1'b0; hab_o = 1'b0;
             if (slow_clk_i)
               next_state = s3;
           end

      s3 : begin
```

```
            opc1_o = 2'b10; opc2_o = 2'b01; eor_o = 1'b0; hab_o = 1'b0;
            next_state = s4;
          end

      s4 : begin
            opc1_o = 2'b01; opc2_o = 2'b01; eor_o = 1'b0; hab_o = 1'b0;
            if (slow_clk_i) begin
              if (cnt_i == 6'd50) begin
                next_state = s7;
              end else begin
                if (cnt_i >= 6'd18)
                  next_state = s5;
                else
                  next_state = s3;
              end
            end
          end

      s5 : begin
            opc1_o = 2'b10; opc2_o = 2'b10; eor_o = 1'b0; hab_o = 1'b0;
            next_state = s6;
          end

      s6 : begin
            opc1_o = 2'b01; opc2_o = 2'b01; eor_o = 1'b0; hab_o = 1'b0;
            if (slow_clk_i)
              next_state = s3;
          end

      s7 : begin
            opc1_o = 2'b01; opc2_o = 2'b01; eor_o = 1'b0; hab_o = 1'b1;
            next_state = s0;
          end

      default : begin
                  next_state = s0;
                end
    endcase
  end

  always @(posedge clk_i, posedge rst_i) begin
    if (rst_i)
      present_state <= s0;
    else
      present_state <= next_state;
  end

endmodule
```

Código A.16: SIPO shift register.

```
// Author: Julisa Verdejo Palacios
// Name: sipo_reg.v
//
// Description: Registro con entrada en serie y salida en paralelo con desplazamiento a la izquierda.

module sipo_reg #(
  parameter Width = 16
) (
  input               rst_i,
  input               clk_i,
  input               din_i,
  input         [1:0] op_i,
  output  [Width-1:0] dout_o
);

  reg [Width-1:0] reg_q, mux_d;

  always @(din_i, op_i, reg_q) begin
    case (op_i)
      2'b00   : mux_d = 0;
      2'b01   : mux_d = reg_q;
      2'b10   : mux_d = {reg_q[Width-2:0],din_i};
      2'b11   : mux_d = 0;
      default : mux_d = 0;
    endcase
  end

  always @(posedge clk_i, posedge rst_i) begin
    if (rst_i)
      reg_q <= 0;
    else
      reg_q <= mux_d;
  end

  assign dout_o = reg_q;

endmodule
```

Código A.17: PIPO shift register.

```
// Author: Julisa Verdejo Palacios
// Name: pipo_reg.v
//
// Description: Registro con entrada y salida en paralelo con habilitacion.

module pipo_reg #(
  parameter Width = 12
) (
  input                   rst_i,
  input                   clk_i,
  input                   hab_i,
  input       [Width-1:0] din_i,
  output reg  [Width-1:0] dout_o
);

  always @(posedge clk_i, posedge rst_i) begin
    if (rst_i)
      dout_o <= 0;
    else if (hab_i)
      dout_o <= din_i;
  end

endmodule
```

Código A.18: SPI READ IP.

```
// Author: Julisa Verdejo Palacios
// Name: spi_read.v
//
// Description: Instanciacion de todos los modulos utilizados en la lectura del ADC.

module spi_read_ip (
  input         rst_i,
  input         clk_i,
  input         strr_i,
  input         miso_i,
  input         slow_clk_i,
  output [11:0] dout_o,
  output        eor_o
);

  wire [5:0] cnt;
  wire [1:0] opc1, opc2;
  wire [15:0] data;
  wire hab;

  counter_r  #(.Width(6))  mod_cnt_r  (.rst_i(rst_i), .clk_i(clk_i), .opc_i(opc1), .cnt_o(cnt));
  sipo_reg   #(.Width(16)) mod_sipo   (.rst_i(rst_i), .clk_i(clk_i), .din_i(miso_i), .op_i(opc2), .dout_o(data));
  pipo_reg   #(.Width(12)) mod_pipo   (.rst_i(rst_i), .clk_i(clk_i), .hab_i(hab), .din_i(data[15:4]), .dout_o(dout_o));

  fsm_spi_read mod_fsm_spi_read (
    .rst_i(rst_i),
    .clk_i(clk_i),
    .strr_i(strr_i),
    .slow_clk_i(slow_clk_i),
    .cnt_i(cnt),
    .opc1_o(opc1),
    .opc2_o(opc2),
    .eor_o(eor_o),
    .hab_o(hab)
  );

endmodule
```

A.2.3.3. SPI DAC IP

Código A.19: SPI DAC IP.

```
// Author: Julisa Verdejo Palacios
// Name: spi_write.v
//
// Description: Instanciacion de todos los modulos utilizados para la escritura.

module spi_write_dac (
  input         rst_i,
  input         clk_i,
  input         strw_i,
  input   [7:0] kmax_i,
  input  [15:0] din_i,
  output        mosi_o,
  output        sck_o,
  output        cs_o,
  output        eow_o
);

  wire  [1:0] opc1, opc2;
  wire slow_clk, flag, hab;


  piso_reg_dac   #(.Width(16)) mod_piso    (.rst_i(rst_i), .clk_i(clk_i), .din_i(din_i), .op_i(opc1), .dout_o(mosi_o));
  counter_w_dac  #(.Width(5))  mod_cnt_w   (.rst_i(rst_i), .clk_i(clk_i), .opc_i(opc2), .flag_o(flag));
  clk_div_dac    #(.Width(8))  mod_clkdiv  (.rst_i(rst_i), .clk_i(clk_i), .h_i(hab), .kmax_i(kmax_i), .slow_clk_o(slow_clk
      ));

  fsm_spiw_dac  mod_fsm_w (
    .rst_i(rst_i),
    .clk_i(clk_i),
    .strw_i(strw_i),
    .slow_clk_i(slow_clk),
    .flag_i(flag),
    .opc1_o(opc1),
    .opc2_o(opc2),
    .cs_o(cs_o),
    .sck_o(sck_o),
    .hab_o(hab),
    .eow_o(eow_o)
  );

endmodule
```

A.2.3.4. SPI ADC IP

Código A.20: SPI ADC IP.

```
// Author: Julisa Verdejo Palacios
// Name: spi_wr.v
//
// Description: Instanciacion de los modulos escritura y lectura del spi.

module spi_write_read (
  input         rst_i,
  input         clk_i,
  input         strc_i,
  input   [7:0] cmd_i,
  input   [7:0] kmax_i,
  input         miso_i,
  output        mosi_o,
  output [11:0] dout_o,
  output        dclk_o,
  output        cs_o,
  output        eoc_o
);

  wire slow_clk, eow, eor;

  spi_write_ip mod_spiw (
    .rst_i(rst_i),
    .clk_i(clk_i),
    .strw_i(strc_i),
    .cmd_i(cmd_i),
    .kmax_i(kmax_i),
    .mosi_o(mosi_o),
    .dclk_o(dclk_o),
    .cs_o(cs_o),
    .eow_o(eow),
    .slow_clk_o(slow_clk)
  );

  spi_read_ip mod_spir (
    .rst_i(rst_i),
    .clk_i(clk_i),
    .strr_i(strc_i),
    .miso_i(miso_i),
    .slow_clk_i(slow_clk),
    .dout_o(dout_o),
    .eor_o(eor)
  );

  assign eoc_o = eow & eor;

endmodule
```

A.2.3.5. Códigos de caracterización de resistencias y multiplexores

Código A.21: Memoria ROM.

```
// Author: Julisa Verdejo Palacios
// Name: rom_volts.v
//
// Description:


module rom_volts (
  input      [4:0] addr_i,
  output reg [11:0] rom_o
);

  always@(addr_i)
    case(addr_i)
       0 : rom_o = 12'b000000000000; //0.00
       1 : rom_o = 12'b000001111100; //0.10
       2 : rom_o = 12'b000011111000; //0.20
       3 : rom_o = 12'b000101110100; //0.30
       4 : rom_o = 12'b000111110000; //0.40
       5 : rom_o = 12'b001001101100; //0.50
       6 : rom_o = 12'b001011101000; //0.60
       7 : rom_o = 12'b001101100100; //0.70
       8 : rom_o = 12'b001111100000; //0.80
       9 : rom_o = 12'b010001011100; //0.90
      10 : rom_o = 12'b010011011000; //1.00
      11 : rom_o = 12'b010101010100; //1.10
      12 : rom_o = 12'b010111010000; //1.20
      13 : rom_o = 12'b011001001100; //1.30
      14 : rom_o = 12'b011011001000; //1.40
      15 : rom_o = 12'b011101000100; //1.50
      16 : rom_o = 12'b011111000000; //1.60
      17 : rom_o = 12'b100000111100; //1.70
      18 : rom_o = 12'b100010111000; //1.80
      19 : rom_o = 12'b100100110100; //1.90
      20 : rom_o = 12'b100110110000; //2.00
      21 : rom_o = 12'b101000101100; //2.10
      22 : rom_o = 12'b101010101000; //2.20
      23 : rom_o = 12'b101100100100; //2.30
      24 : rom_o = 12'b101110100000; //2.40
      25 : rom_o = 12'b110000011100; //2.50
      26 : rom_o = 12'b110010011000; //2.60
      27 : rom_o = 12'b110100010100; //2.70
      28 : rom_o = 12'b110110010000; //2.80
      29 : rom_o = 12'b111000001100; //2.90
      30 : rom_o = 12'b111010001000; //3.00
      default: rom_o = 12'b000000000000;
    endcase
endmodule
```

Código A.22: Memoria RAM.

```
// Author: Julisa Verdejo Palacios
// Name: ram_volts.v
//
// Description:


module ram_volts #(
  parameter Width = 12
) (
    input              clk_i,
    input              we_i,
    input  [4:0]       addr_i,
    input  [Width-1:0] dinram_i,
    output [Width-1:0] doutram_o
);

  reg [Width-1:0] ram [30:0];

  always @(posedge clk_i) begin
    if (we_i)
      ram[addr_i] <= dinram_i;
    end

  assign doutram_o = ram[addr_i];

endmodule
```

Código A.23: Contador de direcciones de ROM.

```
// Author: Julisa Verdejo Palacios
// Name: counter_volts.v
//
// Description:

module counter_volts #(
  parameter Width = 5
) (
  input              rst_i,
  input              clk_i,
  input        [1:0] opc1_i,
  output [Width-1:0] count_o
);

  reg [Width-1:0] mux_d, reg_q;

  always @(opc1_i, reg_q) begin
    case (opc1_i)
      2'b00   : mux_d = 5'd0;
      2'b01   : mux_d = reg_q;
      2'b10   : mux_d = reg_q + 5'd1;
      //2'b10   : mux_d = reg_q + 12'd819;
      2'b11   : mux_d = 0;
      default : mux_d = 0;
    endcase
  end

  always @(posedge clk_i, posedge rst_i) begin
    if (rst_i)
      reg_q <= 5'd0;
    else
      reg_q <= mux_d;
  end

  assign count_o = reg_q;

endmodule
```

Código A.24: Contador de direcciones de RAM.

```
// Author: Julisa Verdejo Palacios
// Name: counter_w.v
//
// Description: Contador descendente utilizado para contar los 25 ciclos de dclk.

module counter_address #(
  parameter Width = 5
) (
  input              rst_i,
  input              clk_i,
  input  [1:0]       opc2_i,
  output [Width-1:0] count_o,
  output             flag_o
);

  reg [Width-1:0] mux_d, reg_q;

  always @(opc2_i, reg_q) begin
    case (opc2_i)
      2'b00   : mux_d = 5'd0;
      2'b01   : mux_d = reg_q;
      2'b10   : mux_d = reg_q + 5'd1;
      2'b11   : mux_d = 5'd0;
      default : mux_d = 5'd0;
    endcase
  end

  always @(posedge clk_i, posedge rst_i) begin
    if (rst_i)
      reg_q <= 5'd0;
    else
      reg_q <= mux_d;
  end

  assign count_o = reg_q;

  assign flag_o = (reg_q == 5'd31) ? 1'b1 : 1'b0;
  //assign flag_o = (reg_q == 7'd6) ? 1'b1 : 1'b0;

endmodule
```

Código A.25: Multiplexor.

```
// Author: Julisa Verdejo Palacios
// Name: mux_ch.v
//
// Description:

module mux_ch #(
  parameter Width = 8
) (
  input               sel_i,
  input   [Width-1:0] dmsb_i,
  input   [Width-1:0] dlsb_i,
  output  [Width-1:0] data_o
);

  assign data_o = sel_i ? dlsb_i : dmsb_i;
endmodule
```

Código A.26: Máquina de estados de caracterización.

```
// Author: Julisa Verdejo Palacios
// Name: fsm_spiw.v
//
// Description: Maquina de estados utilizada para controlar (escribir) el ADC.

module fsm_dac_adc_tx (
  input            rst_i,
  input            clk_i,
  input            start_i,
  input            eodac_i,
  input            eoadc_i,
  input            eotx_i,
  input            flag_i,
  output reg       stdac_o,
  output reg       stadc_o,
  output reg       stx_o,
  output reg [1:0] opc1_o,
  output reg [1:0] opc2_o,
  output reg       we_o,
  output reg       sel_o,
  output reg       eos_o
);

  localparam [4:0]  s0 = 5'b00000, //
                    s1 = 5'b00001, //
                    s2 = 5'b00010, //
                    s3 = 5'b00011, //
                    s4 = 5'b00100, //
                    s5 = 5'b00101, //
                    s6 = 5'b00110,
                    s7 = 5'b00111,
                    s8 = 5'b01000,
                    s9 = 5'b01001,
                   s10 = 5'b01010,
                   s11 = 5'b01011,
                   s12 = 5'b01100,
                   s13 = 5'b01101,
                   s14 = 5'b01110,
                   s15 = 5'b01111,
                   s16 = 5'b10000; //


  reg [4:0] present_state, next_state;

  always @(start_i, eodac_i, eoadc_i, eotx_i, flag_i, present_state) begin
    next_state = present_state;
    stdac_o = 1'b0; stadc_o = 1'b0; stx_o = 1'b0; opc1_o = 2'b00; opc2_o = 2'b00; we_o = 1'b0; sel_o = 1'b0; eos_o = 1'b1;
    case (present_state)
       s0 : begin
              stdac_o = 1'b0; stadc_o = 1'b0; stx_o = 1'b0; opc1_o = 2'b00; opc2_o = 2'b00; we_o = 1'b0; sel_o = 1'b0;
     eos_o = 1'b1;
              if (start_i)
                next_state = s1;
            end

       s1 : begin
              stdac_o = 1'b1; stadc_o = 1'b0; stx_o = 1'b0; opc1_o = 2'b01; opc2_o = 2'b01; we_o = 1'b0; sel_o = 1'b0;
     eos_o = 1'b0;
              next_state = s2;
            end

       s2 : begin
              stdac_o = 1'b0; stadc_o = 1'b0; stx_o = 1'b0; opc1_o = 2'b01; opc2_o = 2'b01; we_o = 1'b0; sel_o = 1'b0;
     eos_o = 1'b0;
              if (eodac_i)
                next_state = s3;
            end

       s3 : begin
              stdac_o = 1'b0; stadc_o = 1'b0; stx_o = 1'b0; opc1_o = 2'b01; opc2_o = 2'b01; we_o = 1'b0; sel_o = 1'b0;
     eos_o = 1'b0;
              next_state = s4;
            end
```

```
      s4 : begin
             stdac_o = 1'b0; stadc_o = 1'b0; stx_o = 1'b0; opc1_o = 2'b01; opc2_o = 2'b01; we_o = 1'b0; sel_o = 1'b0;
    eos_o = 1'b0;
             next_state = s5;
           end

      s5 : begin
             stdac_o = 1'b0; stadc_o = 1'b1; stx_o = 1'b0; opc1_o = 2'b01; opc2_o = 2'b01; we_o = 1'b0; sel_o = 1'b0;
    eos_o = 1'b0;
             next_state = s6;
           end

      s6 : begin
             stdac_o = 1'b0; stadc_o = 1'b0; stx_o = 1'b0; opc1_o = 2'b01; opc2_o = 2'b01; we_o = 1'b0; sel_o = 1'b0;
    eos_o = 1'b0;
             if (eoadc_i)
               next_state = s7;
           end

      s7 : begin
             stdac_o = 1'b0; stadc_o = 1'b0; stx_o = 1'b0; opc1_o = 2'b01; opc2_o = 2'b01; we_o = 1'b1; sel_o = 1'b0;
    eos_o = 1'b0;
             next_state = s8;
           end

      s8 : begin
             stdac_o = 1'b0; stadc_o = 1'b0; stx_o = 1'b0; opc1_o = 2'b10; opc2_o = 2'b10; we_o = 1'b0; sel_o = 1'b0;
    eos_o = 1'b0;
             next_state = s9;
           end

      s9 : begin
             stdac_o = 1'b0; stadc_o = 1'b0; stx_o = 1'b0; opc1_o = 2'b01; opc2_o = 2'b01; we_o = 1'b0; sel_o = 1'b0;
    eos_o = 1'b0;
             if (flag_i) begin
               next_state = s10;
             end else begin
               next_state = s1;
             end
           end

     s10 : begin
             stdac_o = 1'b0; stadc_o = 1'b0; stx_o = 1'b0; opc1_o = 2'b00; opc2_o = 2'b00; we_o = 1'b0; sel_o = 1'b0;
    eos_o = 1'b0;
             next_state = s11;
           end

     s11 : begin
             stdac_o = 1'b0; stadc_o = 1'b0; stx_o = 1'b1; opc1_o = 2'b01; opc2_o = 2'b01; we_o = 1'b0; sel_o = 1'b0;
    eos_o = 1'b0;
             next_state = s12;
           end

     s12 : begin
             stdac_o = 1'b0; stadc_o = 1'b0; stx_o = 1'b0; opc1_o = 2'b01; opc2_o = 2'b01; we_o = 1'b0; sel_o = 1'b0;
    eos_o = 1'b0;
             if (eotx_i)
               next_state = s13;
           end

     s13 : begin
             stdac_o = 1'b0; stadc_o = 1'b0; stx_o = 1'b1; opc1_o = 2'b01; opc2_o = 2'b01; we_o = 1'b0; sel_o = 1'b1;
    eos_o = 1'b0;
             next_state = s14;
           end

     s14 : begin
             stdac_o = 1'b0; stadc_o = 1'b0; stx_o = 1'b0; opc1_o = 2'b01; opc2_o = 2'b01; we_o = 1'b0; sel_o = 1'b1;
    eos_o = 1'b0;
             if (eotx_i)
               next_state = s15;
           end

     s15 : begin
             stdac_o = 1'b0; stadc_o = 1'b0; stx_o = 1'b0; opc1_o = 2'b01; opc2_o = 2'b10; we_o = 1'b0; sel_o = 1'b0;
    eos_o = 1'b0;
             next_state = s16;
           end

     s16 : begin
             stdac_o = 1'b0; stadc_o = 1'b0; stx_o = 1'b0; opc1_o = 2'b01; opc2_o = 2'b01; we_o = 1'b0; sel_o = 1'b0;
    eos_o = 1'b0;
             if (flag_i) begin
               next_state = s0;
             end else begin
               next_state = s11;
             end
           end

     default: begin
                next_state = s0;
              end
  endcase
end

always @(posedge clk_i, posedge rst_i) begin
  if (rst_i)
    present_state <= s0;
  else
```

```
      present_state <= next_state;
  end

endmodule
```

Código A.27: Módulo de caracterización de una resistencia.

```
// Author: Julisa Verdejo Palacios
// Name: spi_write.v
//
// Description: Instanciacion de todos los modulos utilizados para la escritura.

module dac_adc_tx (
  input   rst_i,
  input   clk_i,
  input   sw_i,
  input   miso_adc_i,
  output  mosi_dac_o,
  output  dclk_o,
  output  cs_dac_o,
  output  mosi_adc_o,
  output  sck_o,
  output  cs_adc_o,
  output  tx_o,
  output  eos_o
);

  wire start;
  wire eodac, eoadc, eotx;
  wire stdac, stadc, stx;
  wire we, sel, flag;
  wire [4:0] addr, count;
  wire [1:0] opc1, opc2;
  wire [11:0] romout, dataram, doutram;
  wire [7:0] dmsb, dlsb, data;


  //Configuraciones debouncer
  localparam fpga_freq = 100_000_000;  // 100 MHz
  localparam db_baud = 1_000;          // 10 ms debounce time

  //Configuraciones DAC
  localparam  [3:0] ctrl = 4'b0011; // DAC-A
  localparam  [7:0] kmax_dac = 8'd7; //160ns

  //Configuraciones ADC
  localparam cmd = 8'b10010111; //Ch0
  localparam  [7:0] kmax_adc = 8'd39; // Periodo dclk = 800ns

  //Configuraciones RS232_TX
  wire [10:0] dvsr = 11'd54;           // 115200 baudrate
  assign dmsb = {4'b0000, doutram[11:8]};
  assign dlsb = doutram[7:0];

  debouncer_ip #(
    .ClkRate(fpga_freq),
    .Baud(db_baud)
  ) debouncer_inst (
    .clk_i(clk_i),
    .rst_i(rst_i),
    .sw_i(sw_i),
    .db_level_o(),
    .db_tick_o(tick)
  );

  fsm_dac_adc_tx  mod_fsm_dac_adc_tx (
    .rst_i(rst_i),
    .clk_i(clk_i),
    .start_i(start),
    .eodac_i(eodac),
    .eoadc_i(eoadc),
    .eotx_i(eotx),
    .flag_i(flag),
    .stdac_o(stdac),
    .stadc_o(stadc),
    .stx_o(stx),
    .opc1_o(opc1),
    .opc2_o(opc2),
    .we_o(we),
    .sel_o(sel),
    .eos_o(eos_o)
  );

  spi_write_dac  mod_spi_write_dac (
    .rst_i(rst_i),
    .clk_i(clk_i),
    .strw_i(stdac),
    .kmax_i(kmax_dac),
    .din_i({ctrl,romout}),
    .mosi_o(mosi_dac_o),
    .sck_o(sck_o),
    .cs_o(cs_dac_o),
    .eow_o(eodac)
  );

  spi_wr_adc  mod_spi_wr_adc (
```

```
    .rst_i(rst_i),
    .clk_i(clk_i),
    .strc_i(stadc),
    .cmd_i(cmd),
    .kmax_i(kmax_adc),
    .miso_i(miso_adc_i),
    .mosi_o(mosi_adc_o),
    .dout_o(dataram),
    .dclk_o(dclk_o),
    .cs_o(cs_adc_o),
    .eoc_o(eoadc)
  );

  uart_ip #(
    .WordLength  (WordLength),
    .StopBitTicks(StopBitTicks)
  ) uart_ip_inst (
    .clk_i(clk_i),
    .rst_i(rst_i),
    .dvsr_i(dvsr),
    .din_i(tx_data),
    .rx_i(rx_i),
    .start_tx_i(stx),
    .dout_o(),
    .tx_o(tx_o),
    .rx_done_tick_o(),
    .tx_done_tick_o(eotx)
  );

  ram_volts       #(.Width(12)) mod_ram_volts (.clk_i(clk_i), .we_i(we), .addr_i(addr), .dinram_i(dataram), .doutram_o(
      doutram));

  mux_ch         #(.Width(8))   mod_mux_ch    (.sel_i(sel), .dmsb_i(dmsb), .dlsb_i(dlsb), .data_o(tx_data));

  counter_volts   #(.Width(5))  mod_cnt_volts (.rst_i(rst_i), .clk_i(clk_i), .opc1_i(opc1), .count_o(count));

  counter_address #(.Width(5))  mod_cnt_addr  (.rst_i(rst_i), .clk_i(clk_i), .opc2_i(opc2), .count_o(addr), .flag_o(flag))
      ;

  rom_volts mod_rom (.addr_i(count), .rom_o(romout));


endmodule
```

A.2.3.6. Códigos de caracterización de una fotoresistencia

Código A.28: Módulo PWM con diferentes duty cycles.

```
// Author: Julisa Verdejo Palacios
// Name: counter_w.v
//
// Description: Contador descendente utilizado para contar los 25 ciclos de dclk.

module counter_pwm #(
  parameter Width = 17
) (
  input         rst_i,
  input         clk_i,
  input  [2:0]  opc_i,
  output        led_o
);

  reg [Width-1:0] mux_d;
  reg [Width-1:0] reg_q = 0;

  always @(opc_i, reg_q) begin
    case (opc_i)
      3'b000   : mux_d = 17'd0;
      3'b001   : mux_d = (reg_q < 25000) ? 1:0;
      3'b010   : mux_d = (reg_q < 50000) ? 1:0;
      3'b011   : mux_d = (reg_q < 75000) ? 1:0;
      3'b100   : mux_d = 17'd1;
      default : mux_d = 17'd0;
    endcase
  end

  always @(posedge clk_i, posedge rst_i) begin
    if (rst_i)
      reg_q <= 17'd0;
    else
        if (reg_q < 100000) begin
          reg_q <= reg_q + 1;
        end else begin
          reg_q <= 17'd0;
        end
  end

  assign led_o = mux_d;

endmodule
```

A.2.3.7. Códigos del sistema de adquisición para generar una imagen

Código A.29: Contador de carrera libre.

```
module counter_ip #(
  parameter Width = 8
) (
  input              clk_i,
  input              rst_i,
  input              ena_i,
  output [Width-1:0] q_o
);

  reg  [Width-1:0] reg_q;
  wire [Width-1:0] sum_d;

  assign sum_d = reg_q + 1;

  always @(posedge clk_i, posedge rst_i) begin
    if (rst_i) begin
      reg_q <= 0;
    end else if (ena_i) begin
      reg_q <= sum_d;
    end
  end

  assign q_o = reg_q;

endmodule
```

Código A.30: Contador programable.

```
// Author: Julisa Verdejo Palacios
// Name: clk_div.v
//
// Description: Contador descendente que genera un timer de 400us

module count_time_mux #(
  parameter Width = 26
) (
  input             rst_i,
  input             clk_i,
  input             h_i,
  input [Width-1:0] kmax_i,
  output            slow_clk_o
);

  wire [Width-1:0] mux1;
  wire [Width-1:0] mux2_d;
  wire comp;
  reg [Width-1:0] reg_q;

  assign mux1 = (h_i) ? reg_q - 1 : reg_q;
  assign mux2_d = (comp) ? kmax_i : mux1;
  assign comp = ( reg_q == {Width{1'b0}} ) ? 1'b1 : 1'b0;

  always @(posedge clk_i, posedge rst_i) begin
    if (rst_i)
      reg_q <= 0;
    else
      reg_q <= mux2_d;
  end

  assign slow_clk_o = comp;

endmodule
```

Código A.31: Máquina de estados para matriz de fototransistores.

```
// Author: Julisa Verdejo Palacios
// Name: .v
//
// Description:

module fsm_pixel(
  input             rst_i,
  input             clk_i,
  input             tick_i,
  input             eospi_i,
  input             tx_done_i,
  input  [2:0]      cnt_row_i,
  input  [2:0]      cnt_col_i,
  input  [5:0]      cnt_ram_i,
  input  [7:0]      cnt_i,
  output reg        st_spi_o,
  output reg        stx_o,
  output reg        ena_cnt_row_o,
  output reg        ena_cnt_col_o,
  output reg        ena_cnt_ram_o,
```

```
  output reg        ena_cnt_o,
  output reg        we_o,
  output reg        sel_o,
  output reg        eos_o
);

  localparam [4:0]  s0 = 5'b00000,
                    s1 = 5'b00001,
                    s2 = 5'b00010,
                    s3 = 5'b00011,
                    s4 = 5'b00100,
                    s5 = 5'b00101,
                    s6 = 5'b00110,
                    s7 = 5'b00111,
                    s8 = 5'b01000,
                    s9 = 5'b01001,
                   s10 = 5'b01010,
                   s11 = 5'b01011,
                   s12 = 5'b01100,
                   s13 = 5'b01101,
                   s14 = 5'b01110,
                   s15 = 5'b01111,
                   s16 = 5'b10000,
                   s17 = 5'b10001;

  reg [4:0] next_state, present_state;

  always @(*) begin
    st_spi_o = 1'b0 ; stx_o = 1'b0; ena_cnt_row_o = 1'b0; ena_cnt_col_o = 1'b0; ena_cnt_ram_o = 1'b0; ena_cnt_o = 1'b0;
     we_o = 1'b0; sel_o = 1'b0; eos_o = 1'b1;
    next_state = present_state;
    case(present_state)
       s0: begin
             st_spi_o = 1'b0; stx_o = 1'b0; ena_cnt_row_o = 1'b0; ena_cnt_col_o = 1'b0; ena_cnt_ram_o = 1'b0; ena_cnt_o =
    1'b0; we_o = 1'b0; sel_o = 1'b0; eos_o = 1'b1;
             if (tick_i)
               next_state = s1;
           end

       s1: begin
             st_spi_o = 1'b0; stx_o = 1'b0; ena_cnt_row_o = 1'b0; ena_cnt_col_o = 1'b0; ena_cnt_ram_o = 1'b0; ena_cnt_o =
    1'b1; we_o = 1'b0; sel_o = 1'b0; eos_o = 1'b0;
            if (cnt_i)
              next_state = s2;
           end

       s2: begin
             st_spi_o = 1'b0; stx_o = 1'b0; ena_cnt_row_o = 1'b0; ena_cnt_col_o = 1'b0; ena_cnt_ram_o = 1'b0; ena_cnt_o =
    1'b0; we_o = 1'b0; sel_o = 1'b0; eos_o = 1'b0;
             next_state = s3;
           end

       s3: begin
             st_spi_o = 1'b0; stx_o = 1'b0; ena_cnt_row_o = 1'b0; ena_cnt_col_o = 1'b0; ena_cnt_ram_o = 1'b0; ena_cnt_o =
    1'b0; we_o = 1'b0; sel_o = 1'b0; eos_o = 1'b0;
             next_state = s4;
           end

       s4: begin
             st_spi_o = 1'b1; stx_o = 1'b0; ena_cnt_row_o = 1'b0; ena_cnt_col_o = 1'b0; ena_cnt_ram_o = 1'b0; ena_cnt_o =
    1'b0; we_o = 1'b0; sel_o = 1'b0; eos_o = 1'b0;
             next_state = s5;
           end

       s5: begin
             st_spi_o = 1'b0; stx_o = 1'b0; ena_cnt_row_o = 1'b0; ena_cnt_col_o = 1'b0; ena_cnt_ram_o = 1'b0; ena_cnt_o =
    1'b0; we_o = 1'b0; sel_o = 1'b0; eos_o = 1'b0;
             if (eospi_i)
               next_state = s6;
           end

       s6: begin
             st_spi_o = 1'b0; stx_o = 1'b0; ena_cnt_row_o = 1'b0; ena_cnt_col_o = 1'b0; ena_cnt_ram_o = 1'b0; ena_cnt_o =
    1'b0; we_o = 1'b1; sel_o = 1'b0; eos_o = 1'b0;
             next_state = s7;
           end

       s7: begin
             st_spi_o = 1'b0; stx_o = 1'b0; ena_cnt_row_o = 1'b0; ena_cnt_col_o = 1'b1; ena_cnt_ram_o = 1'b1; ena_cnt_o =
    1'b0; we_o = 1'b0; sel_o = 1'b0; eos_o = 1'b0;
             next_state = s8;
           end

       s8: begin
             st_spi_o = 1'b0; stx_o = 1'b0; ena_cnt_row_o = 1'b0; ena_cnt_col_o = 1'b0; ena_cnt_ram_o = 1'b0; ena_cnt_o =
    1'b0; we_o = 1'b0; sel_o = 1'b0; eos_o = 1'b0;
             if (cnt_col_i == 0)
               next_state = s9;
             else
               next_state = s1;
           end

       s9: begin
             st_spi_o = 1'b0; stx_o = 1'b0; ena_cnt_row_o = 1'b1; ena_cnt_col_o = 1'b0; ena_cnt_ram_o = 1'b0; ena_cnt_o =
    1'b0; we_o = 1'b0; sel_o = 1'b0; eos_o = 1'b0;
             next_state = s10;
           end

       s10: begin
```

```
            st_spi_o = 1'b0; stx_o = 1'b0; ena_cnt_row_o = 1'b0; ena_cnt_col_o = 1'b0; ena_cnt_ram_o = 1'b0; ena_cnt_o =
    1'b0; we_o = 1'b0; sel_o = 1'b0; eos_o = 1'b0;
            if (cnt_row_i == 0)
              next_state = s11;
            else
              next_state = s1;
          end

       s11: begin
            st_spi_o = 1'b0; stx_o = 1'b1; ena_cnt_row_o = 1'b0; ena_cnt_col_o = 1'b0; ena_cnt_ram_o = 1'b0; ena_cnt_o =
    1'b0; we_o = 1'b0; sel_o = 1'b0; eos_o = 1'b0;
            next_state = s12;
          end

       s12: begin
            st_spi_o = 1'b0; stx_o = 1'b0; ena_cnt_row_o = 1'b0; ena_cnt_col_o = 1'b0; ena_cnt_ram_o = 1'b0; ena_cnt_o =
    1'b0; we_o = 1'b0; sel_o = 1'b0; eos_o = 1'b0;
            if (tx_done_i)
              next_state = s13;
          end

       s13: begin
            st_spi_o = 1'b0; stx_o = 1'b0; ena_cnt_row_o = 1'b0; ena_cnt_col_o = 1'b0; ena_cnt_ram_o = 1'b0; ena_cnt_o =
    1'b0; we_o = 1'b0; sel_o = 1'b1; eos_o = 1'b0;
            next_state = s14;
          end

       s14: begin
            st_spi_o = 1'b0; stx_o = 1'b1; ena_cnt_row_o = 1'b0; ena_cnt_col_o = 1'b0; ena_cnt_ram_o = 1'b0; ena_cnt_o =
    1'b0; we_o = 1'b0; sel_o = 1'b1; eos_o = 1'b0;
            next_state = s15;
          end

       s15: begin
            st_spi_o = 1'b0; stx_o = 1'b0; ena_cnt_row_o = 1'b0; ena_cnt_col_o = 1'b0; ena_cnt_ram_o = 1'b0; ena_cnt_o =
    1'b0; we_o = 1'b0; sel_o = 1'b1; eos_o = 1'b0;
            if (tx_done_i)
              next_state = s16;
          end

       s16: begin
            st_spi_o = 1'b0; stx_o = 1'b0; ena_cnt_row_o = 1'b0; ena_cnt_col_o = 1'b0; ena_cnt_ram_o = 1'b1; ena_cnt_o =
    1'b0; we_o = 1'b0; sel_o = 1'b0; eos_o = 1'b0;
            next_state = s17;
          end

       s17: begin
            st_spi_o = 1'b0; stx_o = 1'b0; ena_cnt_row_o = 1'b0; ena_cnt_col_o = 1'b0; ena_cnt_ram_o = 1'b0; ena_cnt_o =
    1'b0; we_o = 1'b0; sel_o = 1'b0; eos_o = 1'b0;
            if (cnt_ram_i == 0)
              next_state = s0;
            else
              next_state = s11;
          end

      default:  begin
                  next_state = s0;
                end
    endcase
  end

  always @(posedge clk_i, posedge rst_i) begin
    if (rst_i)
      present_state <= s0;
    else
      present_state <= next_state;
  end

endmodule
```

Código A.32: Módulo principal de la matriz de pixeles.

```
////////////////////////////////////////////////////////////////////////////////
// [Filename]       uart_tx.sv
// [Project]        top
// [Author]         Julisa Verdejo
// [Language]       Verilog 2005 [IEEE Std. 1364-2005]
// [Created]        2024.06.22
// [Description]    Basys 3 UART IP testing code.
// [Notes]          This code uses an oversamplig of 16.
// [Status]         Stable
////////////////////////////////////////////////////////////////////////////////

module top #(
    parameter WordLength   = 8,
    parameter StopBitTicks = 16
) (
    input          clk_i,
    input          rst_i,
    input          sw_i,
    input          rx_i,
    input          miso_i,
    output         mosi_o,
    output         dclk_o,
    output         cs_o,
    //output [7:0]   dout_o,
```

```
    output         tx_o,
    output  [2:0]  row_o,
    output  [2:0]  col_o,
    output         eos_o
);

  wire [10:0] dvsr = 11'd54;           // 115200 baudrate
  localparam fpga_freq = 100_000_000;  // 100 MHz
  localparam db_baud = 1_000;          // 10 ms debounce time

  localparam  [7:0] cmd_i = 8'b10010111;    // Canal 0
  localparam  [7:0] kmax_i = 8'd39;         // Periodo de 800ns | kmax = (t*F_FPGA - 1)/2
  localparam [25:0] delay_mux_i = 26'd99999; //1ms

  wire tick;
  wire tx_done;
  wire start_tx;
  wire ena_cnt_row, ena_cnt_col, ena_cnt_ram, ena_cnt;
  wire [2:0] cnt_row, cnt_col;
  wire cnt;
  wire [5:0] cnt_ram;
  wire [7:0] tx_data;
  wire [15:0] ram_out;
  wire sel;
  wire st_spi;
  wire [15:0] doutspi;
  wire [11:0] dspi;
  wire eospi;
  wire we;

  assign doutspi = {4'b0000,dspi};
  assign row_o = cnt_row;
  assign col_o = cnt_col;

  uart_ip #(
    .WordLength  (WordLength),
    .StopBitTicks(StopBitTicks)
  ) uart_ip_inst (
    .clk_i(clk_i),
    .rst_i(rst_i),
    .dvsr_i(dvsr),
    .din_i(tx_data),
    .rx_i(rx_i),
    .start_tx_i(start_tx),
    .dout_o(),
    .tx_o(tx_o),
    .rx_done_tick_o(),
    .tx_done_tick_o(tx_done)
  );

  debouncer_ip #(
    .ClkRate(fpga_freq),
    .Baud(db_baud)
  ) debouncer_inst (
    .clk_i(clk_i),
    .rst_i(rst_i),
    .sw_i(sw_i),
    .db_level_o(),
    .db_tick_o(tick)
  );

  counter_ip #(
    .Width(6)
  ) counter_ip_ram (
    .clk_i(clk_i),
    .rst_i(rst_i),
    .ena_i(ena_cnt_ram),
    .q_o(cnt_ram)
  );

  count_time_mux #(
    .Width(26)
  ) mod_delay_mux (
    .rst_i(rst_i),
    .clk_i(clk_i),
    .h_i(ena_cnt),
    .kmax_i(delay_mux_i),
    .slow_clk_o(cnt)
  );

  counter_ip #(
    .Width(3)
  ) counter_ip_row (
    .clk_i(clk_i),
    .rst_i(rst_i),
    .ena_i(ena_cnt_row),
    .q_o(cnt_row)
  );

  counter_ip #(
    .Width(3)
  ) counter_ip_col (
    .clk_i(clk_i),
    .rst_i(rst_i),
    .ena_i(ena_cnt_col),
    .q_o(cnt_col)
  );

  ram_ip #(
```

```
    .Width(16)
  ) ram_ip_inst (
    .clk_i(clk_i),
    .we_i(we),
    .addr_i(cnt_ram),
    .dinram_i(doutspi),
    .doutram_o(ram_out)
  );

  fsm_pixel fsm_pixel_inst (
    .rst_i(rst_i),
    .clk_i(clk_i),
    .tick_i(tick),
    .eospi_i(eospi),
    .tx_done_i(tx_done),
    .cnt_row_i(cnt_row),
    .cnt_col_i(cnt_col),
    .cnt_ram_i(cnt_ram),
    .cnt_i(cnt),
    .st_spi_o(st_spi),
    .stx_o(start_tx),
    .ena_cnt_row_o(ena_cnt_row),
    .ena_cnt_col_o(ena_cnt_col),
    .ena_cnt_ram_o(ena_cnt_ram),
    .ena_cnt_o(ena_cnt),
    .we_o(we),
    .sel_o(sel),
    .eos_o(eos_o)
  );

  mux_ip #(
  .Width(8)
) mux_ip_inst (
    .in1_i(ram_out[15:8]),
    .in2_i(ram_out[7:0]),
    .sel_i(sel),
    .mux_o(tx_data)
);

  spi_write_read spi_w_r_inst (
  .rst_i(rst_i),
  .clk_i(clk_i),
  .strc_i(st_spi),
  .cmd_i(cmd_i),
  .kmax_i(kmax_i),
  .miso_i(miso_i),
  .mosi_o(mosi_o),
  .dout_o(dspi),
  .dclk_o(dclk_o),
  .cs_o(cs_o),
  .eoc_o(eospi)
);

endmodule
```

Bibliografía

[1] B. N. S. Yii, N. Ahmad, M. H. A. Wahab, W. M. Jubadi, C. Uttraphan, and S. Z. S. Idrus, "Integration of home automation and security system controller with fpga implementation," *Annals of Emerging Technologies in Computing*, vol. 7, pp. 1–10, Oct. 2023.

[2] J. A. Lenero-Bardallo, R. de la Rosa-Vidal, R. Padial-Allue, J. Ceballos-Caceres, A. Rodriguez-Vazquez, and J. Bernabeu-Wittel, "A customizable thermographic imaging system for medical image acquisition and processing," *IEEE Sensors Journal*, vol. 22, pp. 16730–16741, Sept. 2022.

[3] X. She, H. Lu, Q. Liu, P. Xie, and Q. Xia, "Dermatological infrared thermal imaging with human-machine interaction image diagnostics interface using densenet," *Journal of Radiation Research and Applied Sciences*, vol. 17, p. 100826, Mar. 2024.

[4] N. Le Ba, S. Oh, D. Sylvester, and T. T.-H. Kim, "A 256 pixel 21.6 uw infrared gesture recognition processor for smart devices," *Microelectronics Journal*, vol. 86, pp. 49–56, Apr. 2019.

[5] K. Minoglou, N. Nelms, A. Ciapponi, H. Weber, S. Wittig, B. Leone, and P. Crouzet, "Infrared image sensor developments supported by the european space agency," *Infrared Physics and Technology*, vol. 96, pp. 351–360, Jan. 2019.

[6] G. Parihar, S. Saha, and L. I. Giri, "Application of infrared thermography for irrigation scheduling of horticulture plants," *Smart Agricultural Technology*, vol. 1, p. 100021, Dec. 2021.

[7] R. Chang, *Quimica.* McGraw-Hill, 11th ed., 2013.

[8] J. Vincent, *Fundamentals of Infrared and Visible Detector Operation and Testing.* Wiley, 2nd ed., 2016.

[9] T. Hollands, *Thermal Radiation Fundamentals.* Begell House, 2004.

[10] A. Beiser, *Concepts of Modern Physics.* McGraw-Hill, 6th ed., 2003.

[11] F. Sears, *Fisica Universitaria con Fisica Moderna Volumen 2.* Addison-Wesley, 12th ed., 2009.

[12] S. Ling, *Física Universitaria Volumen 3.* Rice University, 2021.

[13] D. A. B. Mora, "Diseño de circuitos integrados de lectura paramicrobolómetros," Master's thesis, INAOE, 2013.

[14] A. Rogalski, *Infrared Detectors.* CRC Press, 2nd ed., 2011.

[15] R. Jiménez-Zavala, "Desarrollo de micro-bolómetrosno enfriados basados en silicio ygermanio polimorfo," Master's thesis, INAOE, 2013.

[16] M. Ghimire, "Remote sensing and image analysis and in environmental and studies," *N.A*, Abril 2021.

[17] M. Švantner, V. Lang, J. Skála, T. Kohlschütter, M. Honner, L. Muzika, and E. Kosová, "Statistical study on human temperature measurement by infrared thermography," *Sensors*, vol. 22, p. 8395, Nov. 2022.

[18] D. Wziątek-Kuczmik, A. Świątkowski, A. Cholewka, A. Mrowiec, I. Niedzielska, and A. Stanek, "Thermal imaging of the tongue surface as a predictive method in the diagnosis of type 2 diabetes mellitus," *Sensors*, vol. 24, p. 2447, Apr. 2024.

[19] A. Rocha, S. I. Lopes, and C. Abreu, "A cost-effective infrared thermographic system for diabetic foot screening," vol. 15, pp. 106–111, Oct. 2022.

[20] H. Farhat, "An infrared thermal image acquisition system for intra-vehicle applications," in *2011 11th International Conference on ITS Telecommunications*, pp. 426–430, IEEE, Aug. 2011.

[21] T. Sosnowski, "Processing of the image from infrared focal plane array using fpga-based system," in *Mixed Design of Integrated Circuits and Systems*, IEEE, 2010.

[22] F. Forsberg, "Cmos-integrated si/sige quantum-well infrared microbolometer focal plane arrays manufactured with very large-scale heterogeneous 3d integration," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 21, pp. 30–40, July 2015.

[23] P. Vybornov, "Short pulse power measurements by uncooled bolometers," *IEEE Photonics Technology Letters*, vol. 32, pp. 51–54, Jan. 2020.

[24] D. Xie, "Design of a micro uncooled infrared imaging system based on vox irfpa," in *Advanced Sensor Systems and Applications VII* (T. Liu, S. Jiang, and R. Landgraf, eds.), vol. 10025, p. 1002516, SPIE, Oct. 2016.

[25] S. Feng, "The system design based on long-wave uncooled infrared detector," *Journal of Physics: Conference Series*, vol. 1633, p. 012015, Sept. 2020.

[26] R. Bayareh, "Development of a thermographic image instrument using the raspberry pi embedded system for the study of the diabetic foot," in *2018 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, vol. 55, pp. 1–6, IEEE, May 2018.

[27] F. Javaid, M. Rashid, and S. Khan, "A configurable high resolution digital pixel readout integrated circuit with on-chip image processing," *Computers And Electrical Engineering*, vol. 86, p. 106720, Sept. 2020.

[28] S. Fusetto, A. Aprile, P. Malcovati, and E. Bonizzoni, "Readout ic architectures and strategies for uncooled micro-bolometers infrared focal plane arrays: A review," *Sensors*, vol. 23, p. 2727, Mar. 2023.

[29] R. Bhan, R. Saxena, C. Jalwani, and S. Lomash, "Uncooled infrared microbolometer arrays and their characterisation techniques," *Defence Science Journal*, vol. 59, pp. 580–589, Nov. 2009.

[30] H. Budzier, *Thermal Infrared Sensors, Theory, Optimisation and Practice*. John Wiley and Sons, 2011.

[31] J. H. Lau, C. Lee, C. S. Premachandran, and Y. Aibin, eds., *Advanced MEMS packaging*. McGraw-Hill's AccessEngineering, New York: McGraw-Hill, 2010. Print version c2010. - Includes bibliographical references and index. - Description based on cover image and table of contents, viewed on February 08, 2010.

[32] C. Liu, J. Fu, Y. Hou, Q. Zhou, and D. Chen, "A self-calibration method of microbolometer with vacuum package," *IEEE Sensors Journal*, vol. 20, pp. 8570–8575, Aug. 2020.

[33] H. Wei, Z. Tong, S. Qin, and S. Yan, "A novel test method for micro-bolometer thermal parameters," Sept. 2015.

[34] J. Hernandez, J. Rangel-Magdaleno, R. Jimenez, M. Moreno, A. Torres, A. Ponce, D. Ferrusca, and J. Castro-Ramos, "An automated v-i acquisition system for microbolometer array with fpga-based drive," May 2021.

[35] M. Moreno, D. Ferrusca, J. Rangel, J. Castro, J. Hernandez, R. Jimenez, A. Torres, A. Ponce, A. Morales, R. Gonzalez-Cruz, E. Mota-Galvan, and G. Amador-Perez, "Towards an infrared camera based on polymorphous silicon-germanium microbolometer arrays," pp. 1–4, July 2022.

[36] P. P. Chu, *FPGA prototyping by systemverilog examples*. Hoboken, NJ, USA: Wiley, 2018. Includes bibliographical references and index.

[37] P. P. Chu, *FPGA prototyping by Verilog examples*. Hoboken, N.J: J. Wiley And Sons, 2008. Includes bibliographical references (pages 485-486) and index.

[38] W. Kester, ed., *Data conversion handbook*. Analog Devices series, Amsterdam: Elsevier Newnes, [nachdr.] ed., 2007. Includes bibliographical references and index.