UNIVERSIDAD CATÓLICA DE LA SANTÍSIMA CONCEPCIÓN

FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

Habilitación y caracterización de sistema para procesamiento digital ultra rápido aplicado a señales en coincidencia

Sebastián Ignacio Aedo Hermosilla

Informe de Habilitación Profesional para optar al título de:
Ingeniero Civil Eléctrico

Profesor Patrocinante:
Dr. Jaime Cariñe Catrileo

Comisión:
Dr. Ricardo Bustos.
Dr. Daniel Martínez.

Concepción, Octubre de 2023

UNIVERSIDAD CATÓLICA DE LA SANTISIMA CONCEPCION
Facultad de Ingeniería
Departamento de Ingeniería Eléctrica

Profesor Patrocinante:
Dr. Jaime Cariñe Catrileo

Habilitación y caracterización de sistema para procesamiento digital ultra rápido aplicado a señales en coincidencia

Sebastián Ignacio Aedo Hermosilla

Informe de Habilitación Profesional
para optar al Título de

Ingeniero Civil Eléctrico

Concepción, Octubre de 2023

Resumen

Los contadores de fotones en coincidencia desempeñan un papel fundamental en las aplicaciones de la óptica cuántica. Debido al creciente nivel de complejidad en los experimentos de óptica cuántica, se ha vuelto cada vez más importante contar con herramientas precisas que permitan el conteo de coincidencias en múltiples canales a alta velocidad.

En este trabajo, se ha diseñado un circuito de detección de coincidencias electrónicas utilizando un field-programmable gate array (FPGA, por sus siglas en inglés), específicamente con la placa ZedBoard Zynq-7000, que permite trabajar a velocidades de hasta 1GSpS. Este sistema cuenta con cuatro canales operativos, que registran la separación temporal entre eventos de detección provenientes de detectores de fotones individuales. Además, la detección en coincidencia se realiza emparejando estos canales en pares.

Se logró diseñar varias ventanas de coincidencia con una serie de optimizaciones en una placa de desarrollo del FPGA disponible. La ventana de coincidencia más estrecha tiene una duración de 1 ns mientras que la más ancha alcanza los 16 ns. Para verificar el tamaño de todas las ventanas de coincidencia disponibles en este sistema, se utilizó un generador de funciones que simulaba la señal proveniente de detectores de fotones individuales. Esta verificación se hizo para todas las combinaciones de canales en pares posibles.

La creación de un software de monitoreo y control del sistema, habilita la visualización en tiempo real de las mediciones de coincidencias para múltiples ventanas de manera simultánea. Además, permite ajustar retardos de los pulsos generados por el generador de funciones. La comunicación entre este programa y la FPGA se llevó a cabo mediante el uso de UART, facilitando el intercambio de información.

A los alumnos del pasado, presente y futuro de la carrera de Ingeniería Civil Eléctrica de la U.C.S.C.

Agradecimientos

Quiero expresar mi sincero agradecimiento al Instituto Milenio de Investigación en Óptica (MIRO) por brindarme la invaluable oportunidad de utilizar sus instalaciones de laboratorio y equipos para llevar a cabo las investigaciones necesarias para esta tesis. También, mi más profundo agradecimiento al profesor Jaime Cariñe, cuya orientación y apoyo fueron fundamentales para llevar a cabo este proyecto. Su dedicación y paciencia en momentos desafiantes fueron de total ayuda.

Además, quiero agradecer al Fondo Nacional de Desarrollo Científico y Tecnológico (Fondecyt Nº 11201348) por su financiamiento para este proyecto.

Por último, no puedo pasar por alto agradecer a mi familia y amigos, quienes han sido un pilar inquebrantable a lo largo de mi trayecto universitario. Su apoyo constante y aliento incondicional fueron esenciales en este viaje académico. Estoy profundamente agradecido por contar con su respaldo.

Tabla de Contenidos

Capítulo 1. Introducción

1.1. Introducción General

El estudio de la luz y su comportamiento a nivel de fotones individuales es de gran importancia, en el campo de la física cuántica, la óptica cuántica y la ingeniería [1]. Para comprender y caracterizar adecuadamente los fenómenos cuánticos relacionados con la luz, se requiere una medición precisa del tiempo y el análisis de la correlación entre eventos. En este contexto, los contadores de fotones en coincidencia se destacan como una herramienta fundamental utilizada en prácticamente todas las aplicaciones de óptica cuántica [2], [3]. Estos dispositivos desempeñan un papel crucial, ya que permiten medir la correlación entre detecciones de fotones individuales utilizando procesamiento digital sobre las señales digitales provenientes de detectores de fotones individuales (Single Photon Detector, SPD).

La detección del fotón desempeña un papel fundamental en la realización de experimentos donde se emplean fotones, los cuales son generados simultáneamente pero pueden seguir caminos o distancias diferentes. La capacidad de detectar estas coincidencias es fundamental para aplicaciones como criptografía cuántica, teleportación de estados cuánticos y potenciales aplicaciones en computación cuántica, donde los fotones se utilizan como unidades de almacenamiento y procesamiento de datos [4], [5], [6].

Un contador de fotones en coincidencia mide la correlación temporal entre dos o más eventos detectados por diferentes detectores de fotones. Su función principal es determinar si los eventos detectados ocurren simultáneamente o dentro de tiempo muy corto, que se conoce como ventana de coincidencia (τ). Por lo anterior, se requiere un procesamiento digital ultra rápido para evaluar las coincidencias de fotones detectados dentro de τ. Estos dispositivos electrónicos empleados en la medición de pulsos en coincidencia, a partir de detectores de fotones individuales, desempeñan un papel fundamental en diversas aplicaciones relacionadas con la seguridad, especialmente en el ámbito de la criptografía cuántica. Sin embargo, el problema asociado a estos dispositivos es que son de alto costo, con un valor aproximado de 13,000 USD, y carecen de flexibilidad, ya que su programación de funcionamiento no puede ser modificada para adaptarse a diferentes esquemas de medición.

1.2. Estado del arte

Para evaluar las soluciones ante los problemas mencionados, los FPGAs son un recurso que permite flexibilidad y procesamiento de alta velocidad en señales digitales. En el diseño de sistemas de contadores de coincidencia basados en FPGA, se ha puesto énfasis en reducir el intervalo de tiempo la ventana de coincidencia. Esto se debe a que, en un experimento, los detectores de fotones pueden generar detecciones falsas [7] debido a factores como el ruido térmico, señales de fondo o imperfecciones en el sistema óptico. Estas detecciones falsas pueden ocurrir dentro del intervalo de tiempo de la ventana de coincidencia. El problema con estas detecciones falsas es que pueden dar lugar a cuentas en coincidencia accidentales, lo que provocaría un registro de una cuenta coincidente entre dos detectores. Por lo tanto, la reducción de la anchura de la ventana de coincidencia disminuye el número de coincidencias accidentales, mejorando la relación señal-ruido.

Para abordar este problema, se ha utilizado una arquitectura secuencial en los sistemas de contadores de coincidencia. A diferencia de arquitecturas no secuenciales [8], esta arquitectura requiere un reloj interno de alta velocidad para mejorar la resolución de τ, limitando el ancho de la ventana de coincidencia a unos pocos nanosegundos [9], [10]. Esto permite reducir la probabilidad de contar detecciones falsas y mejorar la precisión de las mediciones de coincidencia en el sistema. Por otro lado, se han propuesto arquitecturas de bajo costo basadas en compuertas lógicas para la detección de coincidencias, lo que permite reducir aún más el intervalo de tiempo τ [8]. Además, se ha logrado mejorar la resolución de los contadores de coincidencia a subnanosegundos mediante el uso de circuitos externos [11].

1.3. Motivación

En este contexto, el FPGA ZedBoard Zynq-7000 se presenta como un dispositivo versátil y potente para la implementación de un contador de coincidencia de fotones debido a su capacidad de programación y recursos avanzados [12]. Estas características permiten diseñar un sistema eficiente y personalizado para la detección y análisis de coincidencias de fotones en tiempo real, abriendo nuevas posibilidades en la investigación y aplicaciones de fotónica cuántica.

Gracias a la flexibilidad y potencia de este lenguaje de programación, se pueden crear interfaces de usuario intuitivas y funcionales que permiten una interacción fluida con el sistema de detección de coincidencias.

Por lo anterior, en este trabajo se propone la implementación de un contador de coincidencia de fotones de bajo costo y alto rendimiento que puedan detectar múltiples entradas en un FPGA Zynq-7000, permitiendo controlar y monitorear la detección de pulsos, provenientes de los fotones, a una velocidad que puede llegar a los 1GSpS.

1.4. Objetivos

Se proponen los siguientes objetivos:

1.4.1 Objetivo general

Habilitar Hardware en FPGA Zynq-7000 para detección de señales coincidentes, ampliando coincidencias pares entre 4 canales.

1.4.2 Objetivos específicos

- Desarrollar sistema detector de coincidencias en FPGA Zynq-7000.
- Desarrollar Software interfaz para contador de coincidencia.
- Caracterizar el sistema utilizando generador de funciones.

1.5. Alcances y Limitaciones

Para el desarrollo de este proyecto se utilizará una FPGA Zynq-7000 disponible en el Laboratorio de Optoelectrónica de la Universidad Católica de la Santísima Concepción (UCSC). Entre los cuatro canales, se medirán las coincidencias en pares de canales. Por ejemplo, se llevarán a cabo mediciones de coincidencias entre el canal 0 y el canal 1, así como entre el canal 0 y el canal 2, y así sucesivamente totalizando 6 pares de canales.

La caracterización de coincidencias triples y/o cuádruples no podrá llevarse a cabo debido a la carencia de equipos en el laboratorio.

Como limitación no se tiene acceso a experimentos de óptica cuántica funcionando por lo que la caracterización se realizará utilizando generador de funciones con señales similares a la que produciría a un detector de fotón individual comercial.

1.6. Metodología

Para llevar a cabo este proyecto es necesario diseñar la arquitectura en el FPGA Zynq-7000, que se basará en varios módulos fundamentales. Estos módulos incluyen el ISERDES2 para gestionar las cuentas coincidentes a alta velocidad, el módulo delayRAM para controlar el retardo de las señales entrantes, el módulo DTC para convertir la señal digital en información temporal, módulos de cuentas simples y dobles para llevar un registro de los pulsos entrantes, y el módulo CPU, que servirá como unidad central de procesamiento.

Por otro lado, se requerirá el diseño de una interfaz para el monitoreo y control del sistema utilizando Visual C#. Esta interfaz incluirá un entorno gráfico que permitirá visualizar el comportamiento de las señales que ingresan a la FPGA y controlar las ventanas de coincidencia. La comunicación entre esta interfaz y la FPGA se llevará a cabo mediante la interfaz UART.

Además, se realizará el montaje en una caja metálica del FPGA conectado a un Breakout mediante el puerto FMC. En este Breakout se soldarán los conectores SMA, los cuales representarán los cuatro canales del sistema.

Finalmente, se llevará a cabo una evaluación del sistema a través de una caracterización utilizando un generador de funciones para verificar y asegurar el tamaño de las ventanas de coincidencia y el correcto funcionamiento del sistema en general.

Capítulo 2. Marco teórico

2.1. Single-Photon detector

Los single-photon detectors (SPD) son utilizados para detectar fotones individuales. Son los dispositivos más sensibles para la detección de la luz débil [13].

Los detectores convencionales de fotón individual se basan en fotomultiplicadores y fotodiodos de avalancha y se utilizan en una amplia gama de aplicaciones de conteo de fotones únicos correlacionados en el tiempo [14].

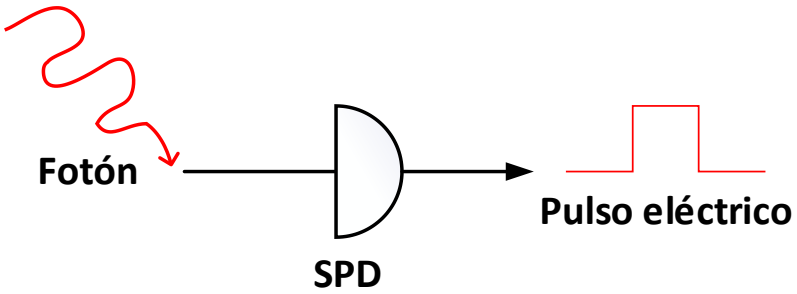


Fig. 2.1 Esquemático representativo de un SPD

Para transformar el fotón en un pulso eléctrico como en la figura 2.1 debe pasar por varios procesos, aunque el mecanismo exacto puede variar según el tipo de SPD. Sin embargo, el proceso general se presenta de la siguiente manera [15]:

1) Absorción del fotón: Cuando un fotón llega al detector, interactúa con el material fotosensible excitando electrones o creando pares electrón-hueco, según el tipo de detector.
2) Amplificación interna (en algunos SPDs): En ciertos SPDs, como APDs y SPADs, los fotones interactúan con una estructura de multiplicación interna, lo que puede provocar una avalancha de electrones y amplificar la señal inicialmente débil.
3) Generación de una corriente eléctrica: La interacción de los electrones excitados con el material sensible genera una corriente eléctrica que se correlaciona con el número de fotones detectados y se traduce en un pulso eléctrico.

4) Amplificación de la señal (en algunos SPDs): En algunos casos, la corriente eléctrica generada en la etapa anterior puede ser amplificada aún más para mejorar la sensibilidad y la relación señal-ruido del SPD.
5) Registro y procesamiento del pulso: La corriente eléctrica generada se procesa mediante circuitos electrónicos del SPD, con posibles amplificaciones, discriminación de eventos y medición de tiempo, adaptándose a las necesidades de la aplicación específica.

Un ejemplo de SPD disponible en el laboratorio de Óptica Cuántica de la Universidad de Concepción es el ID210 que se muestra en la figura 2.2, el cual puede detectar fotones con una probabilidad mayor al 25% a 1550 nm. Además, ofrece retardos ajustables, una duración de apertura ajustable de 0.5 ns a 25 ns y un tiempo muerto ajustable de hasta 100 μs [16].

Fig. 2.2 Single Photon Detector ID210 [16]

Las señales digitales que genera este detector tienen la siguiente forma:

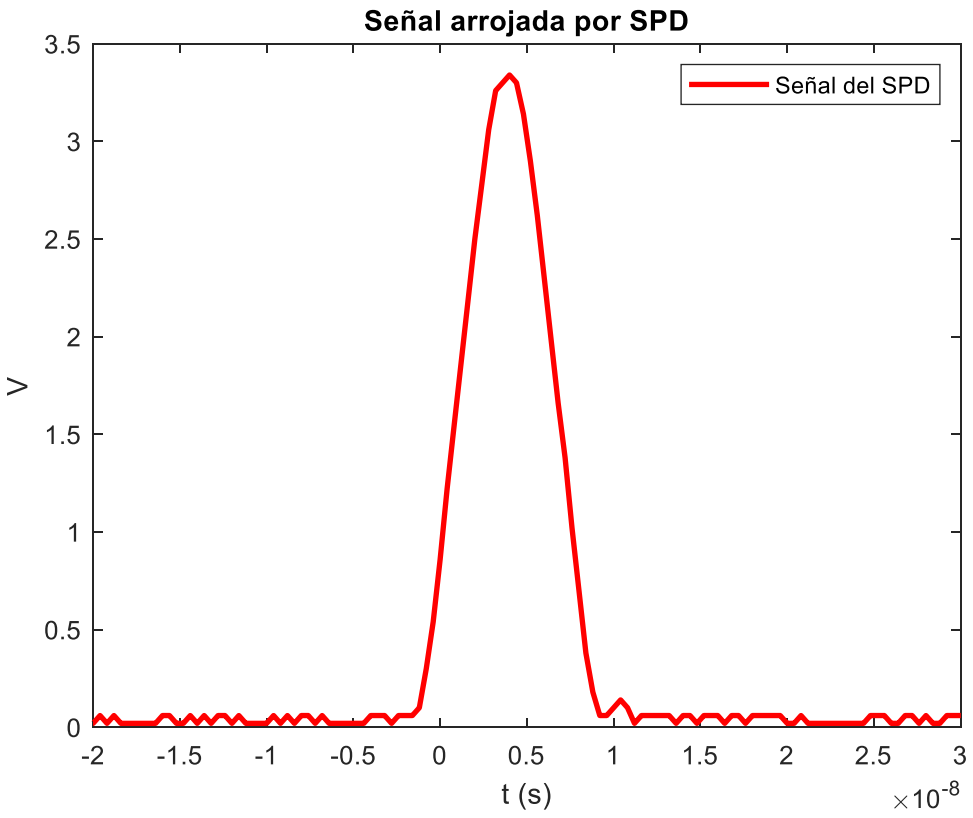


Fig. 2.3 Señal que arroja el ID210

La señal representada en la figura 2.3 tiene una amplitud de 3.34 V y un ancho de 10.4 ns. Esta señal se puede simular utilizando un generador de funciones de la marca Tektronix AFG3021B.

2.2. FMC Breakout

La placa FMC LPC Breakout es un adaptador pasivo que no requiere corriente eléctrica para que funcione. Su propósito principal es proporcionar acceso a todas las señales de los conectores de bajo número de pines (LPC) compatibles con el estándar ANSI/VITA 57.1 del FPGA Mezzanine Card (FMC). Todos los pines de las filas C, D, G y H del conector están dirigidos a un conjunto separado de almohadillas en la parte superior e inferior. Estas almohadillas están dispuestas en un paso de 1.27 mm para facilitar la conexión de sondas o cables soldados adicionales. Para tareas de medición especiales o para su uso como una placa portadora independiente, los pines de alimentación del conector están enrutados a través de vías adicionales en la parte posterior del módulo. Además, se pueden implementar circuitos eléctricos simples en el área de prototipado, que es un conjunto de vías no conectadas con un paso de 2.54 mm. La tarjeta de mezzanine FPGA tiene un factor de forma de grado comercial (ancho único, 78.80 mm x 69 mm) y está diseñada para refrigeración por aire [17].

La placa FMC Breakout se puede utilizar en sistemas de contador de coincidencias de fotones ya que permite conectar fácilmente el módulo FMC de alta velocidad que contienen circuitos de adquisición de datos para la captura de señales de fotones. La razón principal para utilizar la placa Breakout junto al FPGA es que simplifica la conexión del módulo FMC al FPGA y proporciona una fuente de alimentación y un reloj estable, lo que permite a los diseñadores centrarse en la implementación en lugar de preocuparse por la complejidad de las conexiones. En la figura 2.4, se muestra la placa FMC Breakout.

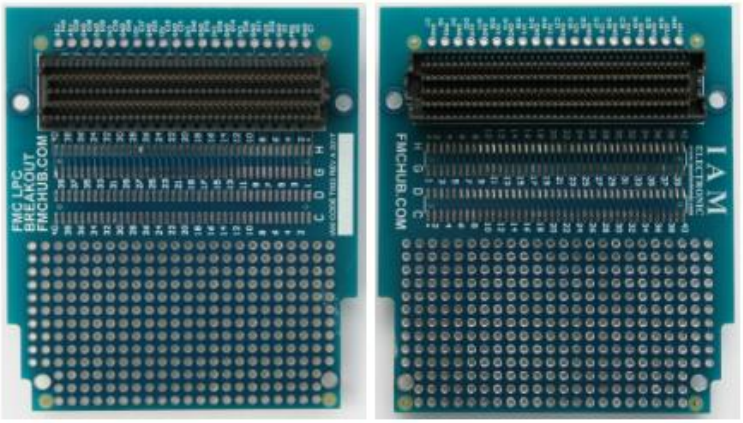


Fig. 2.4 Placa FMC Breakout [17]

2.3. FPGA: Field Programmable Gate Array

Un Field Programmable Gate Array (FPGA) es un dispositivo electrónico versátil y altamente configurable que se utiliza para la implementación de diseños digitales personalizados [18]. En contraste con los circuitos integrados convencionales, cuya funcionalidad está predefinida, un FPGA se destaca por su capacidad de ser programado para realizar múltiples tareas simultáneamente y procesar grandes volúmenes de datos en tiempo real. Gracias a su diseño flexible y su capacidad de reconfiguración, el FPGA resulta altamente idóneo para aplicaciones que demandan una complejidad y velocidad de procesamiento elevadas.

La principal función de un FPGA es permitir la implementación de lógica programable, ofreciendo a los usuarios finales la capacidad de crear nuevos dispositivos hardware. Los FPGA están compuestos por una matriz de bloques de lógica programable dispuestos en una red de interconexión también programable. En los bordes del FPGA se encuentran bloques de entrada/salida (I/O Block) diseñados para conectar las señales de la matriz con el entorno externo. Fue esta combinación de innovaciones la que impulsó el desarrollo de la industria de los FPGA. La Figura 2.5 ilustra la arquitectura básica de un FPGA.

Al principio, los FPGA consistían únicamente en bloques CLB (Configurable Logic Blocks) y I/O Blocks. Sin embargo, a medida que evolucionaron, se introdujeron bloques con funciones específicas, como los DCM (Digital Clock Manager) y los BRAM (Block RAM). Esto fue posible gracias a la capacidad de integrar cada vez más componentes en un área determinada. Esta evolución ha facilitado en gran medida el diseño de sistemas complejos, ya que ahora se pueden aprovechar funcionalidades especializadas directamente en el FPGA, sin necesidad de implementarlas de forma externa [19].

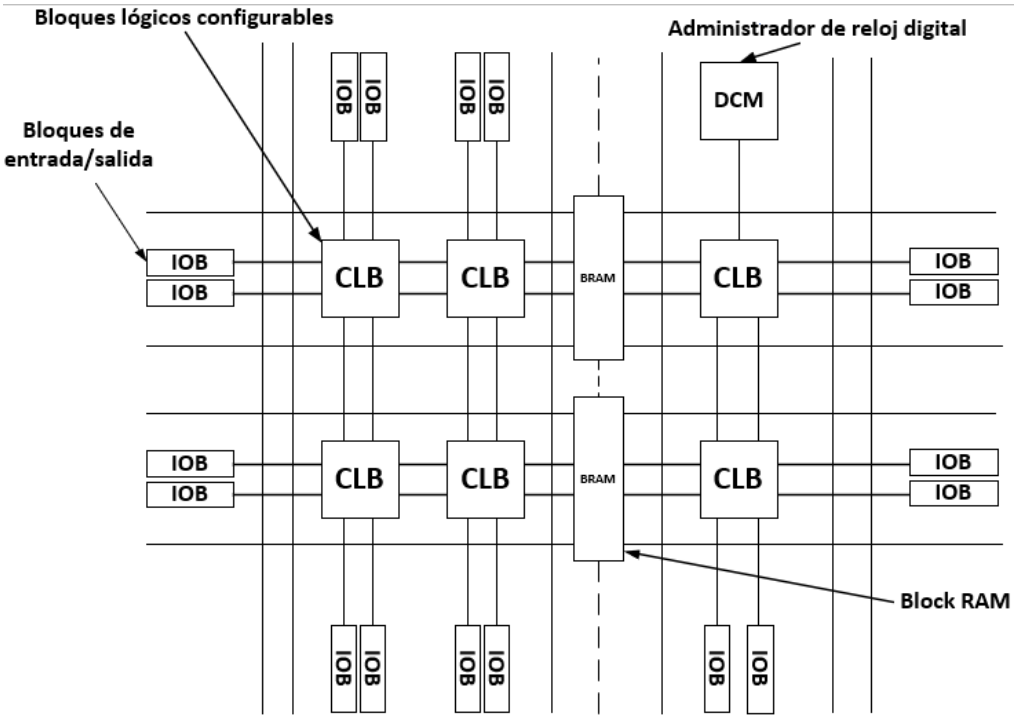


Fig. 2.5 Arquitectura básica de un FPGA

Un FPGA presenta las siguientes características:

- Permiten el procesamiento paralelo, a diferencia de los procesadores convencionales que ejecutan instrucciones secuencialmente. Esto se debe a que los FPGAs contienen múltiples bloques lógicos programables que pueden trabajar simultáneamente en diversas tareas. El paralelismo mejora la velocidad y el rendimiento pero también conlleva una mayor complejidad en la configuración y programación de los FPGAs, lo que requiere un diseño más elaborado y una curva de aprendizaje más pronunciada.
- Los FPGAs permiten la implementación de sistemas combinacionales y secuenciales. Los sistemas combinacionales se basan en la lógica booleana y generan resultados en función de las entradas actuales. En cambio, los sistemas secuenciales incluyen elementos de almacenamiento, como flip-flops y registros, para retener información. Implementar sistemas secuenciales en un FPGA puede ser complejo, ya que implica diseñar y conectar múltiples componentes básicos, como flip-flops y registros de desplazamiento, de manera adecuada para lograr el funcionamiento deseado del sistema secuencial.

- La forma más recomendada y ampliamente utilizada para ingresar diseños en FPGA es a través de Lenguajes de Descripción de Hardware (HDL) como VHDL y Verilog, en lugar de esquemáticos. Estos lenguajes permiten describir con precisión y detalle el comportamiento y la estructura del diseño, lo que facilita su implementación en un FPGA.

2.3.1 ZedBoard Zynq-7000

La tecnología utilizada en este proyecto es el ZedBoard que es una tarjeta de evaluación y desarrollo de bajo costo basada en el All Programmable SoC Z-7020 (AP SoC Z-7020) de la familia Zynq-7000 de Xilinx desarrollada por Avnet [20]. El dispositivo AP SoC Z-7020 es un único circuito integrado que combina un procesador de doble núcleo ARM Cortex-A9 con lógica programable de la familia de FPGAs Artix-7. Además, la tarjeta ZedBoard incluye numerosos periféricos como se puede observar en la tabla 2.1. El Zynq-7000 cuenta con 140 bloques de RAM de 36 Kb que pueden ser utilizados como bloques independientes de 18 Kb, el cual hacen una memoria en total de 5,040 Kb [21] . En la figura 2.6 se muestra los componentes de la tarjeta y en la tabla 2.1 se presentan sus características.

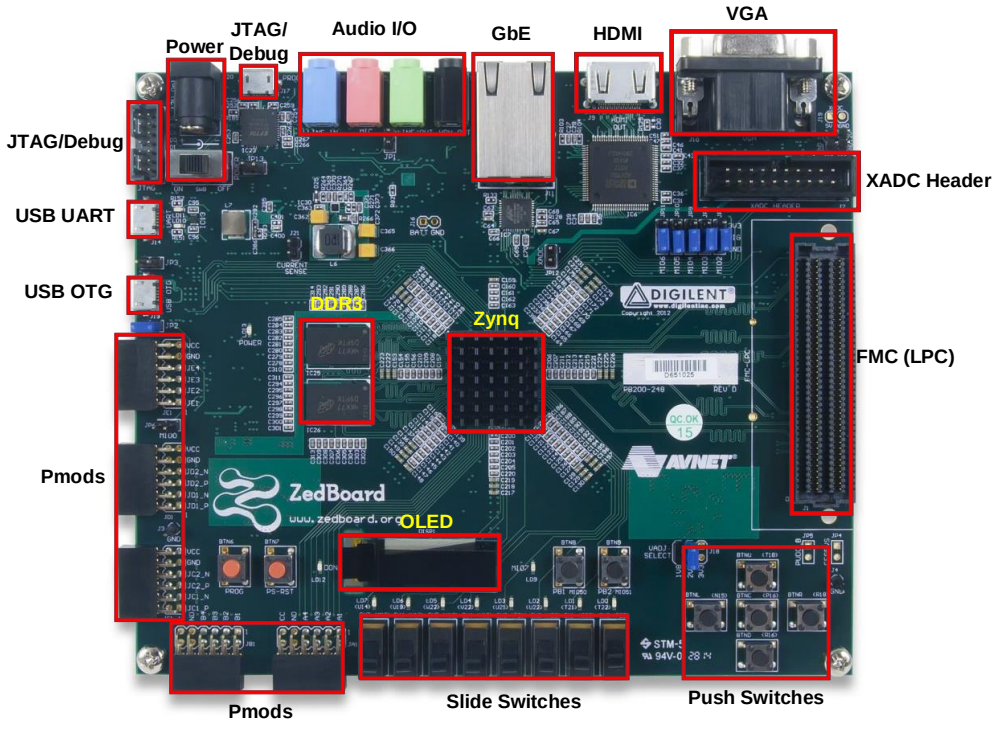


Fig. 2.6 Elementos que contiene una ZedBoard

TABLA 2.1 Caracteristicas de la ZedBoard [12]**.**

SoC	Xilinx® XC7Z020-1CLG484C Zynq-7000 AP SoC
Memoria	512 MB DDR3 (128M x 32) 256 Mb QSPI Flash
Interfaces	USB-JTAG Programming (PS/PL) 10/100/1G Ethernet USB OTG 2.0 SD Card USB 2.0 FS USB-UART bridge 5 Diligent Pmod compatible headers (2x6) (1 PS, 4 PL) 1 LPC FMC 1 AMS Header 2 Reset Buttons (1 PS, 1 PL) 7 Push Buttons (2 PS, 5 PL) 8 dip/slide switches (PL) 9 User LEDs (1 PS, 8 PL) DONE LED (PL)
On-board Oscillators	33.333 MHz (PS) 100MHz (PL)
Display/Audio	HDMI Output VGA (12-bit Color) 128x32 OLED Display Audio Line-in, Line-out, headphone, microphone
Power	On/Off Switch 12V @ 5A AC/DC regulator
Software	ISE® WebPACK Design Software License voucher for ChipScope™ Pro locked to XC7Z020

La estructura interna del Zynq consta de dos partes: Processing System (Sistema de procesamiento) y Programmable Logic (Lógica programable), como se muestra en la figura 2.7. El Processing System (PS) funciona como un procesador tradicional ya que contiene estructuras como ARM Cortex-A9, unidad de punto flotante (FPU), controlador de memoria, controlador Gigabit Ethernet y controlador USB. La parte de Lógica Programable (PL), por otro lado, contiene todas las estructuras de un FPGA estándar. La comunicación en las partes PS y PL se realizan mediante rutas de datos de alto rendimiento (data-paths).

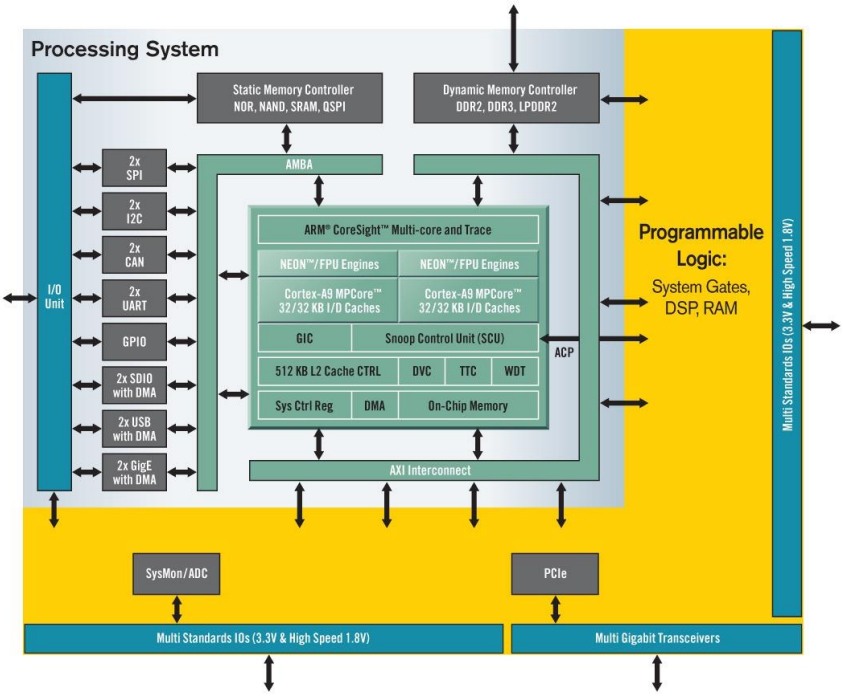


Fig. 2.7 Estructura interna de la arquitectura Zynq [22]

Este modelo presenta las siguientes ventajas:

Zynq-7000 All Programmable SoC es la plataforma ideal para infundir inteligencia a los sistemas empotrados de hoy en día. Al ser All Programmable no sólo se le puede dotar de inteligencia a través de la programación del software, sino que además se puede ejecutar el procesado de datos y decisiones en tiempo real a través del hardware programable y las interfaces del sistema se pueden optimizar y desarrollar a través de E/S programable.

Estos tres aspectos programables de los dispositivos Zynq hacen que se considere la forma más rápida y eficaz de crear sistemas más inteligentes capaces de hacer frente a los problemas de diseño que plantea el mercado.

A ello se suman los bajos costes de diseño, la alta flexibilidad, un alto rendimiento y bajo consumo [23].

2.4. Ventana de coincidencia

La ventana de coincidencia (τ) se refiere a un intervalo de tiempo específico durante el cual se considera que dos o más señales llegan al detector al mismo tiempo y se registra un evento de coincidencia. Esta ventana de tiempo se define mediante dos parámetros: el ancho de la ventana y la posición del centro de la ventana. El ancho de la ventana determina la duración del intervalo de tiempo en el cual se considerará una coincidencia, mientras que la posición del centro de la ventana establece la referencia temporal dentro de la cual se buscarán coincidencias [24]. En la figura 2.8, se visualiza el proceso de detección de una coincidencia, en la cual se reciben dos impulsos eléctricos en momentos temporales distintos (t1 y t2), y se establece la configuración del ancho de la ventana con el propósito de lograr la coincidencia.

La ventana de coincidencia es una herramienta clave para filtrar eventos de ruido y detectar señales genuinas de fotones coincidentes [7]. Al ajustar adecuadamente los parámetros mencionados anteriormente de la ventana de coincidencia, es posible mejorar la eficiencia y la precisión de la detección de coincidencias de señales en un sistema contador de coincidencia. Esto es especialmente importante en experimentos y aplicaciones donde se busca estudiar la correlación cuántica entre fotones y realizar mediciones precisas de fenómenos cuánticos.

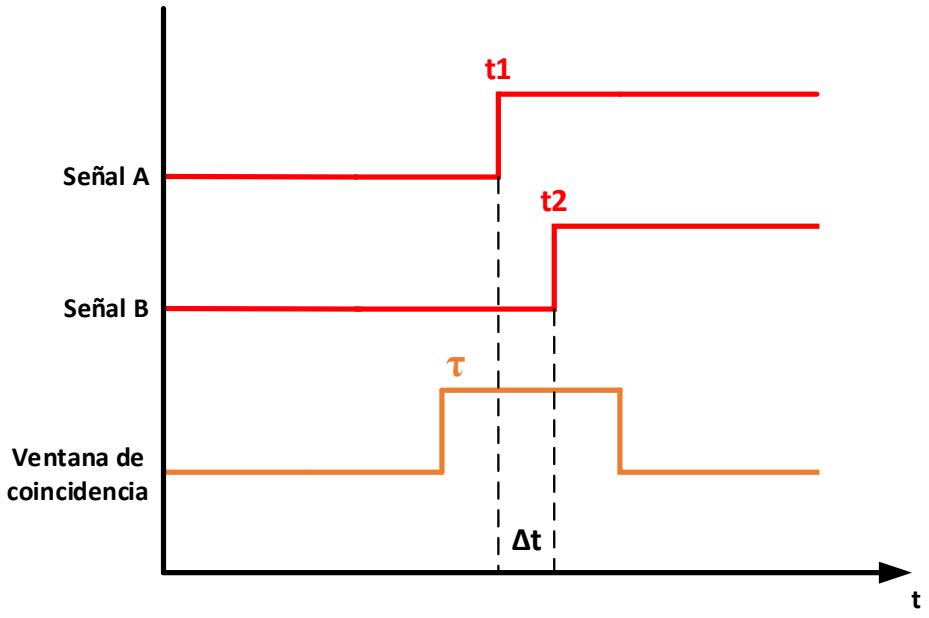


Fig. 2.8 Gráfica de la ventana de coincidencia

Además, con el propósito de asegurar la comprobación efectiva de la llegada de dos pulsos eléctricos dentro de una ventana de coincidencia, se debe cumplir con ecuación 2. Esta ecuación establece que, para obtener un conteo adecuado de la llegada de dos pulsos eléctricos dentro de la ventana de coincidencia, es requisito fundamental que la diferencia aritmética (Δt) entre los tiempos de llegada se mantenga por debajo o igual al valor del parámetro establecido para la ventana de coincidencia [25].

$$\Delta t = |t2 - t1| \quad \textbf{(1)}$$

$$\tau \geq \Delta t \quad \textbf{(2)}$$

2.5. Vivado

Vivado es un programa de diseño de hardware desarrollada por Xilinx, una empresa líder en el campo de la tecnología de FPGAs y sistemas en chip (SoCs). Vivado es ampliamente utilizado por los ingenieros de diseño de hardware para crear, simular, implementar y verificar circuitos digitales en FPGAs y SoCs de Xilinx.

Vivado ofrece un entorno de diseño integrado que incluye herramientas de diseño, herramientas de simulación, herramientas de implementación y herramientas de verificación. Se utiliza este Software para describir su diseño digital utilizando lenguajes de descripción de hardware como VHDL (VHSIC Hardware Description Language) o Verilog, simular su diseño para verificar su funcionamiento, implementar el diseño en un FPGA o SoC específico, y realizar la verificación post-implementación para asegurarse de que el diseño cumple con las especificaciones deseadas [19].

Además, ofrece una amplia gama de características avanzadas, como la optimización de alta velocidad, el análisis de temporización, la depuración en hardware, la generación de informes de rendimiento y consumo de energía, y la capacidad de integración con herramientas de terceros. Vivado se utiliza en una amplia variedad de aplicaciones incluyendo sistemas embebidos, comunicaciones, video, aeroespacial, defensa, médica, industrial y muchas otras áreas de diseño de hardware.

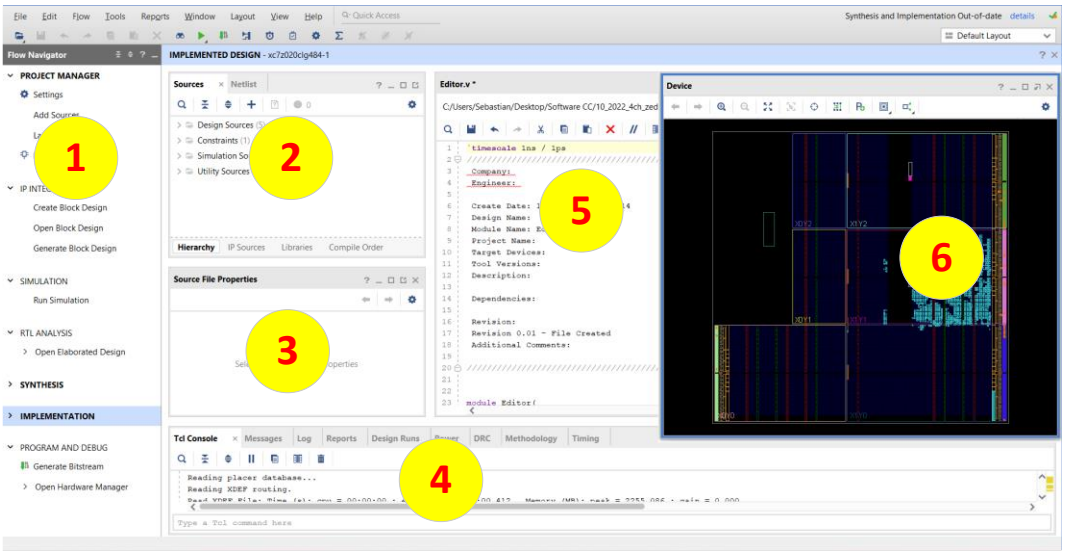


Fig. 2.9 Interfaz del programa Vivado

En la figura 2.9, se muestra el programa Vivado en donde:

1) Navegador: acceso rápido a los pasos del flujo de diseño y opciones de configuración.
2) Archivos de fuente: vista grafica de las fuentes y otros aspectos para las fuentes de diseño.
3) Propiedades de los ítems seleccionados.
4) Muestra la información actual de la consola, como reportes y detalles relacionados a las operaciones.
5) Editor: contiene información de la edición del proyecto.
6) Vista del diseño en hardware, clave para todos los pasos de implementación HDL.

2.5.1 Recursos de Vivado

La Suite de Diseño Xilinx de Vivado proporciona un flujo de diseño centrado en Intellectual Property (IP) que permite agregar módulos IP al diseño desde varias fuentes de diseño [19]. Entre los recursos que entrega Vivado están los siguientes:

1) Ram-Based shift Register

El núcleo LogiCORE IP de Xilinx RAM-based Shift Register genera registros de desplazamiento rápidos y compactos de varios bits de ancho como una línea de retardo. Se pueden crear registros de desplazamiento de longitud fija o de longitud variable [26]. Los registros de desplazamiento son circuitos digitales que almacenan y desplazan datos en serie de un bit a la vez, ya sea hacia la izquierda o hacia la derecha, en respuesta a una señal de reloj y/o a una señal de control. Estos registros de desplazamiento son ampliamente utilizados en una variedad de aplicaciones como la transferencia de datos en serie, la generación de secuencias de control, la implementación de máquinas de estados finitos y la manipulación de datos en sistemas de procesamiento de señales, entre otros.

2) ISERDES2

El ISERDES2 es un deserializador que convierte datos seriales a paralelo con características de reloj y lógica específicas diseñadas para facilitar la implementación de aplicaciones síncronas de alta velocidad de origen [27]. Las características del ISERDES2 incluyen:

- El Deserializador/Conversor de Serie a Paralelo permite transferir datos de alta velocidad sin que la estructura FPGA deba coincidir con la frecuencia de entrada de datos. Puede operar en los modos de tasa de datos única (SDR) y doble (DDR). En SDR, genera palabras paralelas de 2 a 8 bits de ancho, mientras que en DDR, crea palabras paralelas de 4 a 14 bits de ancho.
- Ofrece soporte dedicado para interfaces de memoria basadas en señales de sincronización, incluyendo el pin de entrada OCLK, para gestionar el cruce de dominios de reloj entre la señal de sincronización y la FPGA en el bloque ISERDESE2. Esto mejora el rendimiento y simplifica la implementación.

En la figura 2.10, es posible visualizar el proceso de deserialización en serie de un pulso eléctrico, el cual se presenta en una configuración de arreglo de bits. Al ingresar al módulo del deserializador, se lleva a cabo la transformación del formato serializado a un formato en paralelo, logrando que la señal de salida se presente en una configuración paralela.

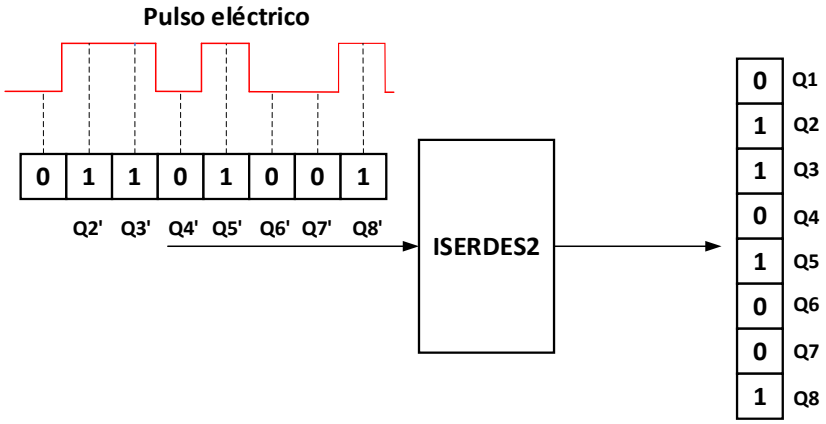


Fig. 2.10 Representación del módulo ISERDES2

El deserializador se caracteriza por su funcionamiento basado en un registro de desplazamiento, cuya aplicación práctica se centra en la conversión de datos serializados en un único cable hacia un formato paralelo distribuido en varios cables. Este proceso implica la recepción de los datos en formato serial y su posterior distribución en diferentes líneas paralelas mediante la utilización de Flip-Flops tipo D.

En este proceso, los datos en formato serial se desplazan a través de la entrada en serie (SI). Después de un número de ciclos de reloj igual al número de etapas, el primer bit de datos emerge en la salida en serie (SO) representada por QD en la figura 2.11.

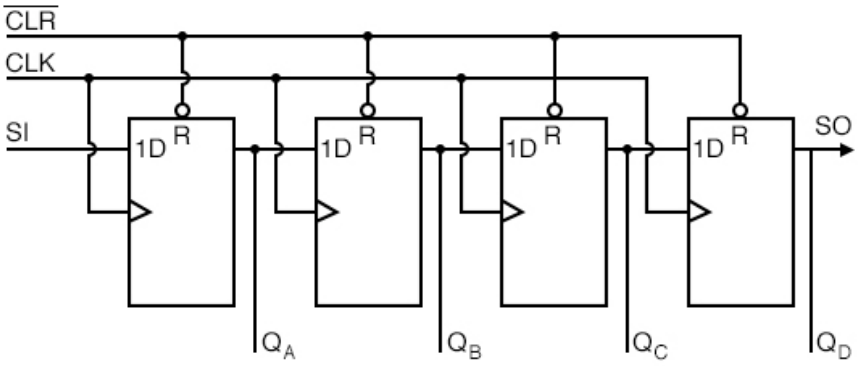


Fig. 2.11 Detalle del desplazamiento del registro de serie a paralelo [28]

En la figura 2.12, se observa el desplazamiento de 4 bits en serie (1011) que se presenta en la entrada SI. Estos datos se sincronizan con el reloj CLK.

En el primer pulso de reloj en t1, el dato 1 en SI se desplaza desde la entrada D hacia la salida QA’ de la primera etapa del registro de desplazamiento. Después de t2, este primer bit de datos se encuentra en QB’. Luego, después de t3, se encuentra en QC’. Después de t4, se ubica en QD’. De esta manera, cuatro pulsos de reloj han desplazado el primer bit de datos hasta la última etapa QD’.

El segundo bit de datos, que es 0, se encuentra en QC’ después del cuarto pulso de reloj. El tercer bit de datos, que es 1, se encuentra en QB’. Y el cuarto bit de datos, que también es 1, se encuentra en QA’.

Por lo tanto, el patrón de entrada de datos en serie 1011 está contenido en (QD' QC' QB' QA'). Ahora, estos datos están disponibles en las cuatro salidas. Entre los instantes t4 y t5, se lleva a cabo una acción de transferencia mediante un pulso RCLK positivo, con el fin de registrar los datos 1011 en las salidas QA, QB, QC y QD. En este intervalo de tiempo, los datos registrados se mantendrán en un estado congelado, incluso si se introducen nuevos datos durante los siguientes pulsos SRCLK (desde t5 hasta t8). No habrá cambios en los datos registrados hasta que se aplique otro pulso RCLK [28].

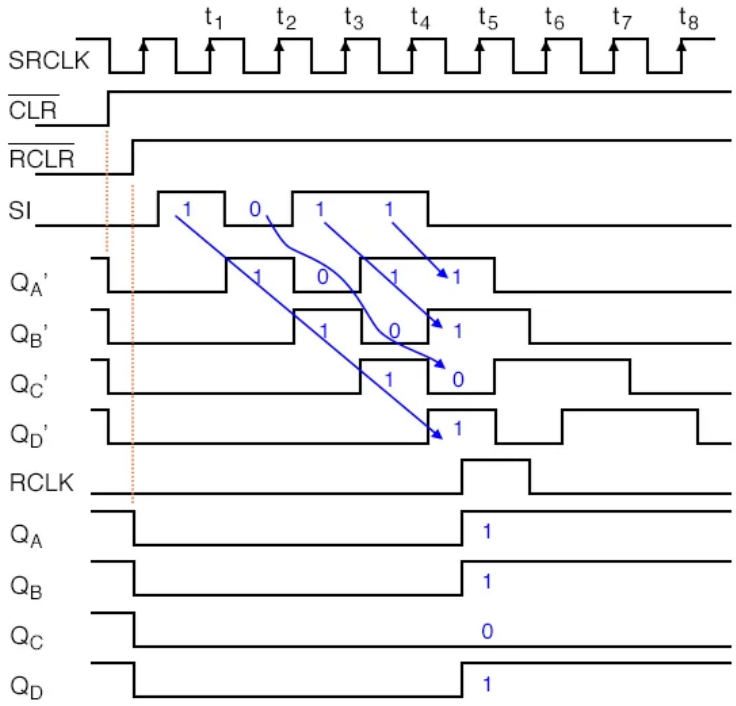


Fig. 2.12 Representación gráfica del registro de desplazamiento [28]

3) Clocking Wizard

El Clocking Wizard es un IP Core de Xilinx que se puede generar utilizando las herramientas de diseño de Xilinx Vivado y simplifica la tarea de creación de código fuente HDL para circuitos de reloj personalizados según los requisitos de sincronización. El asistente guía para configurar los atributos apropiados para la primitiva de sincronización y permite anular cualquier parámetro calculado. Además de proporcionar una envoltura HDL para implementar el circuito de sincronización deseado, el Clocking Wizard también entrega un resumen de parámetros de temporización generado por las herramientas de temporización de Xilinx para el circuito [29].

- La selección del mixed-mode clock manager (MMCM) y phase-locked loop (PLL) se hacen mediante la opción Integrated design environment (IDE) para las características admitidas de las primitivas.
- La función Safe Clock Startup permite obtener un reloj estable y válido en la salida. Al habilitar la función Sequencing, se proporcionan relojes de salida secuenciados.
- Acepta hasta dos relojes de entrada y hasta siete relojes de salida por red de reloj.
- Configura automáticamente una primitiva de reloj en función de las características de reloj seleccionadas.
- Calcula automáticamente la frecuencia del oscilador controlado por voltaje (VCO) para las primitivas con un oscilador y proporciona valores de multiplicación y división en función de los requisitos de frecuencia de entrada y salida.

2.6. Microsoft Visual C#

Visual C# (también conocido como Microsoft Visual C#) es un entorno de desarrollo integrado (IDE, por sus siglas en ingles) y un lenguaje de programación orientado a objetos desarrollado por Microsoft. C# es parte de la plataforma .NET y se utiliza para crear una amplia variedad de aplicaciones, desde aplicaciones de escritorio hasta aplicaciones web y servicios en la nube.

Visual C# proporciona un conjunto de herramientas y características que facilitan el desarrollo de software. Algunas de sus principales características incluyen la gestión de memoria automática a través del recolector de basura, la seguridad de tipos estática, la interoperabilidad con otros lenguajes de .NET y una sintaxis clara y legible.

Con Visual C#, los desarrolladores pueden crear aplicaciones de software robustas y escalables utilizando una combinación de código de alto nivel y componentes de .NET Framework. El IDE de Visual C# ofrece características como el resaltado de sintaxis, depuración visual, asistencia de código y una interfaz intuitiva que facilita el proceso de desarrollo [30]. La interfaz de este programa se muestra en la figura 2.13.

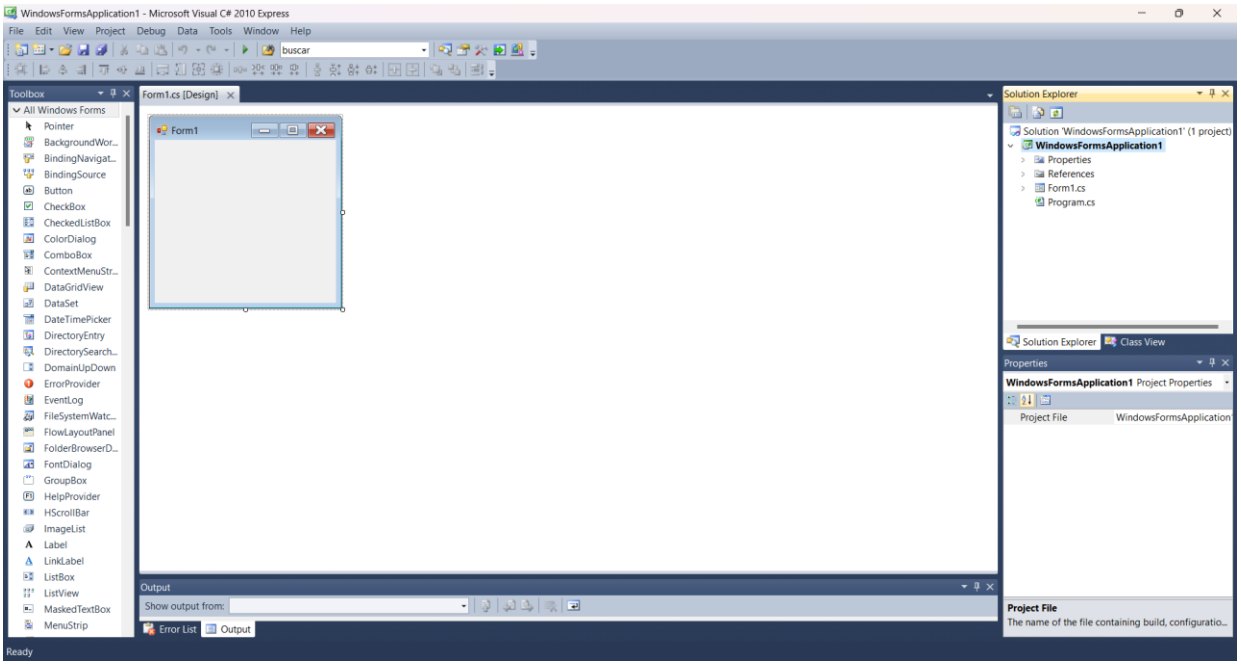


Fig. 2.13 Interfaz del software Visual C#

A continuación, se presentan de forma resumida las principales características de C# [31]:

- C# tiene todas las características de la programación orientada a objetos, como encapsulación, herencia y polimorfismo. Estas características permiten organizar el código de manera modular y reutilizable, facilitando el desarrollo de aplicaciones flexibles y extensibles.
- Ofrece un modelo de programación coherente y consistente basado en programación orientada a objetos. En C#, todo el código se escribe dentro de clases, incluso los tipos de datos básicos son tratados como clases heredadas de System.Object. Esto permite compartir características comunes definidas en la clase base System.Object, lo que facilita el uso de métodos comunes en todos los tipos del lenguaje.
- Es un lenguaje de programación fuertemente tipado, lo que significa que controla estrictamente las conversiones entre tipos para garantizar la compatibilidad. Esto previene errores comunes y protege la integridad de otras aplicaciones, ya que impide el acceso no autorizado a áreas de memoria, evitando problemas de seguridad o funcionamiento incorrecto.
- Incorpora un recolector de basura (Garbage Collector) que elimina la necesidad de que los programadores eliminen manualmente referencias a objetos no utilizados. Este recolector automático detecta y elimina objetos sin uso, evitando la acumulación de basura en la memoria. Esto previene agotar la memoria y evita errores como la liberación inapropiada de áreas de memoria, lo que podría causar problemas inesperados o inestabilidad en el programa.
- Incluye soporte nativo para eventos y delegados. Los delegados son similares a punteros a funciones en otros lenguajes, pero más orientados a objetos, lo que facilita la programación orientada a eventos. Los eventos permiten a los objetos notificar eventos, como la pulsación de un botón o la modificación de un texto, y se usan en interfaces gráficas y programación de respuestas a eventos en aplicaciones.
- Admite el uso de atributos, que son elementos adicionales en las clases y proporcionan información extra sobre ellas. Estos atributos se agregan a la declaración de una clase para indicar características específicas o dar instrucciones adicionales sobre su manejo. Por ejemplo, los atributos pueden determinar si un miembro de la clase debe mostrarse en la ventana de propiedades de Visual Studio.NET, especificar valores válidos para los miembros, y proporcionar detalles relevantes para el entorno de desarrollo.

2.6.1 Clases

Las clases son elementos fundamentales de la programación orientada a objetos en lenguajes como C#. Una clase es una plantilla o estructura que define las propiedades y comportamientos de un objeto. Puede considerarse como un molde o plano a partir del cual se crean los objetos.

En términos simples, una clase es un conjunto de variables (llamadas atributos o propiedades) y funciones (llamadas métodos) que operan en esos datos. Los atributos representan el estado o las características del objeto, mientras que los métodos definen las acciones o comportamientos que el objeto puede realizar.

Las clases permiten la creación de múltiples instancias de objetos que comparten las mismas características y comportamientos definidos por la clase. Cada objeto creado a partir de una clase se conoce como una instancia de esa clase. Las instancias tienen su propio conjunto de datos y pueden realizar acciones utilizando los métodos definidos en la clase [31].

Capítulo 3. Desarrollo del sistema en general

3.1. Arquitectura en hardware

La arquitectura del sistema contador de coincidencia de fotones se muestra en la figura 3.1. El sistema consta de canales establecidos (canal 0, canal 1, canal 2 y canal 3) a los cuales ingresa directamente el pulso eléctrico. Estos canales están conectados al FPGA. El pulso se muestrea a una frecuencia de 1 GHz en el módulo ISERDES2. A continuación, pasa al módulo DelayRAM para introducir un retraso. Este retraso se proporciona cada 8 ns a una frecuencia de reloj de 125 MHz. La señal luego ingresa al módulo conversor digital a tiempo (DTC), donde se convierte en un tiempo que representa el momento de llegada de cada pulso. A las salidas del DTC se calculan las diferencias aritméticas entre los tiempos (Δt) de llegada de los canales 0, 1, 2 y 3 registrando las llegadas coincidentes en diferentes ventanas de detección de manera simultánea.

Cada pulso que entre en el sistema se registrará como cuentas simples en el módulo CS. Además, según las coincidencias entre los canales, las cuentas se registrarán como cuentas dobles (CD). Los resultados se envían al módulo CPU y luego se comparten con el usuario a través de una conexión USB.

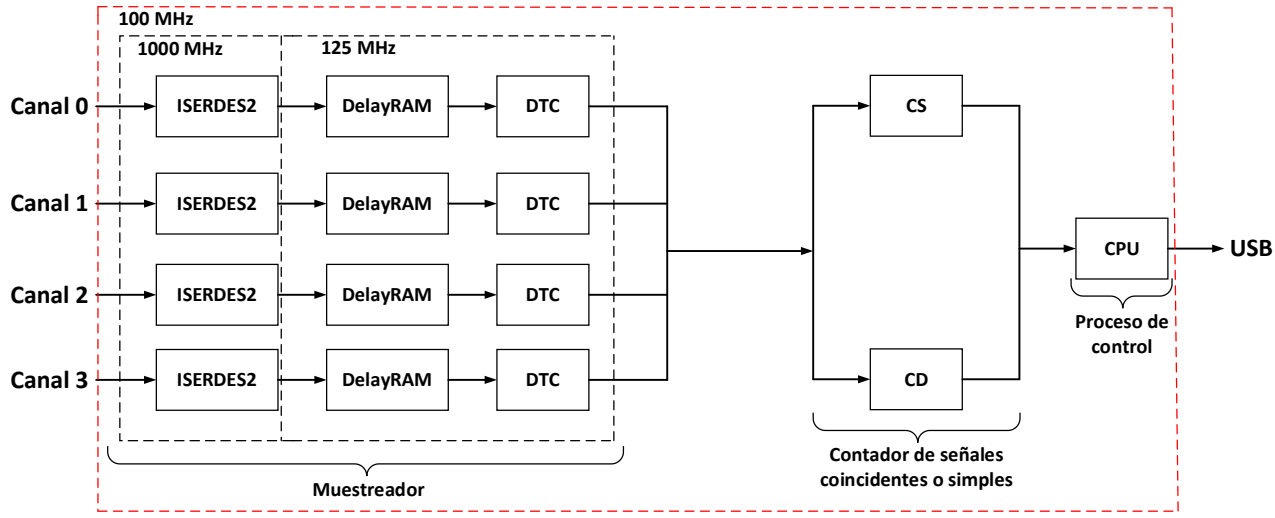


Fig. 3.1 Arquitectura del sistema contador de coincidencia de señales

La arquitectura del sistema, implementada en Vivado, comienza con la creación de un módulo Clocking Wizard. Este módulo requiere un MMCM (Multiple Mode Clock Manager) para generar múltiples frecuencias de reloj de salida independientes y ajustables a partir de una única señal de reloj de entrada. La frecuencia del reloj de entrada se establece en 100 MHz. A través del MMCM, se generan tres frecuencias de reloj de salida específicas: 100 MHz, 125 MHz y 500 MHz, como se muestra en la figura 3.2. Estas frecuencias se adaptan a los requisitos particulares del sistema, lo que permite un funcionamiento óptimo y eficiente.

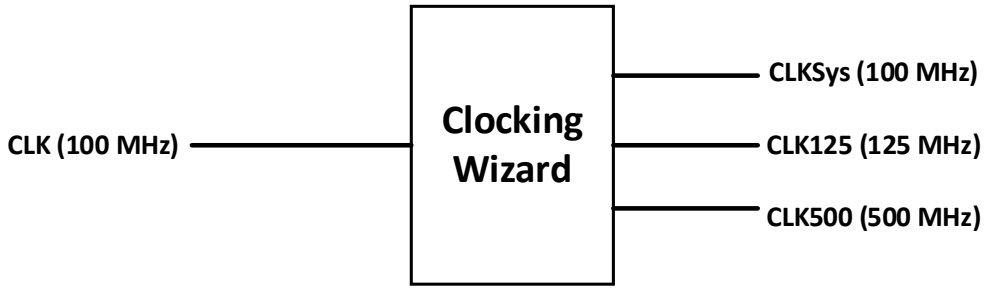


Fig. 3.2 Representación del módulo Clocking Wizard

3.1.1 Módulo ISERDES2

En la arquitectura del sistema se introduce un pulso analógico proveniente del SPD (Single-Photon detector) a través de los canales predefinidos. Este pulso es dirigido al módulo muestreador de alta resolución, donde se establece una conexión directa con un deserializador llamado ISERDES2.

El ISERDES2 desempeña una función esencial en el sistema al realizar la conversión de datos de una señal en serie a una forma paralela, así como digitalizarlos a una tasa de muestreo de 8 ns. Este componente utiliza técnicas de deserialización y muestreo para recibir la señal en serie y separar los bits individuales que la componen, generando una secuencia de datos en paralelo. Para lograr esto, opera en modo DDR (Double Data Rate), lo que implica que captura los datos tanto en el flanco de subida de la señal de reloj de 500 MHz como en la de su inversa. Esta configuración efectivamente duplica la frecuencia de reloj a 1 GHz, lo que resulta en una tasa de muestreo de 1 ns por cada bit del pulso, esto se puede observar en el esquemático de la figura 3.3. Además, es importante destacar que el ISERDES2 se encuentra configurado en una relación de 1:8. Esto significa que puede generar registros en paralelo de 8 bits, los cuales se vuelven accesibles cada 8 ns utilizando un reloj de 125 MHz. Esta relación de configuración permite procesar de manera eficiente y simultánea múltiples bits del pulso de entrada, mejorando así la velocidad y eficacia del sistema en general.

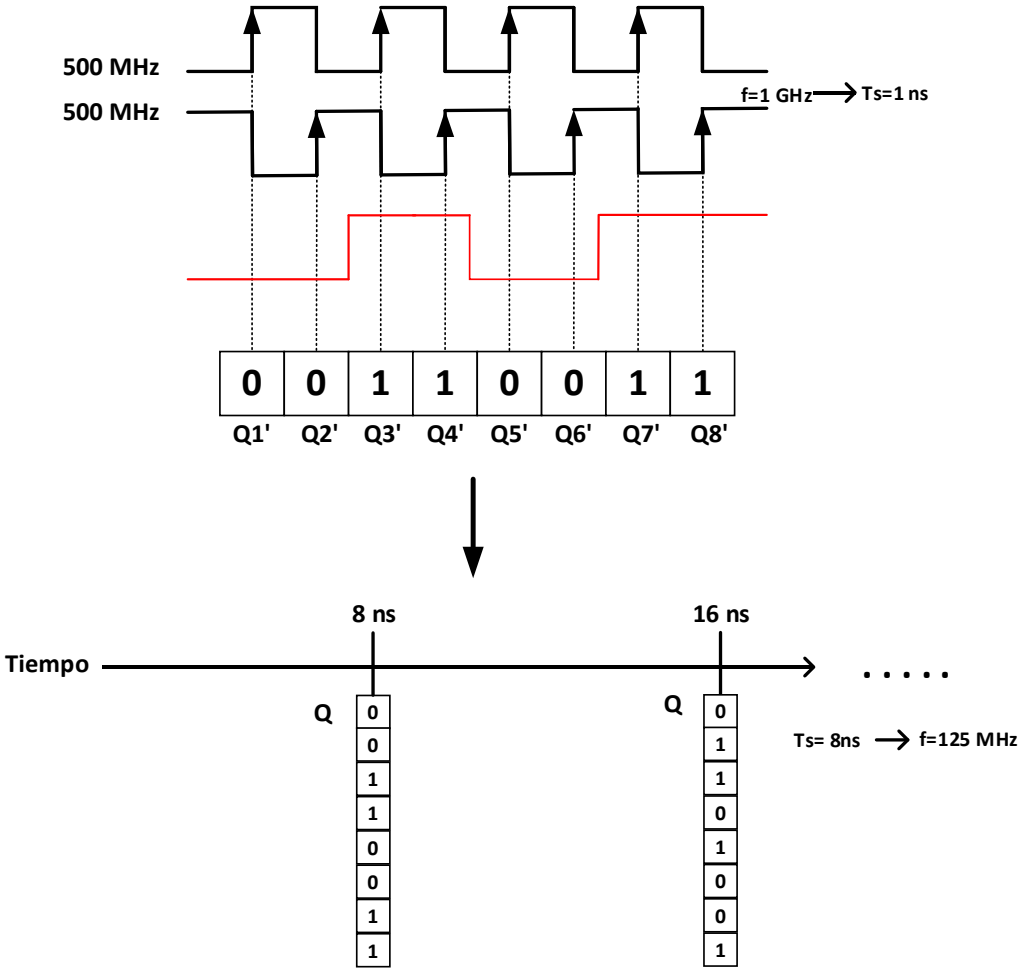


Fig. 3.3 Representación gráfica del Deserializador (ISERDES2)

3.1.2 Módulo DelayRAM

Los vectores resultantes del ISERDE2, que contienen datos de 8 bits, se dirigen al módulo delayRAM. Este módulo se crea empleando el RAM-based Shift Register suministrado por Vivado. Su implementación se enfoca en registrar eventos de coincidencia dentro de la ventana de coincidencia aplicada a todas las cuentas disponibles. El objetivo principal del módulo delayRAM es permitir el retardo de una señal específica en relación con otras señales presentes en el sistema. En la figura 3.4.a, se muestra la señal A sin ningún tipo de retraso. La señal se representa con su forma de onda original, sin modificaciones ni desplazamientos. Por otro lado, en la figura 3.4.b, se ilustra la señal A después de aplicarle el retraso correspondiente. Se puede observar cómo el pulso se desplaza en el tiempo, lo que indica que la señal ha sido retardada de acuerdo con los parámetros establecidos y se registra una cuenta de coincidencia de fotones. Este retraso puede ser ajustado y configurado según las necesidades del sistema.

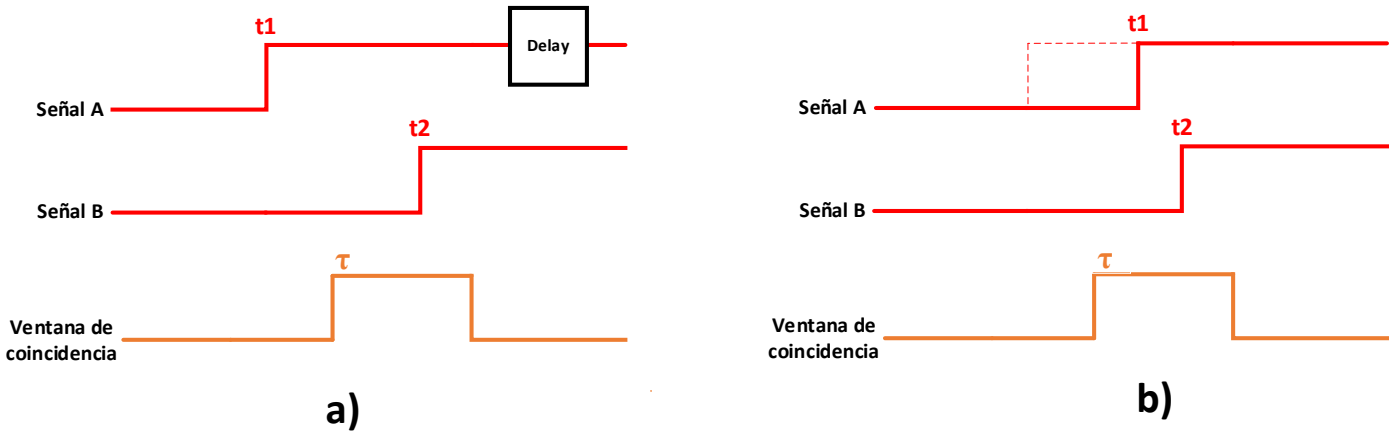


Fig. 3.4 Aplicación de retardo a un pulso
(a) Pulso t1 sin retardo aplicado, **(b)** Pulso t1 con retardo aplicado.

El módulo DelayRAM se encuentra configurado con una entrada de dirección de 10 bits, lo que implica la generación de 1024 casillas de almacenamiento de datos. Cada casilla contiene 8 bits de información. El módulo está sincronizado con una señal de reloj de 125 MHz, lo que conlleva a que cada 8 ns se realice el desplazamiento entre las casillas de almacenamiento. Esta sincronización precisa permite un procesamiento y acceso eficiente a los datos almacenados en el DelayRAM, facilitando el retardo adecuado de las señales según los requisitos del sistema.

El funcionamiento interno de este módulo se representa gráficamente en la figura 3.5. Toda la información ingresa a la casilla 0 y mediante la selección de la dirección A=0 utilizando un selector de 10 bits, durante el primer ciclo de reloj, los datos almacenados en la casilla 0 se transmitirán sin ningún retraso, saliendo del módulo de forma inmediata (figura 3.5.a). Sin embargo, si se mantiene la selección en A=0, durante el siguiente ciclo de reloj, la información se desplazará a la siguiente casilla de almacenamiento (figura 3.5.b). Como resultado, no se obtendrá ningún dato en la casilla 0 en este ciclo.

Por otro lado, si se selecciona A=2, se requerirán tres ciclos de reloj para que los datos se transmitan. Esto implica un tiempo total de retardo de 16 ns, ya que cada flanco de subida del reloj ocurre cada 8 ns. En consecuencia, la información almacenada en la tercera casilla estará disponible para su transmisión después de los tres ciclos de reloj (figura 4.5.c).

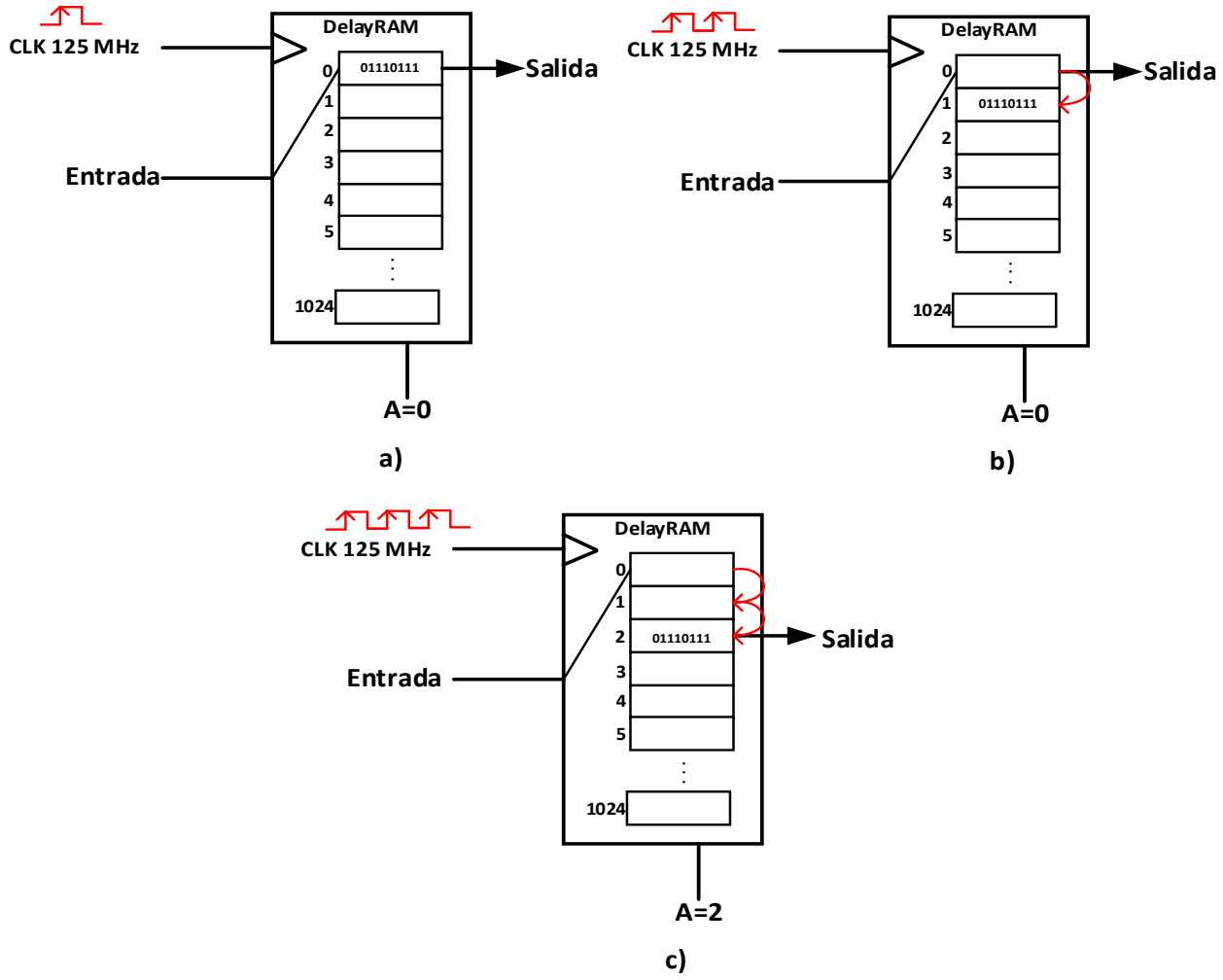


Fig. 3.5 Representación del desplazamiento de información en el DelayRAM
(a) La información sale sin desplazamiento con A=0, **(b)** La información es desplazada una casilla con A=0, **(c)** La información es desplazada con A=2.

3.1.3 Módulo DTC

La señal, independientemente de si tiene o no un retardo, ingresa a un módulo llamado Bit2time, el cual es un convertidor digital a tiempo (DTC en inglés). A través de este proceso, se genera un registro de 9 bits que representa el tiempo. Esta información de tiempo relativo es de vital importancia en un contador de coincidencia de fotones, ya que permite analizar y correlacionar los eventos de detección de fotones en distintos canales. En la Figura 3.6 se muestra el módulo DTC, el cual consta de tres entradas: una señal de reloj de 125 MHz, un retardo fino (df) de 3 bits y la información representada en un arreglo digital (Qd) de 8 bits. La salida de este módulo es un registro de información representado en el dominio del tiempo (tQ) de 9 bits.

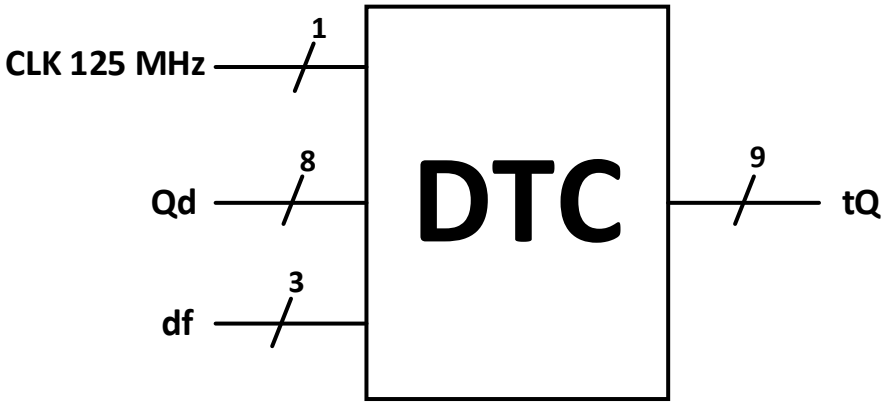


Fig. 3.6 Representación del módulo DTC

En este módulo, se evalúa el pulso entrante y se le asigna un tiempo inicial en función de su patrón. Por ejemplo, si llega un pulso con el patrón “011111111”, se le asigna un tiempo 1 ns. Si llega un pulso con el patrón “001111111”, se le asigna un tiempo 2 ns, y así sucesivamente hasta llegar al patrón “000000001”, que se asigna un tiempo inicial de 8 ns. Cuando el primer bit del pulso es un “0”, se registra una cuenta, lo cual indica la detección de un evento. Sin embargo, si el primer bit es un “1”, se considera un falso positivo, es decir, una repetición de la señal sin una nueva detección.

Los primeros 8 bits del patrón del pulso indican el tiempo de detección, mientras que el último bit indica si se detectó o no un pulso. Si el último bit es “1”, significa que se detectó un pulso, y si es “0”, indica que no se detectó. Inmediatamente después de que se detecta el pulso, se mantiene en alto durante un período determinado.

Una vez que se detecta el pulso eléctrico, se le suma 8 ns al registro de tiempo. Por ejemplo, si se detecta inicialmente a los 1 ns, en el siguiente flanco de subida del reloj el tiempo sería de 9 ns, y en el siguiente ciclo de reloj sería de 17 ns. De esta manera, el tiempo se incrementará en intervalos de 8 ns hasta alcanzar los 200 ns. Al superar este límite de 200 ns, el último bit del registro vuelve a ser 0.

Además, en este módulo se presenta un delay fino (df) de 3 bits, cuya función es variar el tiempo inicial en incrementos de 1 ns. Esto permite ajustar finamente el tiempo de inicio de cada detección y controlar la sincronización precisa entre eventos en intervalos de tiempo muy pequeños.

Una vez que se obtiene el tiempo, representado en una cadena de 9 bits, desde este módulo, ingresa a una serie de módulos de cuentas simples, dobles, triples y cuádruples, que están sincronizados con el reloj del sistema de 100 MHz.

3.1.4 Módulo de Cuentas Simples

Este módulo se utiliza para realizar el seguimiento de los pulsos eléctricos detectados. Cuenta con varias entradas, que incluyen una señal de reloj de 100 MHz, una señal de información de 8 bits (tQ), control de almacenamiento en memoria de 7 bits (ControlWR) y una señal de reinicio para borrar la información almacenada en la memoria y reiniciar el sistema de conteo. La salida del módulo es la cuenta de los pulsos detectados, representada en 24 bits (MemC), que se dirige directamente a la memoria para su almacenamiento. La representación de este módulo se puede observar en la figura 3.7.

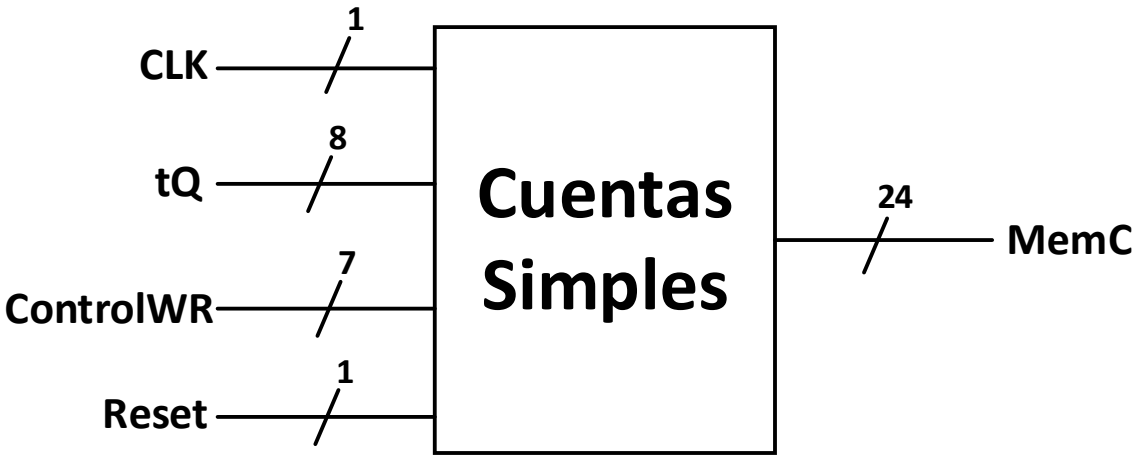


Fig. 3.7 Representación del módulo de cuentas simples

En este módulo la detección de un pulso se basa en matrices de ocho “Oneshot”, que son módulos diseñados para detectar el pulso cuando el último bit de la cadena es un 1 como en la figura 3.8.a y caso contrario es un 0 como en la figura 3.8.b. Una vez que se detecta un pulso, el contador se incrementa en uno, y esta información se guarda en una memoria para poder acceder a ella posteriormente.

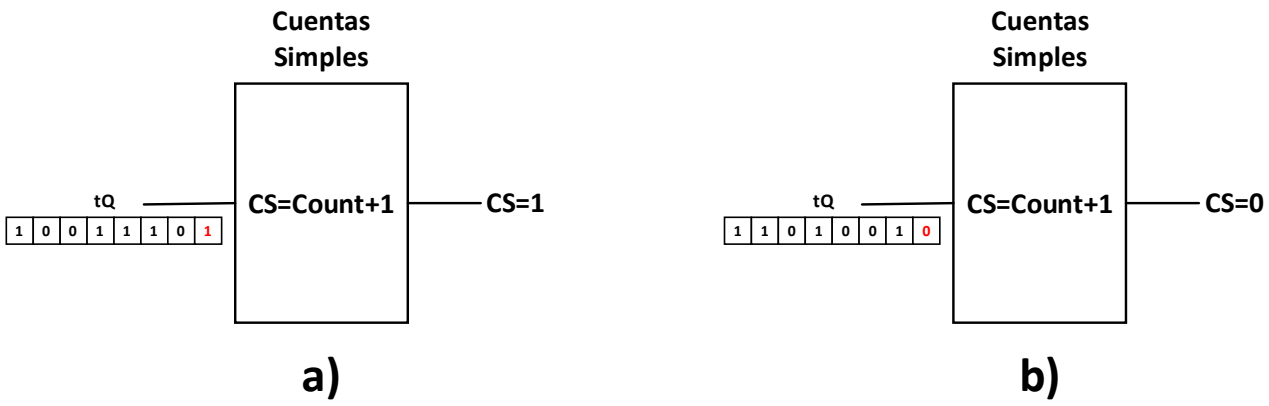


Fig. 3.8 Representación del módulo de cuentas simples.
(a) Consideración de una cuenta cuando el ultimo bit es 1, **(b)** No se considera cuenta cuando el ultimo bit es 0.

3.1.5 Módulo de Cuentas Dobles

El módulo de cuentas dobles, como se muestra en la figura 3.9, requiere varias entradas. Estas incluyen una señal de reloj de 100 MHz, dos señales de tiempo (tQ1 y tQ2) de 9 bits cada una, una ventana de coincidencia configurable (τ) para detectar la coincidencia de dos pulsos. La ventana de coincidencia se puede configurar con un ancho de 1, 2, 4, 8 y 16 ns. Además, el módulo cuenta con una señal de control de memoria de 7 bits (ControlWR) y un reset para reiniciar el sistema.

La salida del módulo es la cuenta de los pulsos coincidentes entre dos pulsos eléctricos, representada en 24 bits (MemC).

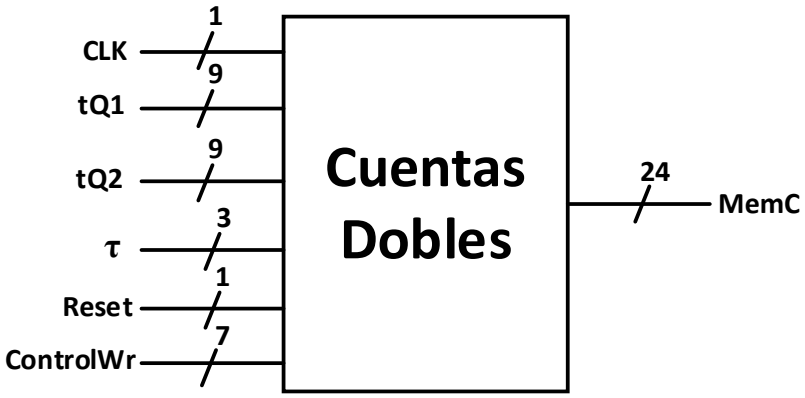


Fig. 3.9 Representación del módulo de cuentas dobles

Este módulo opera utilizando una estructura de control basada en una máquina de estados finitos, en la cual se llevan a cabo de manera secuencial una serie de etapas en donde se ve reflejado en el diagrama de flujo de la figura 3.10. En la etapa cero, se detectan dos pulsos coincidentes provenientes del módulo “Oneshot”. Esta detección se realiza sobre los últimos bits de las señales de tiempo, es decir, si ambos bits son iguales a “1”, se registra. Sin embargo, esto no implica necesariamente que las señales hayan llegado en tiempos iniciales cercanos. Una vez que se detectan dos pulsos, se procede a la etapa 1, donde se evalúa y ajusta el ancho de la ventana de coincidencia, ya sea para tener un ancho de 1, 2, 4, 8 o 16 ns. A continuación, se pasa a la etapa 2, en la cual se realiza una operación aritmética para calcular la diferencia entre los tiempos de llegada de las dos señales. Si el ancho de la ventana de coincidencia es mayor que esta diferencia, se registra una cuenta de coincidencia. Si el tiempo de llegada de la señal 1 es mayor que el tiempo de llegada de la señal 2, se procede a la etapa 3. Por otro lado, si el tiempo de llegada de la señal 2 es mayor que el tiempo de llegada de la señal 1, se dirige a la etapa 4. Esto se hace para evitar los tiempos negativos y considerar la diferencia en valor absoluto.

En la etapa 3, se encuentran cinco contadores que incrementarán en función del valor resultante de la diferencia aritmética de tiempo (Δt). Cada contador corresponde a una categoría específica y se incrementará según el rango en el que se encuentre Δt. Por otro lado, en la etapa 4, hay tres contadores que también aumentan en función del valor resultante de la diferencia de tiempo. Estos contadores representan categorías distintas y se incrementan de acuerdo al rango en el que se ubique Δt. Los ocho contadores en total se representan gráficamente mediante un histograma, donde cada celda del histograma corresponde a los diferentes valores de Δt.

Finalmente, se pasa a la etapa 5, donde se evalúa el último dígito de las dos señales de 9 bits. Si ambos dígitos son 0 o si alguno de ellos es 0, no se considera una cuenta de coincidencia y se regresa a la etapa 0 para comenzar un nuevo ciclo. Este proceso se repite continuamente, permitiendo la detección y evaluación de las coincidencias entre las señales en cada ciclo, mientras se descartan aquellas que no cumplen con los criterios establecidos.

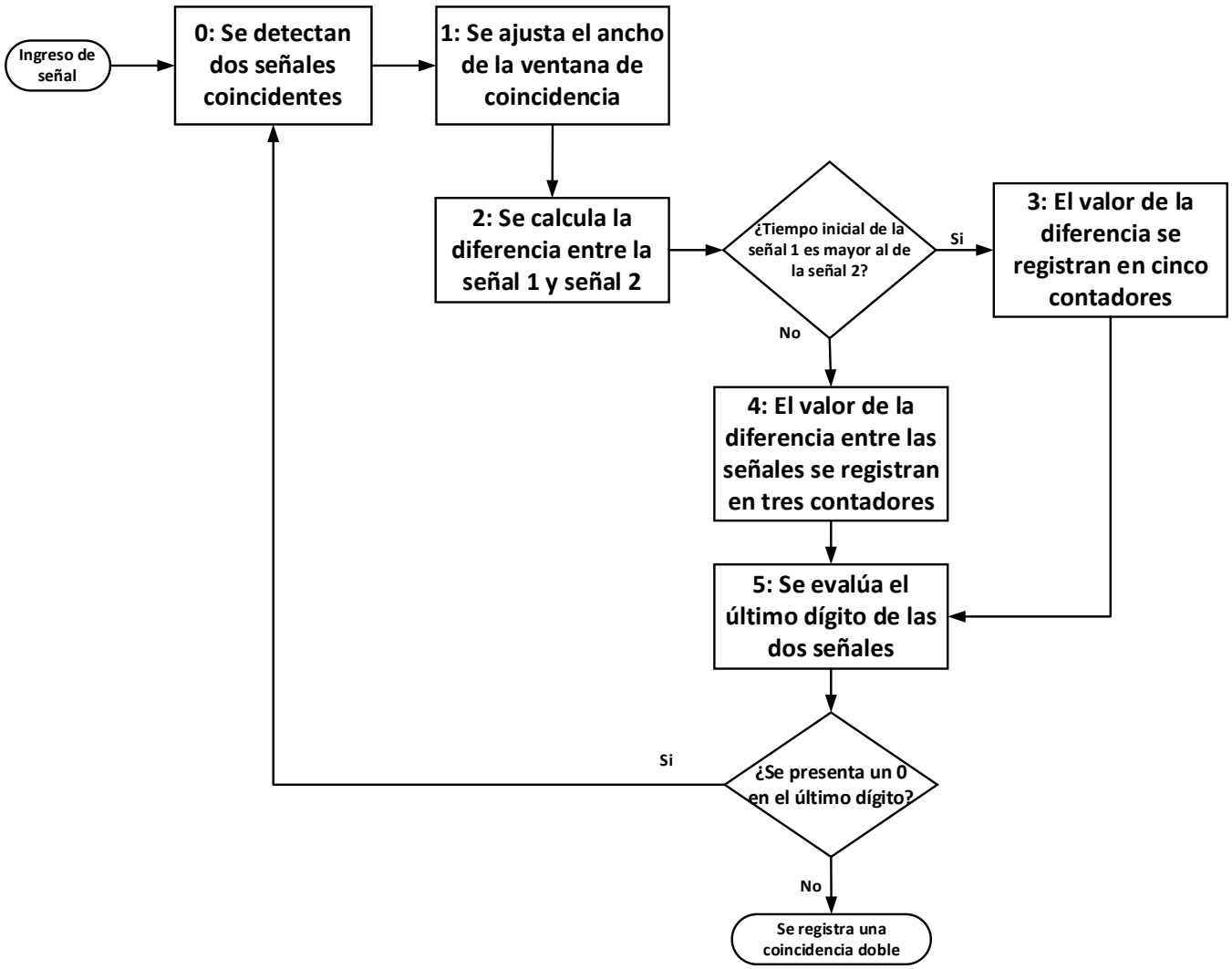


Fig. 3.10 Diagrama de flujo del módulo de cuentas dobles

3.1.6 Módulo CPU

El módulo de la CPU tiene la responsabilidad de almacenar y enviar los datos al software de control y monitoreo aprovechando un reloj de 100 MHz para sus operaciones. Este módulo cuenta con dos componentes internos: uno encargado de recibir la orden de control de la ventana de coincidencia y el retardo (control_rx), y otro encargado de almacenar y enviar los datos hacia el software de control y monitoreo (control_tx). En la Figura 3.11 se puede observar el módulo CPU que cuenta con diversas entradas y salidas. Entre las entradas se encuentra MemC de 24 bits, que representa los datos generados por las cuentas simples y/o dobles. También se encuentra RsRx, una señal que recibe órdenes provenientes del software. En cuanto a las salidas, se destaca RsTx, una señal asociada a la transmisión de datos hacia el software. ControlWR de 7 bits que es una señal que controla directamente el proceso de escritura/lectura en la memoria. Por su parte, Reset se utiliza para restablecer o reiniciar el estado de cada módulo, permitiendo iniciar desde un estado predefinido. Adicionalmente, el parámetro Delay de 16 bits se emplea para controlar el retardo, mientras que τ de 4 bits representa la ventana de coincidencia. Cabe mencionar que estas últimas cuatro salidas se utilizan como entradas en los módulos de cuentas.

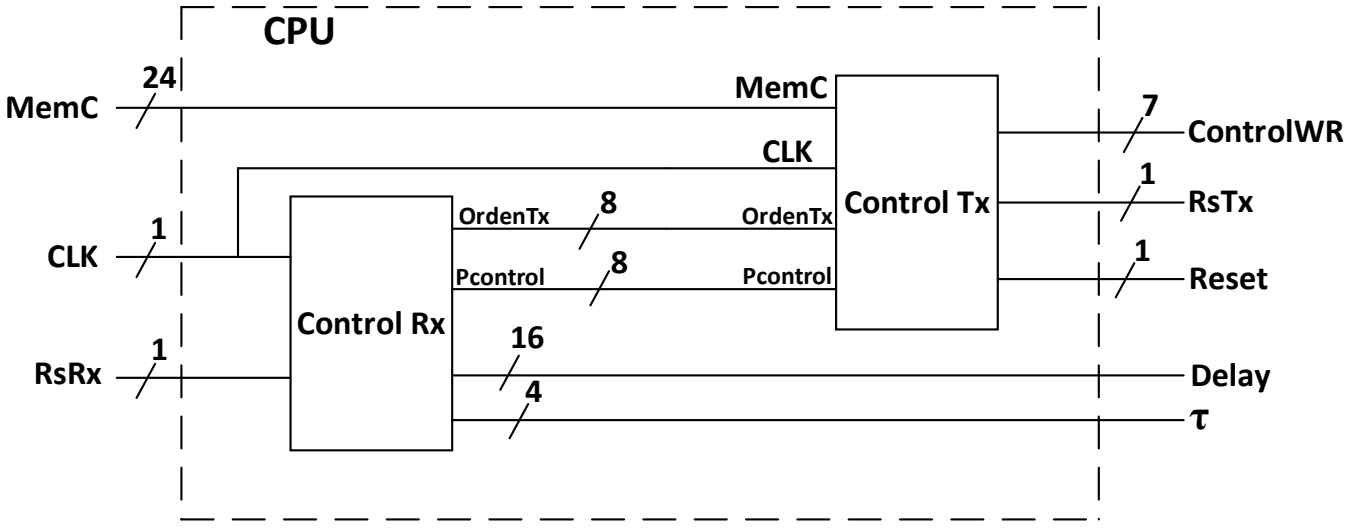


Fig. 3.11 Representación del Módulo CPU

El módulo de control Rx tiene la función principal de administrar la ejecución de órdenes, interpretar los datos recibidos y realizar acciones específicas según los comandos recibidos. Una de las tareas importantes del módulo es ajustar el ancho de la ventana de coincidencia y la duración del retardo para un pulso eléctrico.

Como se observa en la figura 3.12, dentro del módulo de control Rx, se encuentra el módulo UART Rx, responsable de recibir informaciones provenientes del software a través de la señal denominada “RsRx”. Cuando se recibe una señal de orden, el módulo UART Rx captura los datos y los almacena en un registro llamado “ORx”. Este registro tiene una capacidad de 8 bits y su función es mantener temporalmente los datos recibidos. Esto permite que los datos puedan ser procesados posteriormente y que se puedan tomar decisiones en base a ellos. El registro “ORx” actúa como un almacenamiento temporal para los datos recibidos, facilitando su posterior manipulación y análisis.

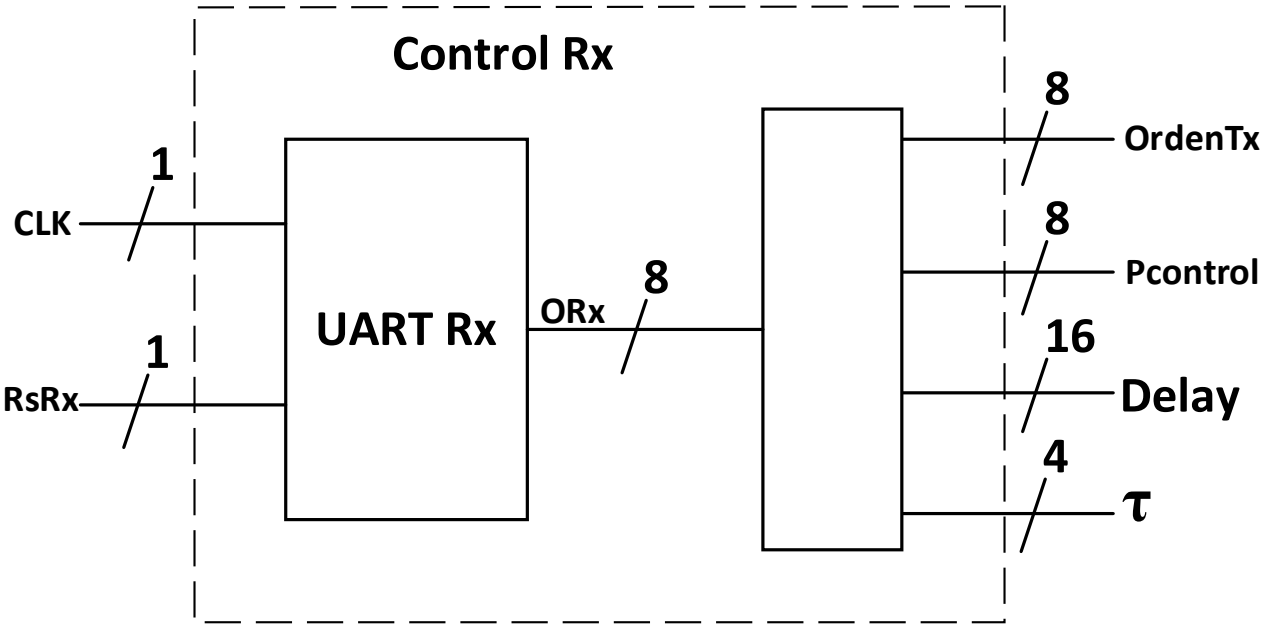


Fig. 3.12 Representación del módulo Control Rx

El control Rx funciona utilizando una estructura de control secuencial que analiza el valor del registro llamado “ControlRx” de 8 bits. Dependiendo del valor que toma este registro, se ejecutan diferentes bloques de códigos para realizar las acciones correspondientes.

Es decir, en el caso 0 de “ControlRx”, se realiza una comprobación para determinar si la señal de entrada “oRxOk” está activa, la cual indica la llegada de información desde el software. Si esta condición se cumple, se procede a asignar el valor de los primeros 8 bits de la señal “ORx” al registro de 8 bits denominado “ControlRx”. Esto implica la espera de recibir una palabra de control, es decir, se asigna un valor al registro “ControlRx”. A partir de esta palabra de control, se ejecuta una acción específica según su contenido.

En el caso 1, el valor de “ControlRx” es “1” y se verifica si la señal “oRxOk” está activa. Si es así, se asigna el valor de los primeros 8 bits de la señal “ORx” a otro registro llamado “B_07_00”, que también es un registro de 8 bits. Además, se establece “ControlRx” en 0.

Para los casos 2, 3 y 4 se sigue la misma lógica que el caso 1, pero se asignan los valores de los primeros 8 bits de la señal “ORx” a diferentes registros (B_15_08, B_23_16, B_31_24) según el valor de “ControlRx”. Todos estos registros son de 8 bits. Esto se ve reflejado en el diagrama de la figura 3.13.

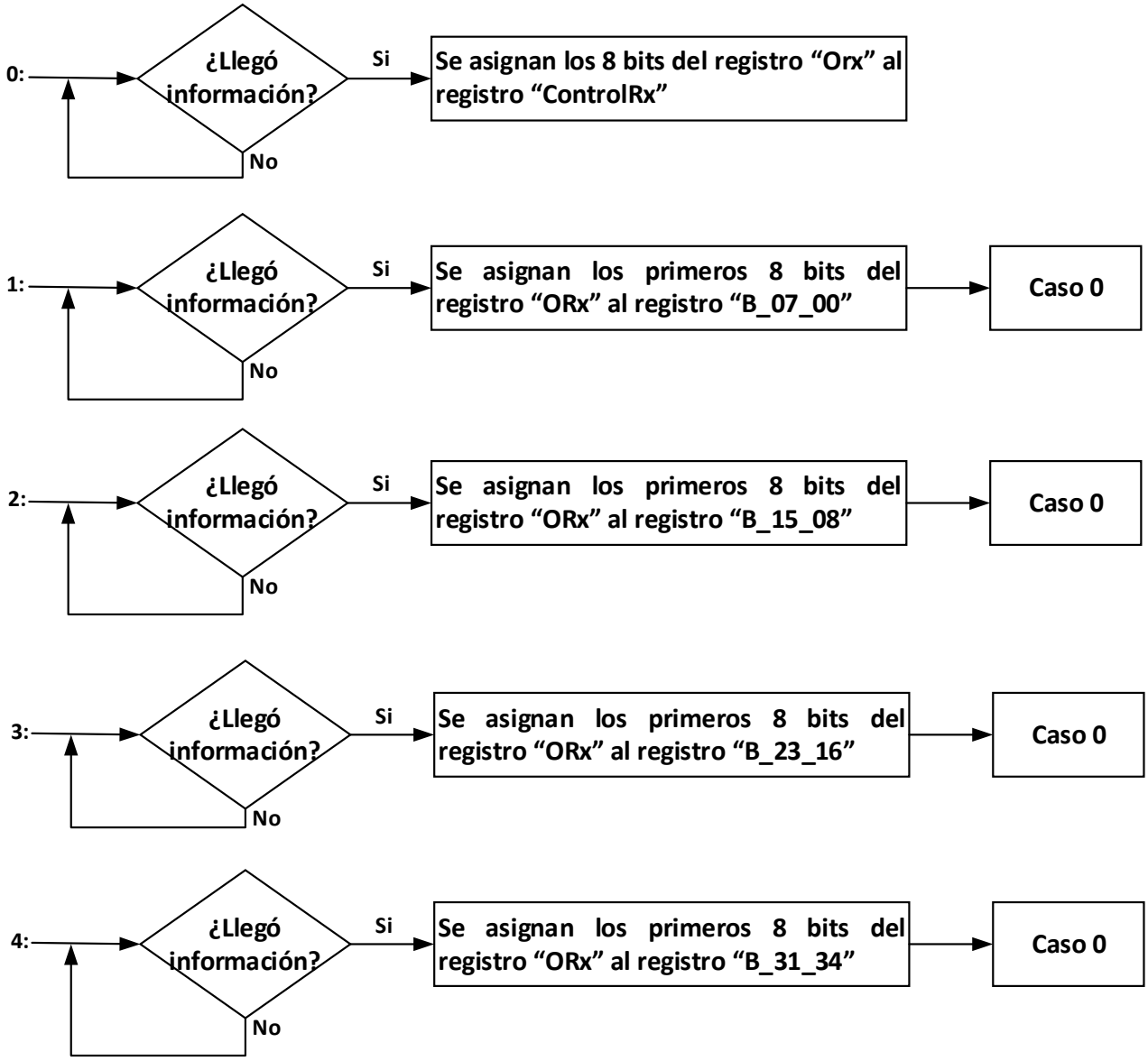


Fig. 3.13 Diagrama del control secuencial del módulo Control Rx

Como se observa en la figura 3.14, cuando la palabra de control es 9, se procede a asignar los valores de los primeros 3 bits del registro “B_07_00”, los cuales representan una parte del retardo deseado. Después de completar estas asignaciones, se establece el valor del registro “ControlRx” en 0, indicando así la finalización del proceso.

Cuando la palabra de control es 10, los 8 bits del registro “B_07_00” se utilizan para configurar los primeros 8 bits del registro “delay”, que representan el retardo principal deseado. Además, los primeros 5 bits del registro “B_15_08” se asignan valores a 5 bits del registro “delay”. Esta asignación permite ajustar el retardo con mayor precisión al agregar una parte fraccionaria o un componente adicional al retardo principal. Una vez que se ha llevado a cabo las asignaciones, se establece el valor del registro “ControlRx” en 0.

Finalmente, cuando la palabra de control es 11, se asigna el valor de los primeros 4 bits del registro “B_07_00” al registro “tw”, que representa el ancho de la ventana de coincidencia. Esto permite configurar el tamaño de la ventana en la cual se busca una coincidencia o se realiza una operación específica. Una vez que se ha llevado a cabo esta asignación, se establece el valor del registro “ControlRx” en 0.

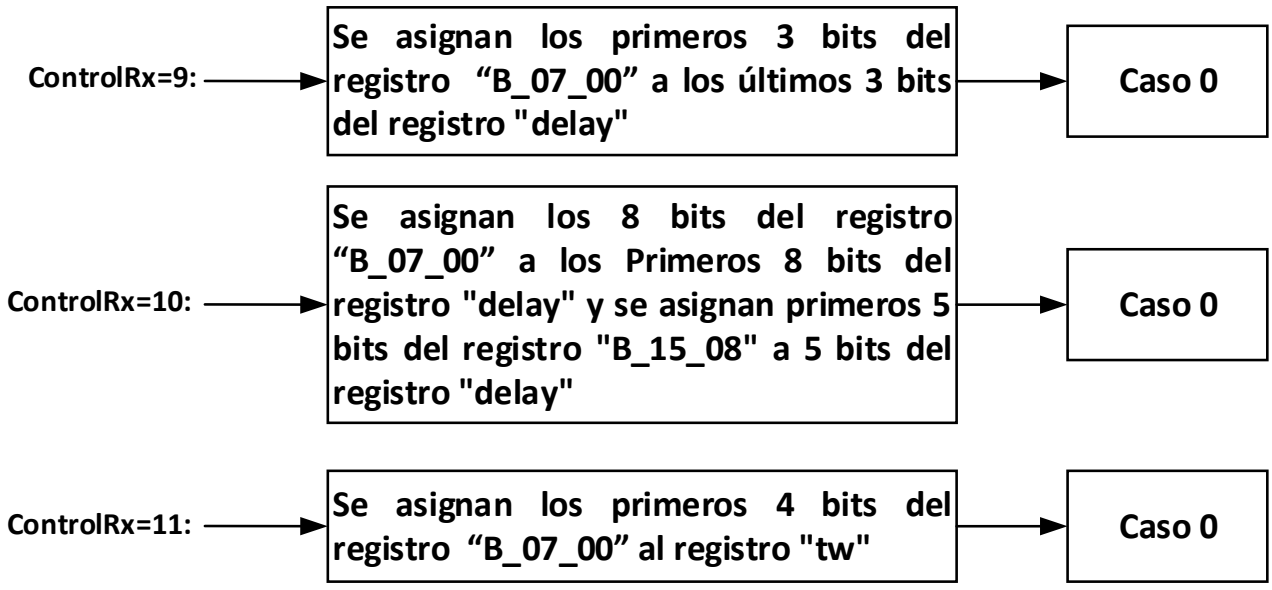


Fig. 3.14 Diagrama del control secuencial del módulo Control Rx

En control Tx tiene como función principal accionar las órdenes recibidas, coordinar la transmisión de datos a través del UART y generar señales de control hacia el software. Este módulo tiene conexión directa con el módulo control Rx. Además, trabaja con un reloj de 100 MHz y cuenta con una memoria interna.

Como se muestra en la figura 3.15, cuenta con módulos internos como EnviarDatos, UART Tx y selector de datos. Además, el módulo presenta varias entradas y salidas importantes. El reloj (CLK) es una entrada que trabaja a una frecuencia de 100 MHz y sincroniza las operaciones internas del módulo. La señal MemC es otra entrada que se conecta directamente al módulo selector de datos y se utiliza para realizar la selección de los datos correspondientes. La señal Pcontrol es una entrada crucial que también se conecta al módulo selector de datos. Esta señal representa una palabra de control que contiene un valor específico utilizado para seleccionar una acción particular a realizar dentro del módulo.

En cuanto a las salidas, el módulo proporciona RsTx, que es la señal de transmisión de datos hacia el software externo. Esta señal se utiliza para enviar datos desde el módulo hacia el software. El módulo también cuenta con la salida COntrolWR, la cual se origina en el módulo Enviar datos y se utiliza para controlar el proceso de escritura/lectura de la memoria. Por último, el módulo presenta la salida Reset, que se utiliza para restablecer o reiniciar el estado de funcionamiento de los componentes dentro del módulo.

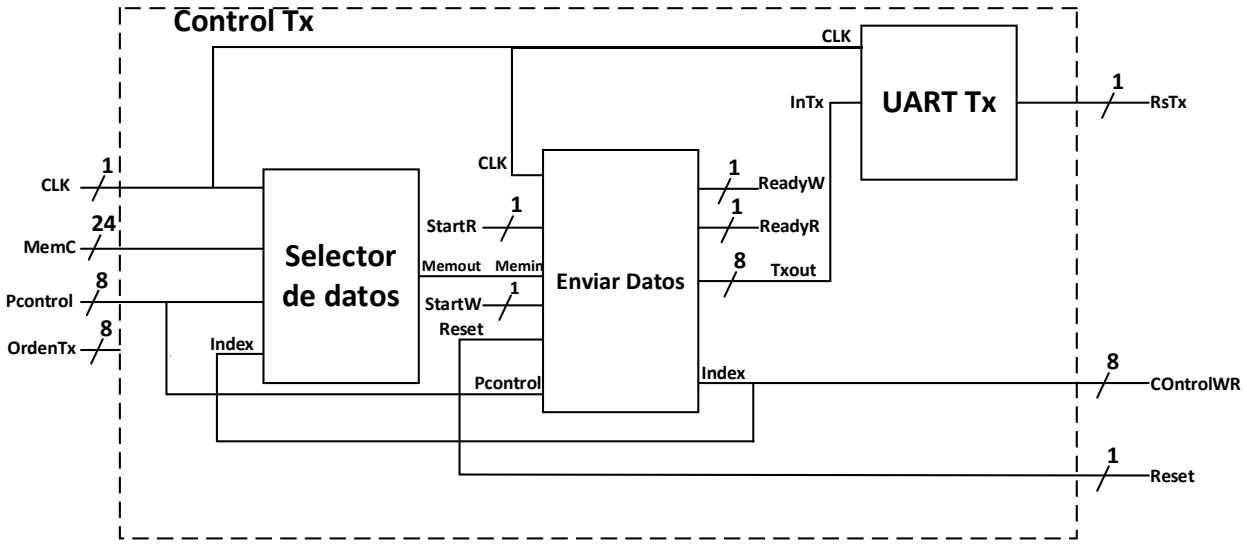


Fig. 3.15 Representación del módulo Control Tx

La operación de este módulo se basa en una máquina de estados finitos para coordinar la transmisión de datos. De acuerdo a la figura 3.16, el módulo comienza en el caso 0, donde se asignan los valores de “OrdenTx” a un índice de control.

Cuando “OrdenTx” sea 10 se va para el caso 10 y empieza la transmisión de datos, donde se activa la señal StartW, la cual indica al módulo “EnviarDatos” que comience a almacenar los datos de las cuentas en la memoria. Si aún no finaliza el almacenamiento de datos en memoria, se avanza al caso 11 para verificar que la operación de almacenamiento haya concluido en el módulo “EnviarDatos”. Si se confirma que el almacenamiento ha finalizado, se procede al caso 12 y se reinicia el sistema por completo.

En el caso 12, se verifica si los datos ya fueron transmitidos por el UART_Tx. Si los datos fueron transmitidos, se avanza al caso 13 y se detiene el estado de transmisión, lo que implica que la transmisión de datos se ha completado. Sin embargo, si los datos aún no están listos, se continúa con la transmisión de datos. Finalmente, en el caso 13, una vez que la transmisión se ha completado, se establece nuevamente en el caso 0, lo que implica reiniciar el estado y prepararse para comenzar un nuevo ciclo de operación.

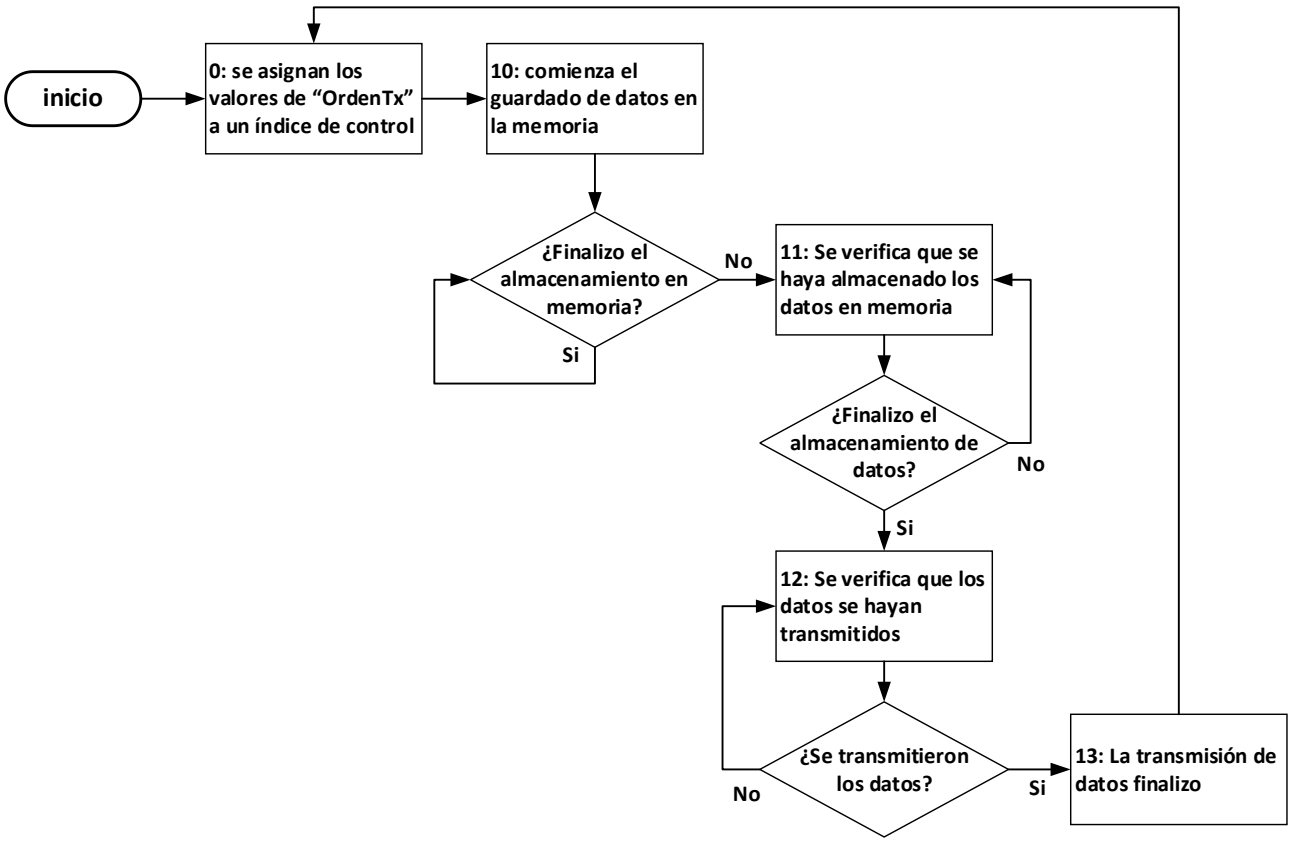


Fig. 3.16 Diagrama de flujo del módulo control Tx

El módulo "EnviarDatos" es una parte interna del módulo "Control Tx" y desempeña un papel fundamental en el control de la escritura y lectura de datos en una memoria, así como en la generación de la salida de datos destinada a ser transmitida hacia el software. Dentro de este módulo, se incluye otro componente llamado "memoria", el cual se encarga específicamente de almacenar los datos provenientes de los contadores simples y dobles.

Como se ve en la figura 3.17 la memoria tiene varias entradas importantes, como "We" que representa la señal de escritura, responsable de guardar los datos en la memoria. La señal "Addr" de 7 bits se encarga de distribuir los datos en las ubicaciones correspondientes dentro de la memoria, mientras que "Din" de 24 bits es la señal a través de la cual ingresan los datos provenientes de los módulos de cuentas.

Además, la memoria cuenta con una salida llamada “Dout” de 24 bits, que se encarga de enviar los datos almacenados hacia el software a través de la interfaz UART, para su posterior procesamiento o visualización.

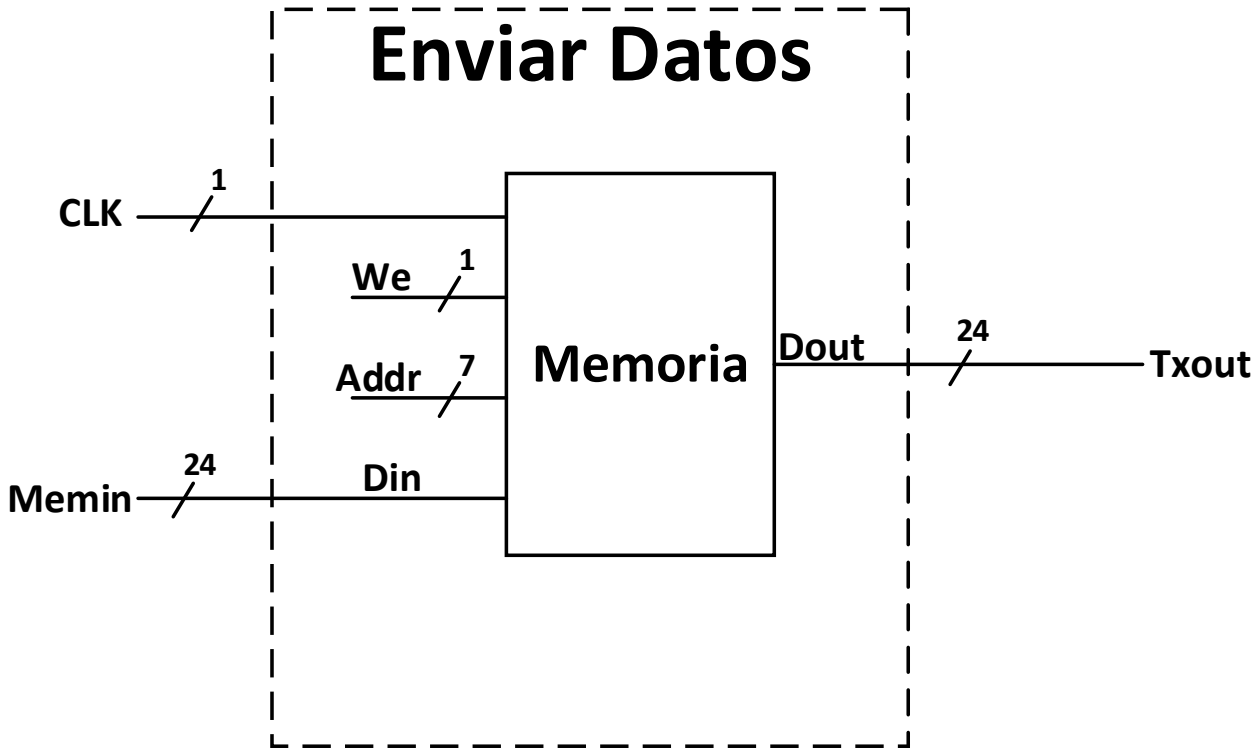


Fig. 3.17 Representación del módulo Enviar Datos

Para transmitir los datos que están almacenados en la memoria, el módulo “EnviarDatos” trabaja con una máquina de estados finitos el cual, en la etapa 0 se recibe una señal para comenzar a guardar los datos extraídos de las cuentas simples y dobles en la memoria, luego se va a la etapa 1 donde se leen los datos que están guardados en los módulos de cuentas y se va para la etapa 2 donde se selecciona las casillas de la memoria en donde se quiere guardar los datos, luego se va a la etapa 3 donde se preparan los datos para ser guardados en la memoria, luego se va a la etapa 4 donde se activa la escritura y se guarda los datos en memoria, luego se va a la etapa 5 donde se verifica si las casillas se llenaron, si las casillas llenadas es igual que la cantidad de datos (en este caso son 80 datos, repartidos en 24 bits) se va a la etapa 10, pero si todavía queda espacio para llenar se devuelve a la etapa 1.

Durante la secuencia de estados de la etapa 10 a la etapa 15, se transmite una máscara a través del UART, la cual contiene valores específicos. Esta máscara tiene como propósito informar al software receptor que se enviará una secuencia de números. En la etapa 10, se envía el valor “0”, seguido de la etapa 11 donde se envía otro “0”. Posteriormente, se procede a la etapa 12 en el cual se transmite el valor “255”. A continuación, se avanza a la etapa 13 donde se envía nuevamente el valor “0”, y seguidamente a la etapa 14 donde se envía el valor “254”. Finalmente, se llega a la etapa 15 donde se transmite la palabra de control mencionada previamente. Una vez completada esta etapa, se pasa a la etapa 20 para continuar con el flujo del sistema.

En la etapa 20 se selecciona la casilla para ser transmitida y se avanza a la etapa 21, donde se transmite el dato de la casilla seleccionada y se avanza a la etapa 22, donde se transmite los primeros 8 bits de los datos, en la etapa 23 se transmiten los siguientes 8 bits de datos y en la etapa 24 se transmiten los últimos 8 bits de un total de 24 bit de datos y se va a la etapa 25 donde se verifican que se transmitieron los 24 bits de información, si es así se va a la etapa 26 de lo contrario se devuelve a la etapa 20 y finalmente en la etapa 26 se finaliza la transmisión y se espera la señal de comienzo de leer los datos de la memoria para volver a la etapa 0. Todo este procedimiento se puede apreciar mejor en la figura 3.18.

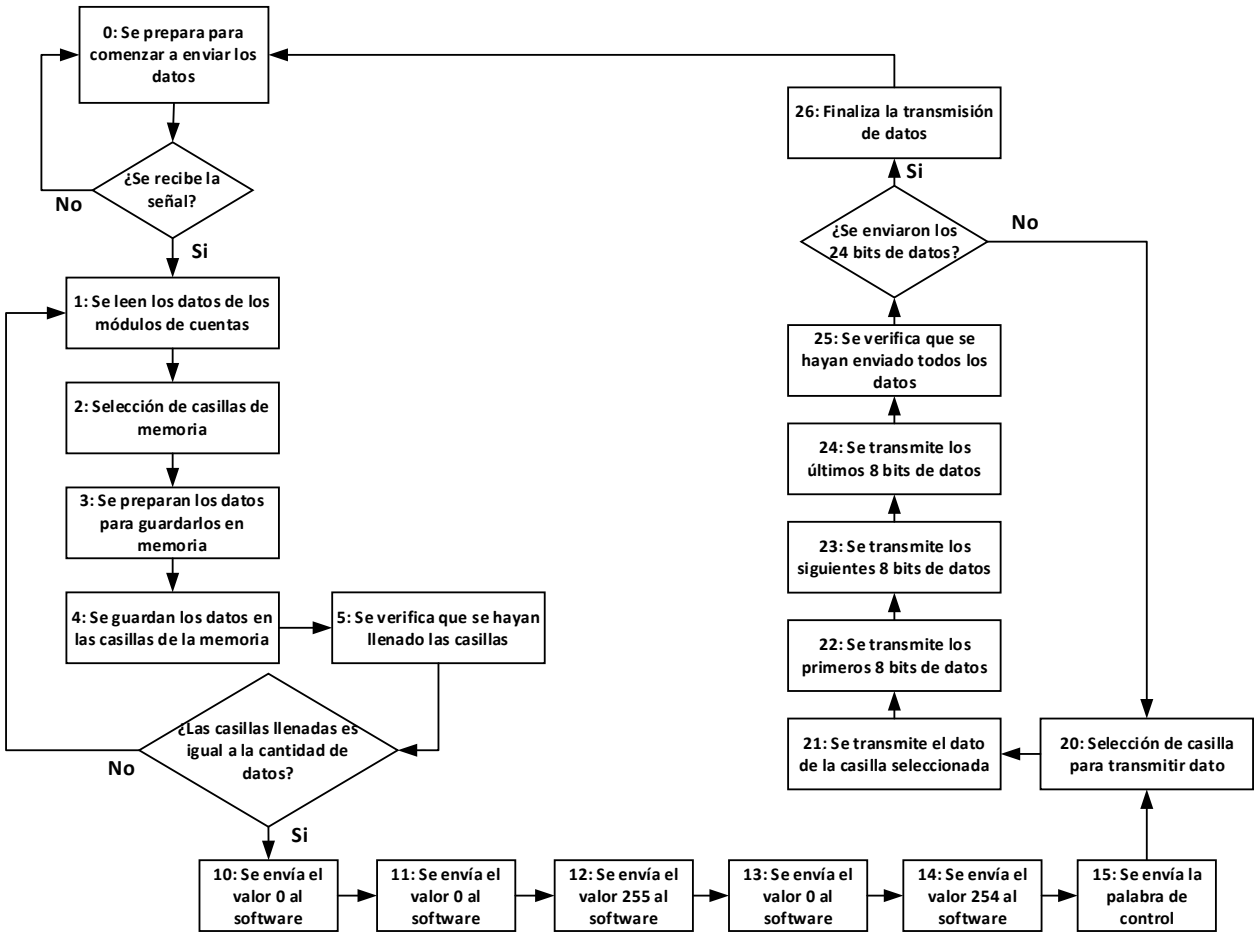


Fig. 3.18 Diagrama de flujo del módulo Enviar Datos

3.2. Explicación de software

El software de control y monitoreo del sistema contador de coincidencias de fotones se desarrolló utilizando la plataforma Visual C#. Para su implementación, se aprovecharon las capacidades de programación orientada a objetos que ofrece esta plataforma.

Se utilizó controladores visuales para crear una interfaz gráfica en el software desarrollado. Para lograr esto, se aprovechó el Toolbox del Window Form design. El Toolbox es una ventana que ofrece una amplia gama de controladores visuales predefinidos, los cuales pueden ser arrastrados y soltados en la ventana principal del software. Los más utilizados en este software son los PictureBox, NumericUpDown, Button, CheckBox, Label, etc. En la figura 3.19 se muestra la interfaz final del software.

Dentro de la plataforma Visual C#, se diseñó el software utilizando el concepto de clases. Estas clases son estructuras de programación que permiten definir objetos con características y comportamientos específicos. En este caso, se crearon clases que representan los diferentes componentes y funcionalidades del software.

Cada clase contiene variables, métodos y propiedades que definen su estado y comportamiento. En este contexto en específico, se puede tener una clase dedicada a la visualización gráfica de cuentas de coincidencia, mientras que otra clase se encarga de almacenar los datos pertinentes, y así sucesivamente. Estas clases interactúan y se comunican entre sí mediante la creación y manipulación de objetos, permitiendo así el funcionamiento y sincronizado del sistema en su conjunto.

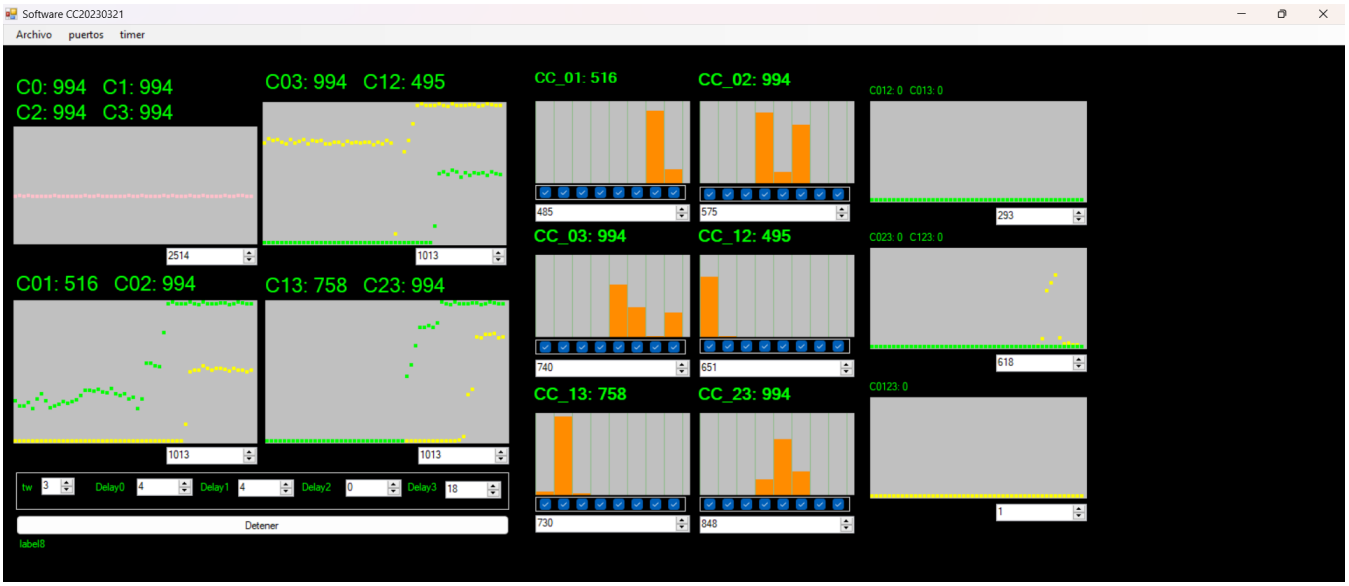


Fig. 3.19 Interfaz del Software contador de coincidencias

3.2.1 Clase puertos

Esta clase se encarga de la interacción con dispositivos conectados a través de puertos seriales, específicamente con el FPGA. Su función principal es facilitar la conexión con el dispositivo, enviar comandos y recibir datos desde el mismo.

Dentro de esta clase, se encuentra el objeto “BuscarPuertos”, el cual se encarga de buscar los puertos seriales disponibles en el sistema y mostrarlos en la interfaz a través de la herramienta “ToolStripComboBox”. Esto permite al usuario seleccionar un puerto para establecer la conexión.

Otros objetos utilizados en esta clase son los siguientes:

- “Conectar”: Establece una conexión con el puerto serial especificado.
- “Desconectar”: Cierra cualquier conexión serial abierta.
- “EnviarString”: Envía una cadena de caracteres al puerto serial.
- “EnviarInt”: Convierte un número entero en un arreglo de bytes y lo envía al puerto serial.
- “RecepcionSerial”: Controlador de eventos que se ejecuta cuando se reciben datos, se encarga de procesarlos y transmitirlos.

Cuando se envía la máscara desde el módulo “Enviar Datos” de Vivado, los datos son recibidos en esta clase, específicamente en “RecepcionData”.

Además, se utiliza la variable “ControlRx”, que es un vector predefinido de 10 valores. Este vector almacena valores numéricos que representan el control de recepción de datos en el puerto serial. Estos valores son recibidos por el objeto “RecepcionSerial” con el objetivo de establecer la cantidad de bytes que deben ser leídos desde el puerto serial, en función del control de recepción actual. Posteriormente, se envían al editor principal de Window Form, donde se determina que se deben leer 240 bytes y se almacenan en la posición 5 del vector “ControlRx” al seleccionar la palabra de control con un valor de 50.

Para controlar la ventana de coincidencia y el retardo a aplicar a los pulsos eléctricos, se utiliza el objeto “EnviarControlValor”. Este objeto se encarga de enviar la palabra de control y un valor específico al puerto serial. En primer lugar, el valor se convierte en un arreglo de 4 bytes (32 bits) y luego se envían los bytes al puerto serial para su procesamiento.

3.2.2 Clase conversor

Cuando los datos son recibidos desde el hardware, llegan en forma de cadenas de 8 bits, lo que equivale a 1 byte y son guardados en un vector. En el caso específico, se reciben tres cadenas de bytes, lo que totaliza 24 bits en total, correspondientes a los datos de las cuentas. Para poder trabajar con estos datos de manera más conveniente, es necesario convertirlos a valores de tipo double ya que es posible realizar cálculos matemáticos, aplicar algoritmos de procesamiento de señales, generar visualizaciones gráficas, entre otras operaciones.

Esta clase se utiliza para convertir un arreglo de bytes en un arreglo de valores double. Para realizar esta conversión, se utiliza el objeto “byte2double”. Este objeto requiere como parámetros el arreglo de bytes, un índice inicial, un índice final y el número de bytes que se deben considerar para formar un valor double. Luego, calcula el tamaño del arreglo de salida, crea un arreglo de valores double con ese tamaño y realiza la conversión de cada secuencia de bytes en un valor double utilizando las potencias correspondientes. Finalmente, devuelve el arreglo resultante de valores double. De esta manera, se logra obtener una representación más adecuada de los datos y facilitar su posterior procesamiento y análisis.

3.2.3 Clase graficar puntos

Esta clase se utiliza para generar gráficas de puntos dentro un elemento llamado PictureBox de Windows Forms design, en la figura 3.20 se muestra un ejemplo de este tipo de grafica. En la aplicación, se muestra un gráfico individual para cada canal de cuentas simples, lo que permite visualizar la evolución de cada canal de forma independiente. Además, se generan gráficos adicionales que representan las combinaciones de canales, mostrando las cuentas coincidentes dobles. Estos gráficos combinados ofrecen una perspectiva más completa de las coincidencias entre los diferentes canales, brindando información adicional sobre las interacciones y correlaciones entre ellos.

El objeto principal de esta clase, para que se pueda apreciar gráficamente las cuentas, es “graficarNentradas”. Este objeto se encarga de generar una representación gráfica en tiempo real de los datos de entrada en forma de puntos en un PictureBox. Los puntos se desplazan horizontalmente a medida que se reciben nuevos datos, creando una animación visual de la evolución de los valores.

Para generar este gráfico, se utiliza el método que recibe como parámetros un PictureBox donde se mostrará la gráfica, un arreglo de datos de tipo double, el número de gráficos a mostrar y un valor máximo de índice. Los datos de entrada se almacenan en una matriz que tiene capacidad para 10 gráficos con 1000 puntos cada uno. Se lleva un seguimiento del índice actual de la matriz, y si este índice es menor que el ancho del PictureBox, se incrementa permitiendo avanzar horizontalmente en la generación de la gráfica. Si el índice alcanza o supera el ancho del PictureBox, se realiza un desplazamiento de los datos en la matriz hacia la izquierda, creando espacio para los nuevos datos en la parte derecha de la gráfica. Además, se calcula el valor máximo de los datos de entrada y se actualiza el valor máximo de índice si es necesario.

En cuanto a la visualización de la gráfica, se crea un objeto Bitmap llamado “imagen_fondo” con el tamaño del PictureBox. Se itera sobre los puntos de la gráfica, generando una representación visual para cada uno de ellos. La matriz se utiliza para obtener las coordenadas de los puntos a dibujar en la imagen. Para cada punto, se determina su posición vertical en la imagen en función del valor del dato y el valor máximo de índice. A su vez, se utiliza el objeto “PonerColor” para asignar un color específico a cada gráfico. Finalmente, la imagen generada se asigna al PictureBox pasado como parámetro y se devuelve el valor máximo del índice actualizado.

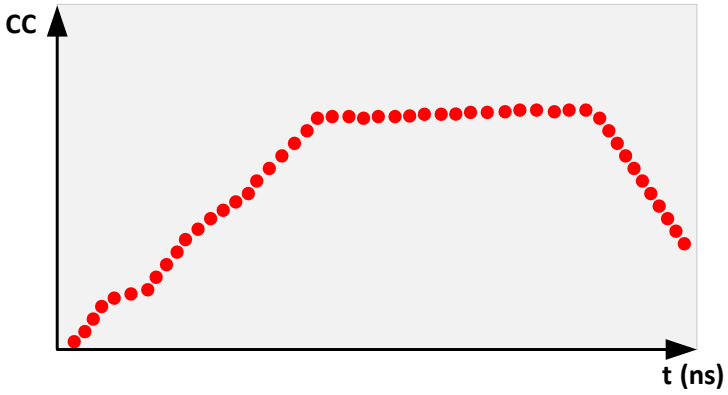


Fig. 3.20 Ejemplo del gráfico de puntos

3.2.4 Clase graficar histograma

Esta clase se utiliza para generar y visualizar histogramas gráficos. Permite representar la distribución de datos en forma de barras verticales en un control PictureBox, en la figura 3.21 se muestra como ejemplo el grafico de histogramas.

Dentro de esta clase se utiliza el objeto “graficarHisto”, el cual recibe varios parámetros, incluyendo PictureBox donde se mostrará el histograma, un valor máximo especificado mediante un control NumericUpDown, un arreglo de datos “dataCC”, el número de bins del histograma “Nbins” y un arreglo “BinOn” que indica qué bins están habilitados. Los bins corresponden a las divisiones o intervalos en los que se agrupan los datos para construir el histograma, es decir son diferentes Δt que se pueden tomar. Cada bin representa la frecuencia o cuentas de los datos que caen dentro de ese bin en particular. En otras palabras, el bin captura cuántas veces los datos se encuentran en el rango específico correspondiente a ese bin, en este caso estos valores son las cuentas de coincidencia entre canales. Esto proporciona información sobre la distribución de los datos y cómo se agrupan en diferentes rangos

Para generar el histograma, el objeto realiza cálculos para determinar el ancho de cada bin en función del tamaño del PictureBox y la cantidad de bins especificada. Además, crea una imagen de fondo con el tamaño adecuado para representar el histograma. Durante el proceso de iteración sobre los bins, el método verifica si el bin está habilitado y, en caso afirmativo, suma el valor correspondiente del arreglo “dataCC” a la variable “Cuentas”. A continuación, se recorre cada bin y se calcula la altura de la barra del histograma en función del valor del bin y el valor máximo especificado. Si el valor del bin supera el máximo establecido, se ajusta dicho máximo si la bandera “ajustarMax” está activada; de lo contrario, se establece la altura de la barra en la parte superior del PictureBox.

Posteriormente, se procede a dibujar la barra del histograma en la imagen de fondo, asignando colores diferentes según si el bin está habilitado o no. Finalmente, se asigna la imagen de fondo al PictureBox y se devuelve el valor máximo actualizado.

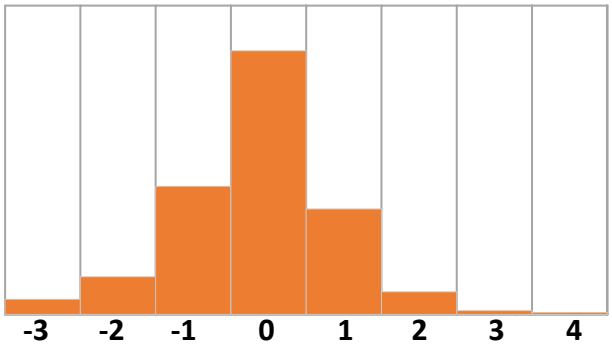


Fig. 3.21 Ejemplo del gráfico histograma

3.2.5 Clase Bin Activos

La clase binActivos se emplea para gestionar la interacción y el control de un conjunto de casillas de verificación (CheckBox) que actúan como indicadores de valores binarios. Cada CheckBox representa una columna en el gráfico de histograma, permitiendo seleccionar los bins que se encuentran realizando el recuento de datos, como se muestra en la figura 3.22. Esta clase proporciona métodos que permiten llevar a cabo conversiones entre la representación binaria y decimal de estos valores, además de facilitar el almacenamiento y la carga de los mismos en un archivo. En definitiva, el uso de esta clase permite una selección precisa de los bins que están siendo considerados en el recuento de datos, lo que contribuye a obtener resultados más precisos.

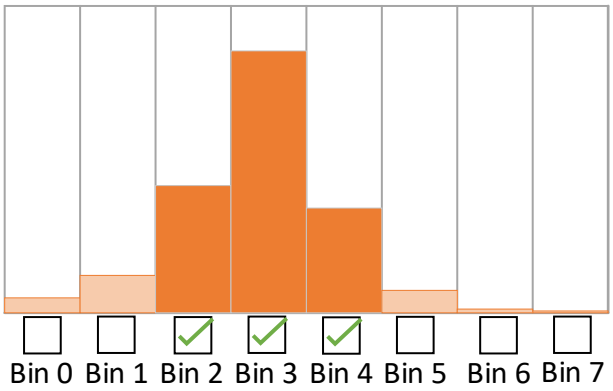


Fig. 3.22 Ejemplo gráfico de la interacción con los Bins

3.2.6 Clase Guardar Data

Esta clase se encarga de implementar funcionalidades para el almacenamiento y guardado de datos. A través de sus métodos, es posible guardar matrices de valores en archivos de texto y también insertar líneas de datos en archivos existentes.

Entre los objetos utilizados se encuentra “SeleccionarCarpeta”, el cual permite al usuario seleccionar la carpeta de destino para guardar los archivos. Este objeto retorna un valor booleano que indica si la selección de la carpeta fue exitosa.

El objeto “GuardarVector_InsertarLinea” se utiliza para almacenar un vector de valores en una nueva línea de un archivo existente. Recibe como parámetros el vector de valores y los guarda utilizando una representación decimal que emplea el símbolo de coma o punto, según se requiera.

El objeto “NombreArchivo” es una variable de tipo String que almacena el nombre del archivo en el cual se desean guardar los datos.

La variable de tipo String “cabecera” almacena una línea de encabezado que describe el contenido de los datos que se van a guardar en el archivo. Esta línea de encabezado generalmente se coloca al comienzo del archivo y brinda información sobre el significado o la estructura de los datos presentes en las líneas siguientes.

El objeto “GuardarVector_conTiempo” permite almacenar un vector de valores junto con un valor de tiempo asociado en una nueva línea del archivo.

Además, en esta aplicación se especifica la ubicación en la que se guardarán los datos. Esta información se obtiene mediante la variable “DirCarpeta”, que es un atributo de esta clase y almacena la dirección de la carpeta de destino para el almacenamiento de los archivos.

3.2.7 Clase Timing

La clase timing ofrece funcionalidades relacionadas con la medición y manipulación del tiempo, permitiendo establecer una referencia temporal para los datos obtenidos. Para este propósito, hace uso de la variable “tiempoAcc”, un atributo de tipo double que almacena el tiempo acumulado.

La clase proporciona el objeto “TiempoReset”, el cual reinicia el tiempo acumulado desde que el software está en funcionamiento. Esto resulta útil para obtener datos a partir de un punto específico en el tiempo. Además, el método “timeNow” permite obtener el tiempo transcurrido en segundos desde el último reinicio de tiempo, proporcionando así una medida actualizada del tiempo transcurrido. Para crear un nombre de archivo que contenga información sobre la fecha y hora actual se debe usar el método “nametime”. Este método utiliza los objetos proporcionados por la clase “System.DateTime.Now” para obtener la hora, minutos, segundos, año, mes y día actuales. Luego, combina estos valores en un formato específico, generalmente en el formato “AAAAMMDD_HHMMSS”, donde “AAAAMMDD” representa la fecha en formato año, mes y día, y “HHMMSS” representa la hora, minutos y segundos. El objetivo de este método es proporcionar un nombre de archivo único que incluya la marca de tiempo actual, lo cual es útil para guardar y organizar datos o registros con identificadores únicos basados en el tiempo.

3.2.8 Clase Propiedades

La clase Propiedades ofrece métodos para cargar, guardar y manipular propiedades del software, proporcionando así un mecanismo para almacenar y recuperar configuraciones específicas. Estos métodos permiten guardar las propiedades en un archivo persistente, lo que garantiza que los parámetros se conserven incluso después de cerrar y volver a abrir el software. Al utilizar esta clase, se brinda la capacidad de mantener la configuración personalizada y el estado del software.

Esta clase utiliza métodos como “Leer_propierdades” el cual, lee las propiedades almacenadas en un archivo llamado “propiedades.txt”. Si el archivo no existe, se genera y se inicializan todas las propiedades con un valor predeterminado.

Para guardar las propiedades se utiliza “Guardar_propiedad” el cual permite guardar una propiedad especificada por su nombre y valor. Busca en la matriz de propiedades si ya existe una propiedad con ese nombre. Si existe, actualiza su valor; de lo contrario, crea una nueva propiedad. Luego, guarda todas las propiedades en el archivo “propiedades.txt”.

Para cargar los valores guardados en el NumericUpDown (para los casos de ventana de coincidencia y retardo) se utiliza el método “cargar_numericUpDown” que carga un valor en un objeto NumericUpDown.

3.3. Montaje del sistema en caja

Para realizar el ensamblaje en una carcasa metálica, se debe soldar previamente los pines correspondientes del Breakout, a fin de establecer una comunicación directa con el FPGA.

Es importante tener en cuenta que la placa FMC Breakout cuenta con una conexión FMC LPC con la matriz de pads para acceder de manera general a las señales ANSI/VITA 57.1, tal como se muestra en la figura 3.23.

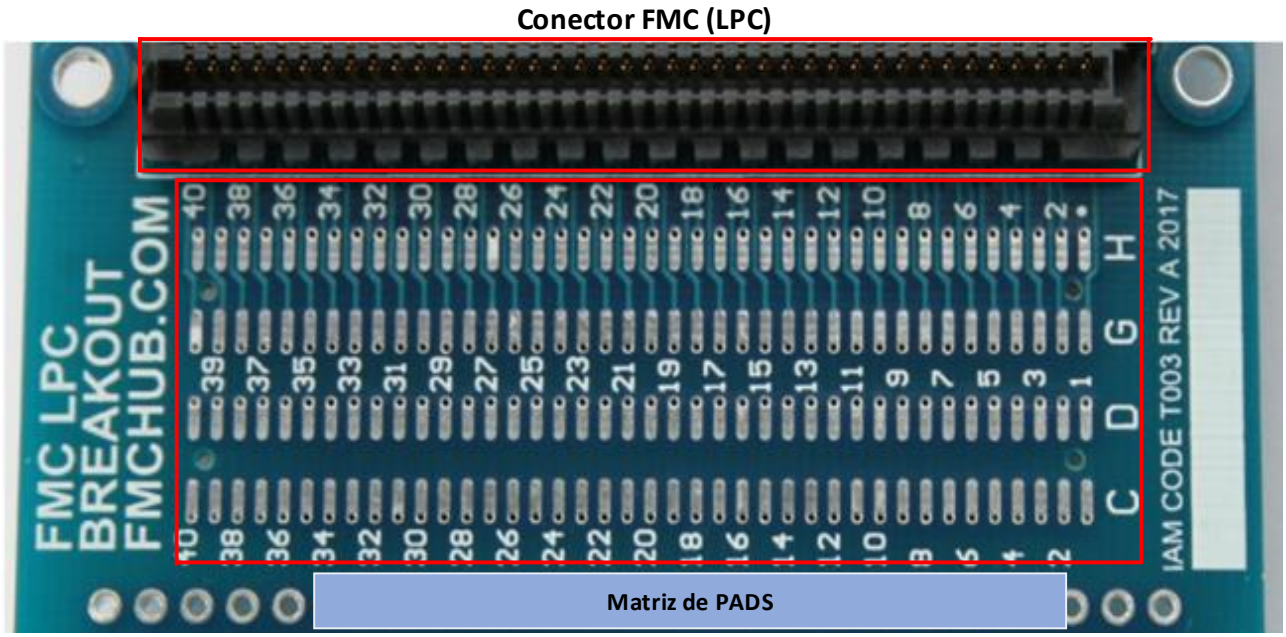


Fig. 3.23 Matriz de pads de la placa Breakout.

Por lo tanto, al realizar la soldadura, se debe considerar la ubicación de las señales ANSI/VITA 57.1 en dicha matriz de pads. Para ello, se puede hacer referencia a la tabla 3.1 que indica la correspondencia entre las señales y los pines respectivos. Es importante destacar que las señales presentes en el FMC del Breakout son iguales a las presentes en el FPGA.

Tabla 3.1: Señales del conector ANSI/VITA 57.1 FMC LPC y matriz de pads.

Canal	Pin #	Fila	ANSI/VITA 57.1
Ch1	10	C	LA06 P
	9	C	GND
Ch2	14	C	LA10 P
	13	C	GND
Ch3	26	C	LA27 P
	25	C	GND
Ch4	38	H	LA32 N
	39	H	GND

En la figura 3.24 se muestra la conexión establecida mediante la soldadura de los conectores SMA (SubMiniature version A), los cuales se enlazan a los pines correspondientes de la placa FMC Breakout.

Los conectores SMA son pequeños conectores coaxiales de RF (Radio Frecuencia). Tienen una impedancia de 50 Ω. Estos conectores utilizan un mecanismo de acoplamiento tipo tornillo que minimiza reflejos y atenuación, lo que los hace ampliamente utilizados en aplicaciones de antena para tecnologías de menos de 6 GHz, como Wi-Fi y Bluetooth [32]. Estos conectores SMA representan los cuatro canales presentes en el sistema.

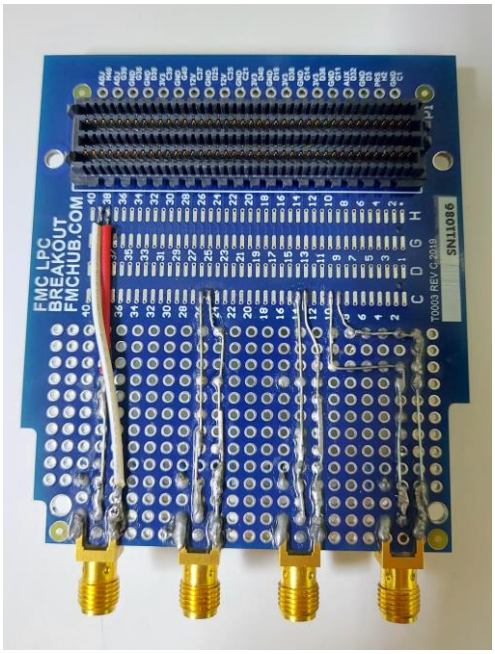


Fig. 3.24 Placa Breakout soldada

Ya con el Breakout preparado, se conecta por medio del FMC al FPGA, este FPGA se ensambla a la base de una caja metálica para darle firmeza al sistema. Los conectores SMA se conecta directamente al Breakout y salen por un costado de la caja, el usuario podrá manipular a su gusto sin que se suelte estos conectores. Este ensamblaje final queda demostrado en la figura 3.25.

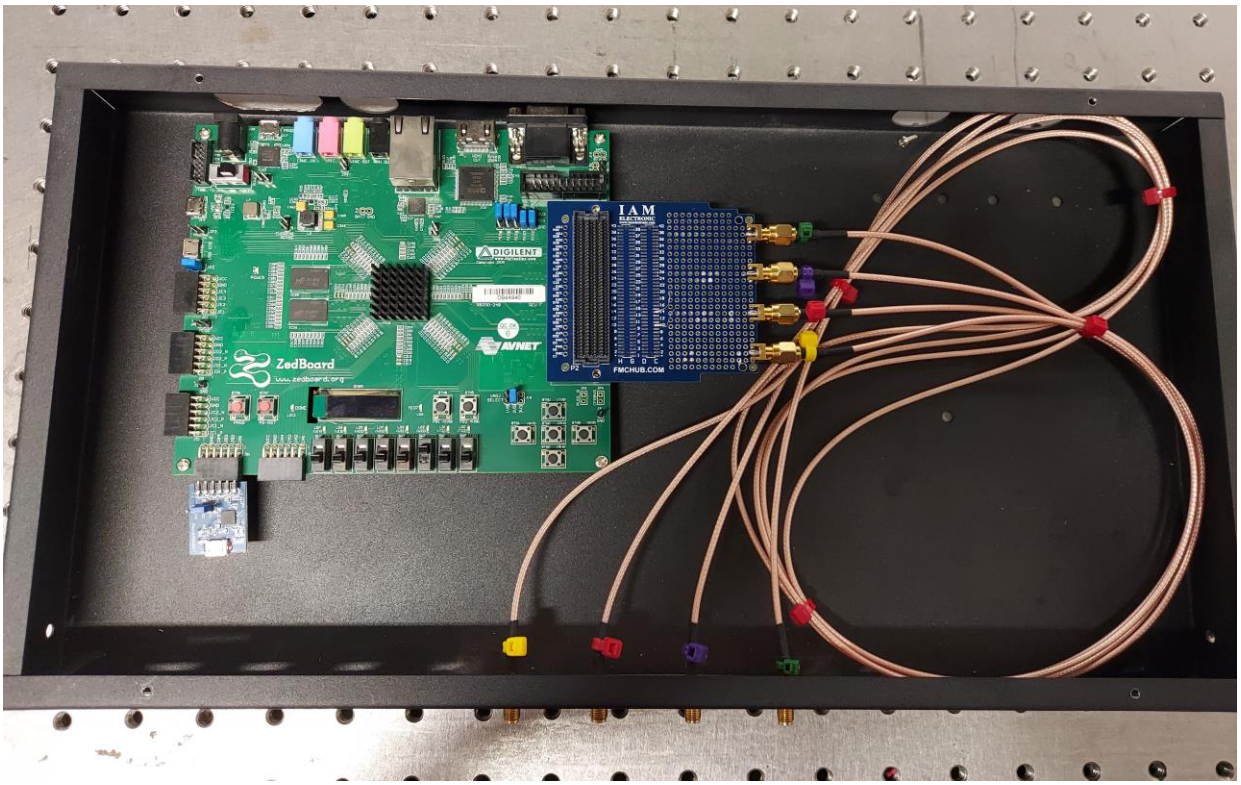


Fig. 3.25 Sistema ensamblado en caja metálica

Capítulo 4. Caracterización del sistema

4.1. Configuración del Set-Up utilizado

Una vez ensamblado el sistema en la caja metálica, se procedió a llevar a cabo la evaluación del sistema, mediante la utilización del generador de funciones Tektronix AFG3021B.

Para realizar la medición, primero se debió configurar el generador de funciones el cual se requirió generar un pulso con las siguientes características:

- Frecuencia 1 KHz.
- Voltaje máximo 1.650 V.
- Voltaje mínimo 0 V.
- Ancho del pulso 100 ns.

Con estos parámetros, se generó un pulso junto con una señal TTL para simular la representación de dos pulsos, que a su vez representaban a los fotones. Estos pulsos se visualizaron en el osciloscopio. En la figura 4.1 se observa el Osciloscopio (1) y Generador de funciones (2).

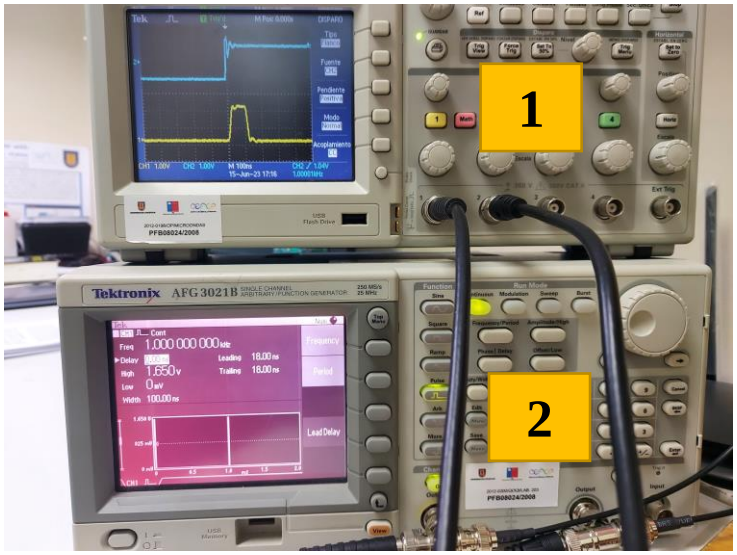
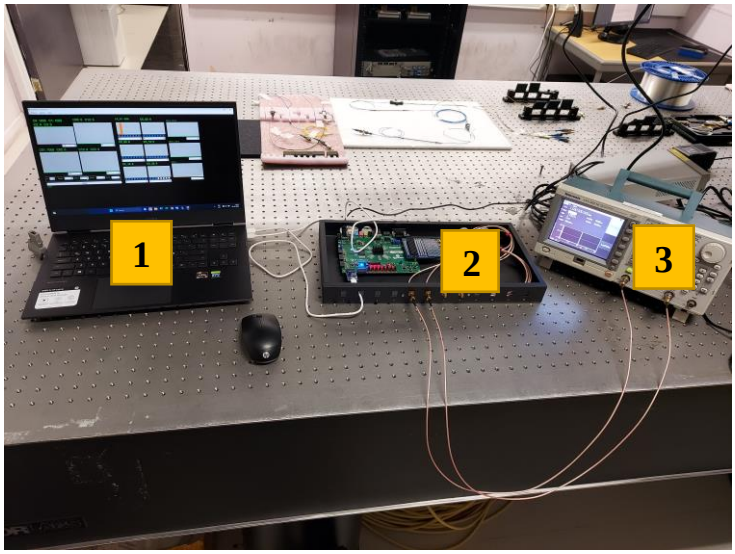
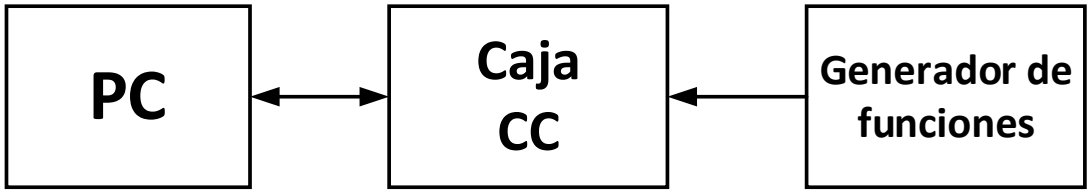


Fig. 4.1 Configuración utilizada al generador de funciones

Una vez configurado el generador de funciones, se preparó el Set-Up completo, como se muestra en la figura 4.2.a, donde se observa el PC (1), el sistema ensamblado (2) y el generador de funciones (3). En la figura 4.2.b se presenta el diagrama de este Set-Up.



a)



b)

Fig. 4.2 Set-Up del sistema completo

(a) Set-Up del sistema en físico, **(b)** Diagrama del Set-up del sistema.

La caracterización se llevó a cabo utilizando dos canales que se combinaban entre los cuatro disponibles. Se ajustaba el comando Delay del generador de funciones, lo que resultaba en un desplazamiento en el tiempo del pulso generado en relación al TTL. A medida que se incrementaba este Delay, se observaba un aumento en el número de cuentas coincidentes, llegando a un máximo de 1000 cuentas, para luego disminuir gradualmente hasta alcanzar 0 cuentas. Se realizaron mediciones para diferentes anchuras de ventana de coincidencia, que fueron de 1, 2, 4, 8 y 16 ns.

4.2. Evaluación de ventana de coincidencia

Para la evaluación del sistema, se tiene que definir la ventana de coincidencia (τ) como una función de la resolución de muestreo t_r, que viene de la frecuencia con la que se muestrea el pulso, es decir, 1 GHz (1 ns), de la forma:

$$\tau = mt_r + d_w \tag{3}$$

Donde m es el numero de ranuras que se encuentran contenidas en una ventana, mientras que d_w representa el ruido total de Skew durante un intervalo de tiempo de resolución de muestreo t_r, como se muestra en la figura 4.3.

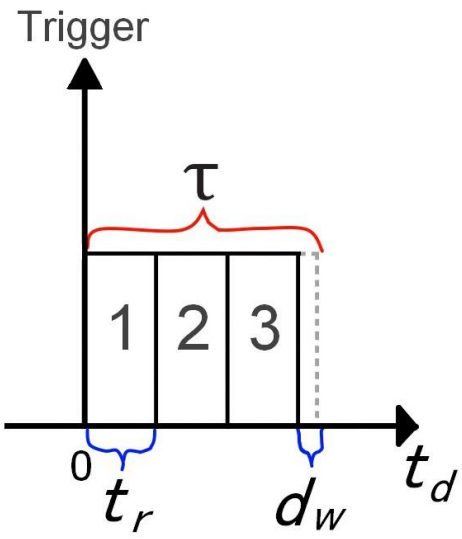


Fig. 4.3 Parámetros de ventana de coincidencia ($w(t_d)$)

Es factible establecer una función que opere cuando la ventana de coincidencia se encuentra habilitada, tomando como base el tiempo de retraso t_d, dado por:

$$w(t_d)\begin{cases}1 & si\ 0 \le t_d \le \tau \\ 0 & e.o.c.\end{cases} \tag{4}$$

Por otro lado, se puede expresar como una función de un retraso temporal t_d. En este contexto, A eventos de detección mostrarán una variación Gaussiana $(g(t))$ con una desviación estándar σ y amplitud A', lo cual se expresa de la siguiente manera:

$$g(t_d) = A' e^{-\frac{(t_d)^2}{2\sigma^2}} \quad \textbf{(5)}$$

Entonces, se define como una cuenta coincidente T_{CC}, cuando $g(t_d)$ está dentro de la función de ventana de coincidencia $w(t_d)$, como la siguiente convolución entre las dos funciones, para todo t_d:

$$T_{CC} = \int_{-\infty}^{\infty} w(t) \cdot g(t_d - t) dt \quad \textbf{(6)}$$

La convolución expresada en la ecuación 6 se puede realizar experimentalmente usando un generador de funciones que sea capaz de retrasar dos pulsos. Por lo tanto, para evitar resolver una integral Gaussiana, se hace un simple cambio de variables obteniendo $T_{CC} = -\int_{t_d}^{t_d-\tau} g(u)du$. Diferenciando con respecto a t_d, se obtiene:

$$\frac{dT_{CC}}{dt_d} = A' \left\{ e^{-\frac{(t_d)^2}{2\sigma^2}} - e^{-\frac{(t_d-\tau)^2}{2\sigma^2}} \right\} \quad \textbf{(7)}$$

Las expresiones de la ecuación 7 y la ecuación 3, permite obtener experimentalmente los valores de τ, σ, t_r y d_w.

Utilizando el generador de funciones se obtiene T_{CC} para las diferentes ventanas de coincidencias, obteniendo las siguientes graficas:

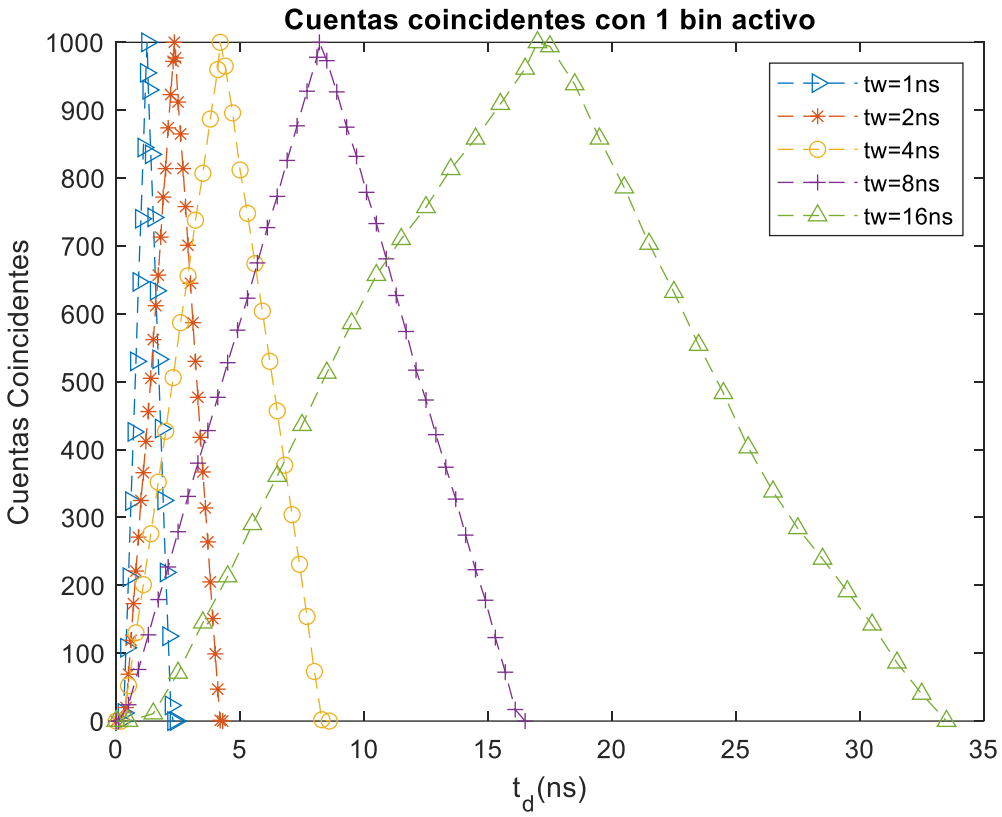


Fig. 4.4 Mediciones experimentales de varios anchos de τ utilizando un bin activo

Como se muestra en la figura 4.4, estas mediciones se llevaron a cabo considerando únicamente un bin activo del software, lo que representa el tamaño real de cada ventana de coincidencia.

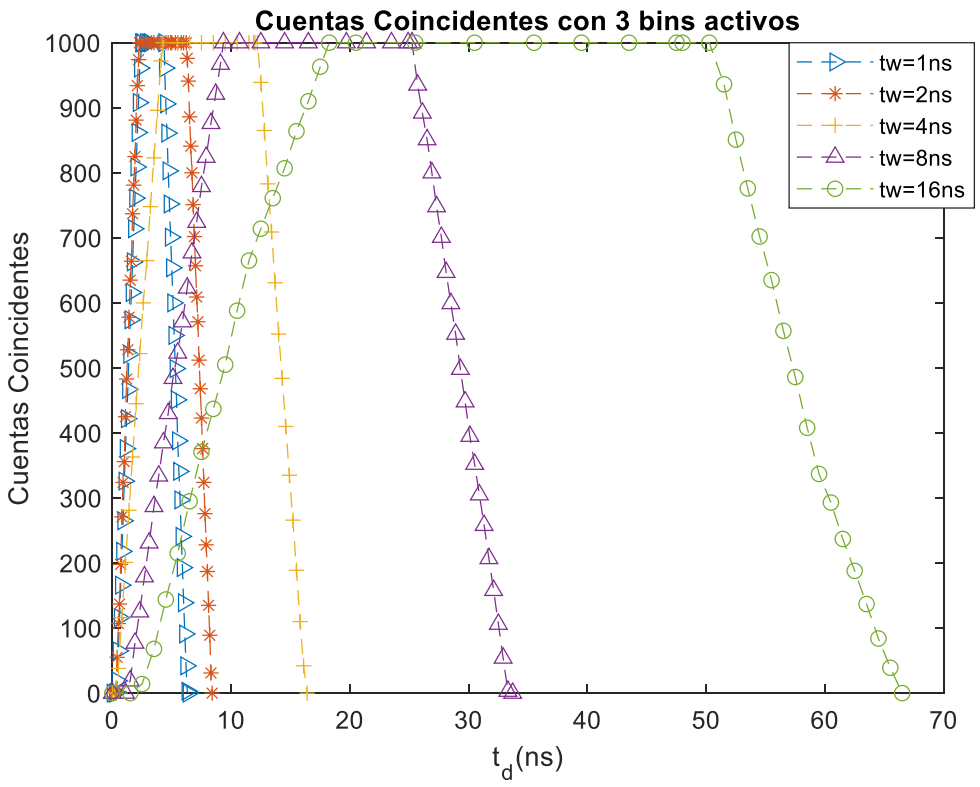


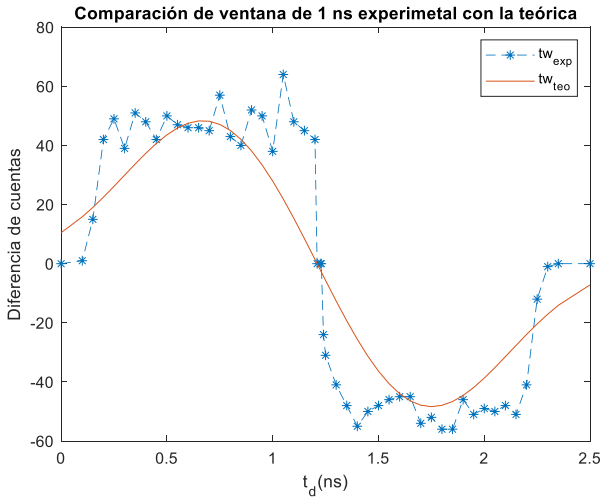
Fig. 4.5 Mediciones experimentales de varios anchos de τ utilizando tres bins activos

En la figura 4.5, se presentan las cuentas coincidentes medidas teniendo en cuenta tres bins activos, lo que significa que el ancho de cada ventana de coincidencia se ha ampliado tres veces su valor original.

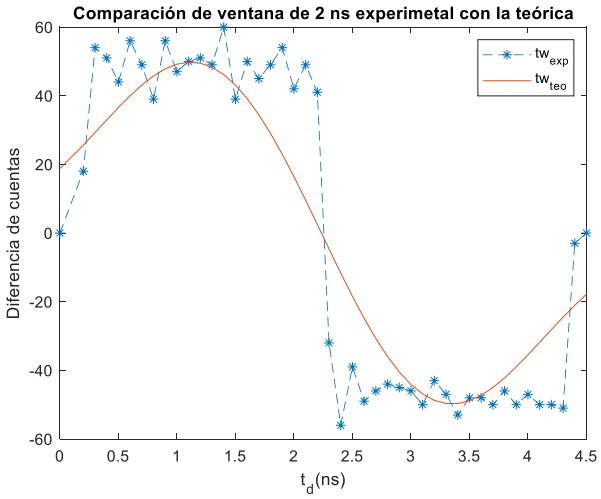
En ambos casos, se varió el Delay entre los dos pulsos eléctricos para las distintas ventanas de coincidencia. Para las ventanas de 1 ns y 2 ns, se modificó cada 100 ps; en el caso de la ventana de 4 ns, se ajustó cada 300 ps; para la ventana de 8 ns, se cambió cada 400 ps; y para la ventana de 16 ns, se ajustó cada 1000 ps.

Además, se puede observar que a medida se va aumentando el tamaño de la ventana de coincidencia, estos aumentan su tiempo de muestreo.

Por otro lado, para medir los parámetros mencionados anteriormente se diferencia los resultados de las gráficas anteriores y se ajusta los datos con la expresión de la ecuación 7 obteniendo los siguientes resultados:



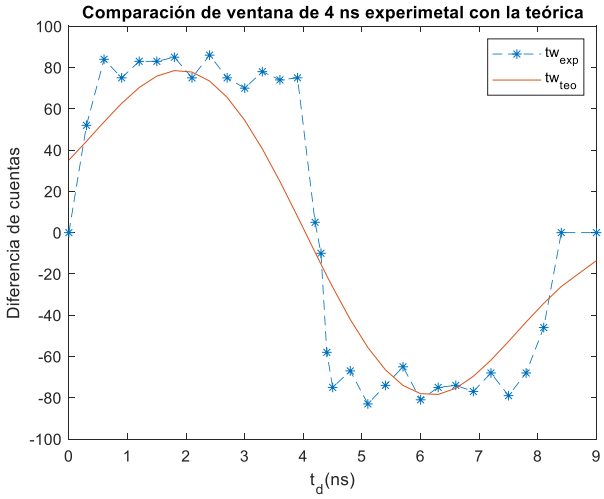
a)



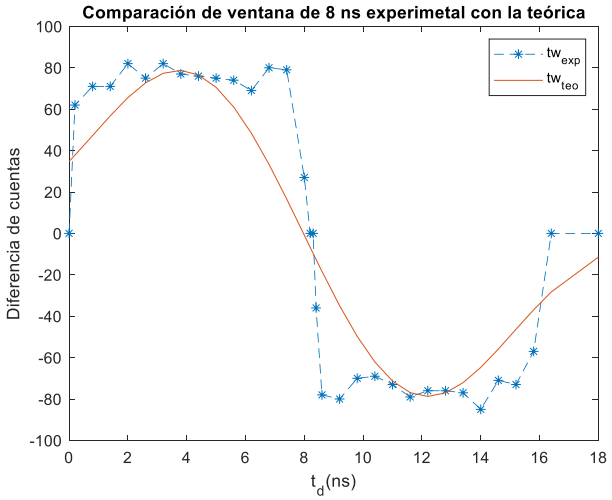
b)

Fig. 4.6 Ajuste de la ventana de coincidencia teórica a la experimental con la ecuación 7 aplicando un bin activo

(a) Ventana de coincidencia de 1 ns, **(b)** Ventana de coincidencia de 2 ns.



a)



b)

Fig. 4.7 Ajuste de la ventana de coincidencia teórica a la experimental con la ecuación 7 aplicando un bin activo

(a) Ventana de coincidencia de 4 ns, **(b)** Ventana de coincidencia de 8 ns.

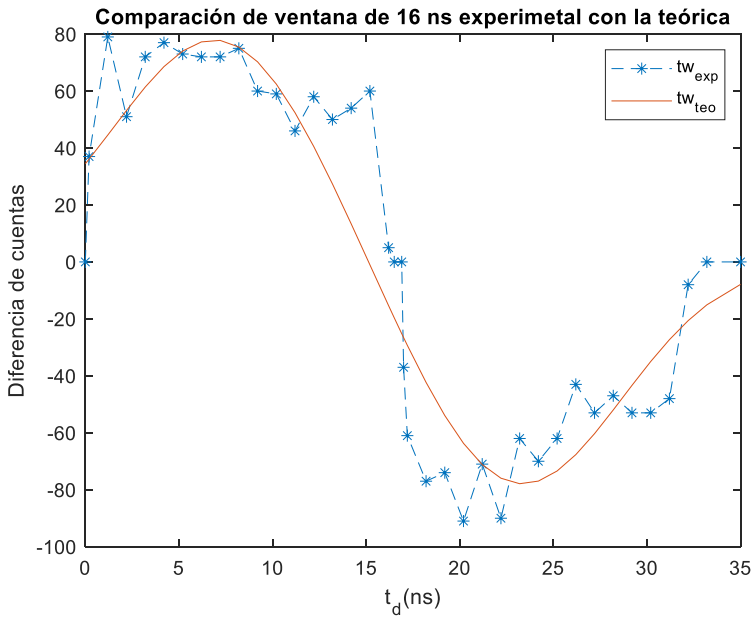
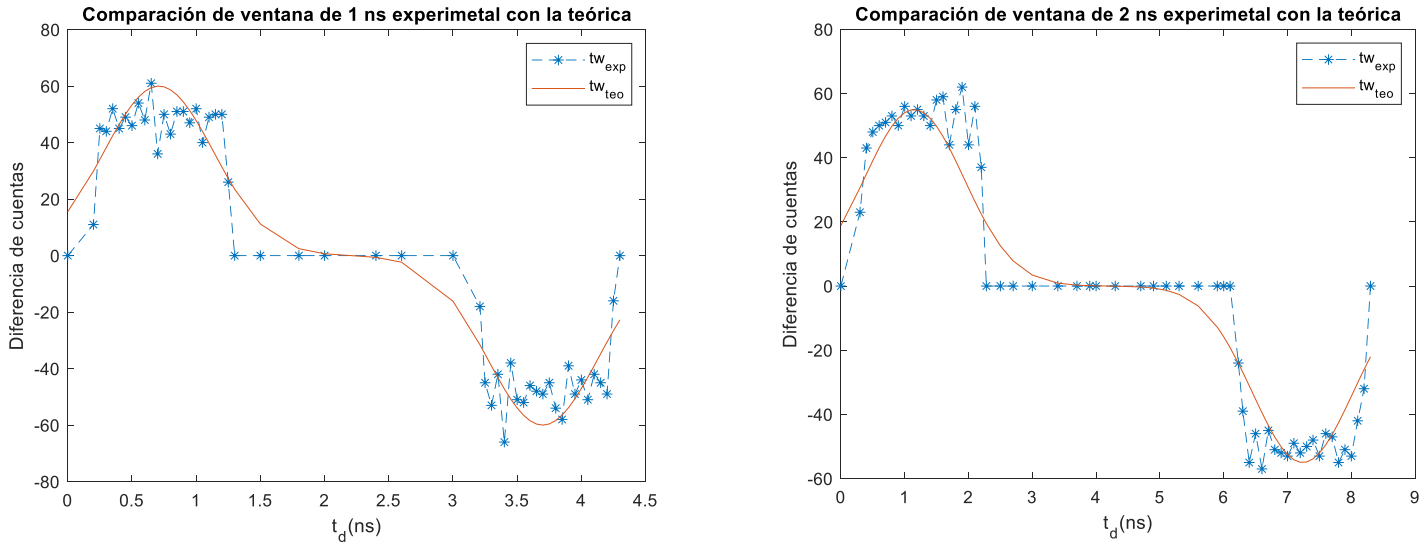


Fig. 4.8 Ajuste de la ventana de coincidencia de 16 ns teórica a la experimental con la ecuación 7 aplicando un bin activo

Es importante señalar que las gráficas presentadas en las figuras 4.6, 4.7 y 4.8 se generaron aplicando la diferencia a las cuentas coincidentes medidas que se realizó con un solo bin activo (como se muestra en la figura 4.4) y utilizando la ecuación 7.



a) **b)**

Fig. 4.9 Ajuste de la ventana de coincidencia teórica a la experimental con la ecuación 7 aplicando tres bins activos **(a)** Ventana de coincidencia de 1 ns, **(b)** Ventana de coincidencia de 2 ns.

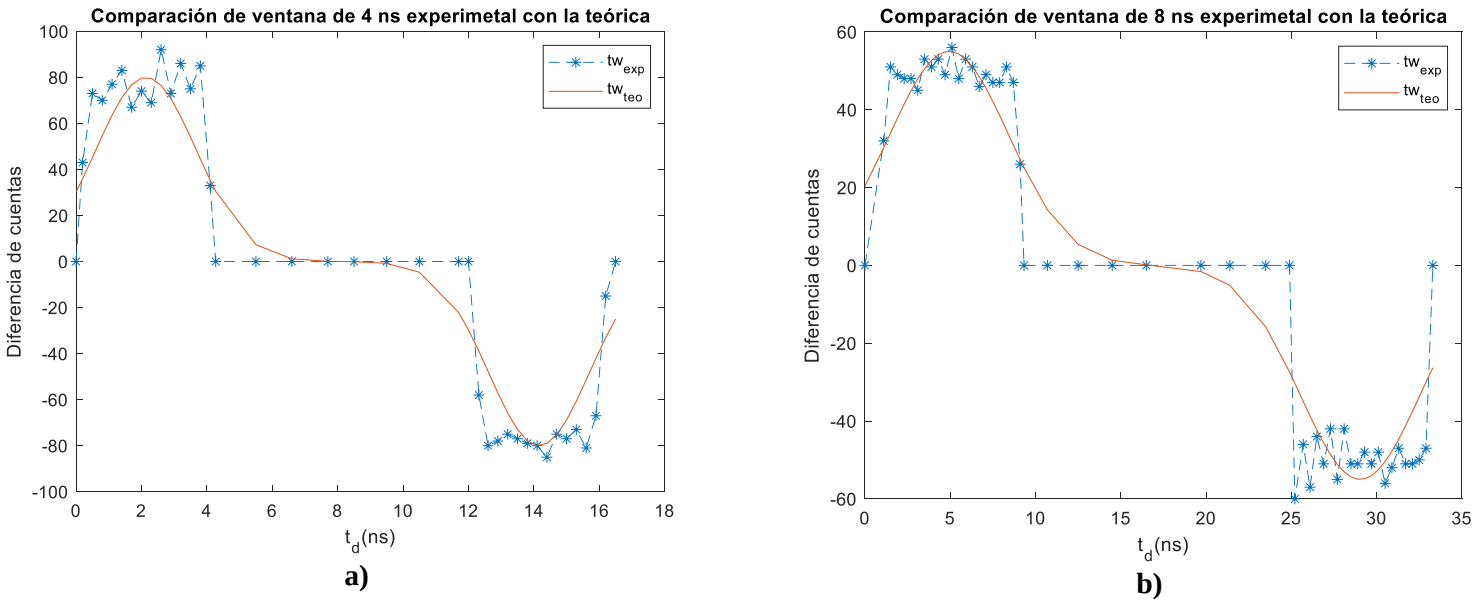


Fig. 4.10 Ajuste de la ventana de coincidencia teórica a la experimental con la ecuación 7 aplicando tres bins activos **(a)** Ventana de coincidencia de 4 ns, **(b)** Ventana de coincidencia de 8 ns.

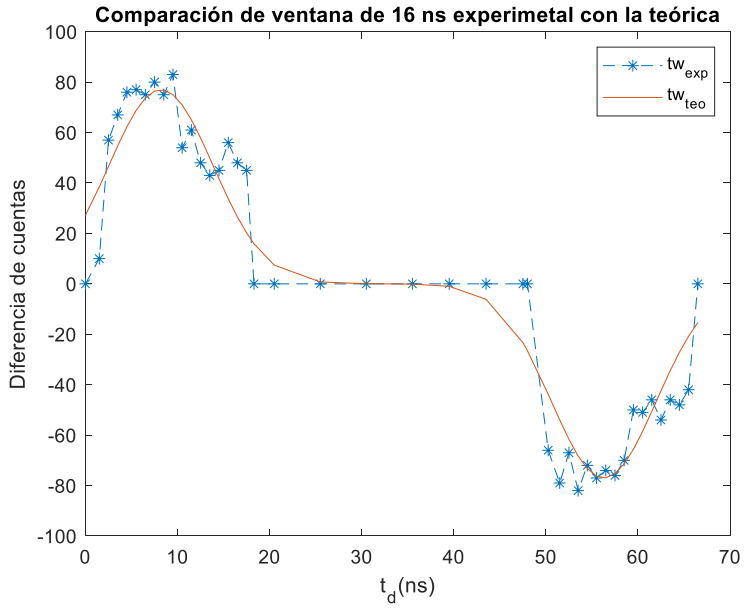


Fig. 4.11 Ajuste de la ventana de coincidencia de 16 ns teórica a la experimental con la ecuación 7 aplicando tres bins activos

Cabe destacar que las gráficas presentadas en las figuras 4.9, 4.10 y 4.11 se generaron aplicando la diferencia a las cuentas coincidentes medidas que se realizó con tres bins activos (como se muestra en la figura 4.5) y utilizando la ecuación 7.

De las gráficas presentadas anteriormente, se aprecia una ventana de coincidencia experimental, que se derivó de las mediciones experimentales aplicando la diferencia, y una ventana teórica, que se obtuvo mediante la ecuación 7. Para lograr que la ventana teórica se ajustara a la ventana experimental, fue necesario ajustar σ, la amplitud y el centrado de la ventana teórica que ayudaron a determinar el valor real de las ventanas de coincidencia. Estos valores se confirmaron al demostrar una alta correlación entre ambas ventanas. Todos los parámetros se pueden ver en la tabla 4.1 para el caso de un solo bin activo y la tabla 4.2 para los tres bins activos.

Tabla 4.1: Parámetros obtenidos de las mediciones experimentales a un bin activo con la ecuación 7.

τ (ns)		σ	R^2
Teórico	**Experimental**		
1	1.0201$\pm$0.0190	0.5310$\pm$0.0104	0.9392
2	2.0267$\pm$0.0153	1.1200$\pm$0.0755	0.9384
4	4.0189$\pm$0.0097	2.1324$\pm$0.0458	0.9389
8	8.0267$\pm$0.0115	4.2433$\pm$0.0643	0.9338
16	16.0267$\pm$0.0058	7.9300$\pm$0.0529	0.9393

Tabla 4.2: Parámetros obtenidos de las mediciones experimentales a tres bins activos con la ecuación 7.

τ (ns)		σ	R^2
Teórico	**Experimental**		
1	1.0044 $\pm$0.0192	0.5893$\pm$0.0181	0.9647
2	2.0233$\pm$0.0196	1.1386$\pm$0.0623	0.9597
4	4.0139$\pm$0.0095	2.2250$\pm$0.0766	0.9607
8	8.0294$\pm$0.0108	4.9950$\pm$0.0761	0.9625
16	16.0244$\pm$0.0050	8.0333$\pm$0.0175	0.9675

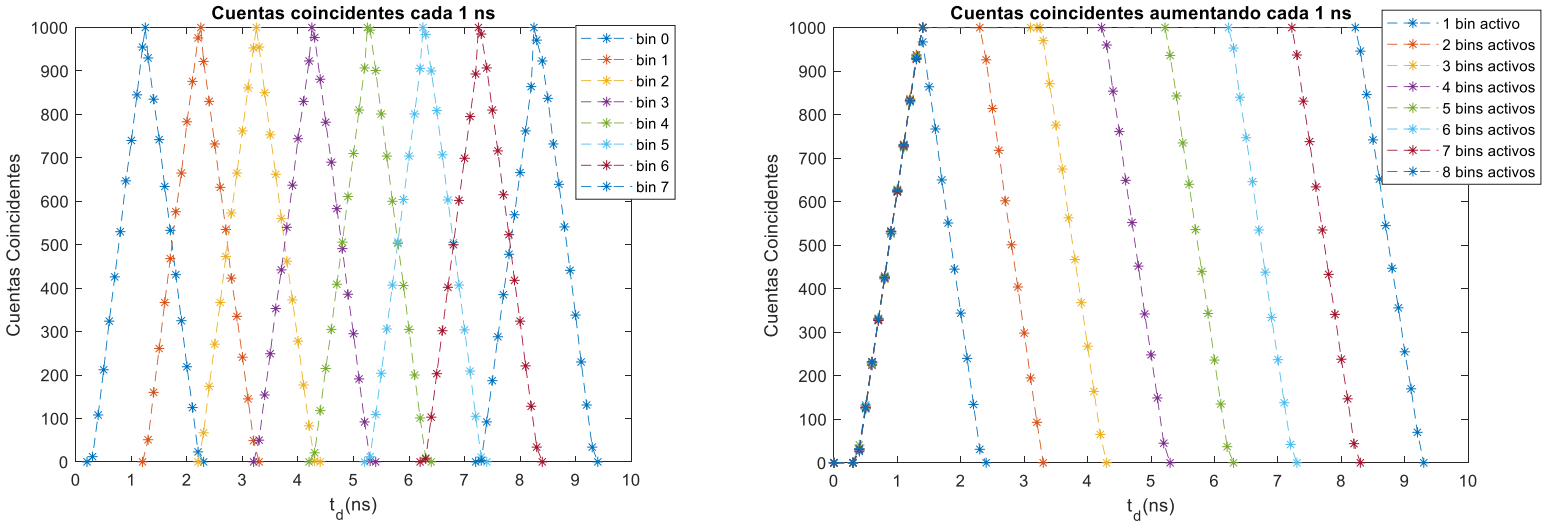
Los resultados de las mediciones realizadas con el generador de funciones se presentan en la tabla 4.1 y tabla 4.2. Se observa que, para todas las ventanas de coincidencia, las mediciones experimentales son muy similares a las teóricas, y cada una de ellas muestra un coeficiente de correlación cuadrática por encima del 90%. Estos resultados indican que la desviación estándar empiricas del ruido electrónico para la ventana de 1 ns es $\sigma = 0.5601 \pm 0.0143$ ns, para la ventana de 2 ns es $\sigma = 1.1293 \pm 0.0132$ ns, para la ventana de 4 ns es $\sigma = 2.1787 \pm 0.0655$ ns, para la ventana de 8 ns es $\sigma = 4.6191 \pm 0.5315$ ns y para la ventana de 16 ns es $\sigma = 7.9817 \pm 0.0730$ ns. Estas desviaciones se originan principalmente en el Jitter del generador de funciones.

Además, con la ecuación 3 se puede obtener la resolución temporal empírica sobre todas las anchuras de ventanas de coincidencia obteniendo $t_r = 1.0068$ ns con un $d_w = 0.0052$ ns.

4.2.1 Evaluación en pares de canales

Se realizaron mediciones para verificar la funcionalidad de diversas combinaciones de canales, los cuales son el canal 0 con el canal 1, el canal 0 con el canal 2, el canal 0 con el canal 3, el canal 1 con el canal 2, el canal 1 con el canal 3 y el canal 2 con el canal 3. Las mediciones se realizaron utilizando una ventana de coincidencia de 1 ns, inicialmente con solo un bin activo. Luego, se activaron progresivamente uno a uno los bins hasta alcanzar un total de 8 bins activos. Los resultados de estas mediciones se muestran en las gráficas siguientes:

Para el canal 0 con canal 1:

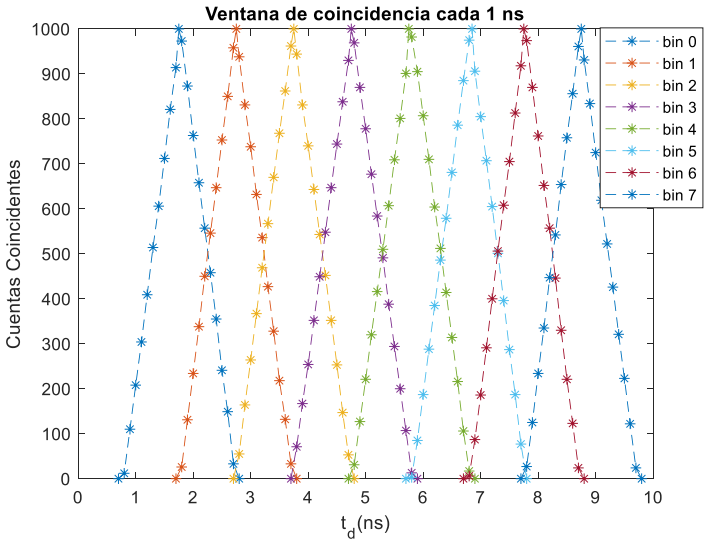


a) **b)**

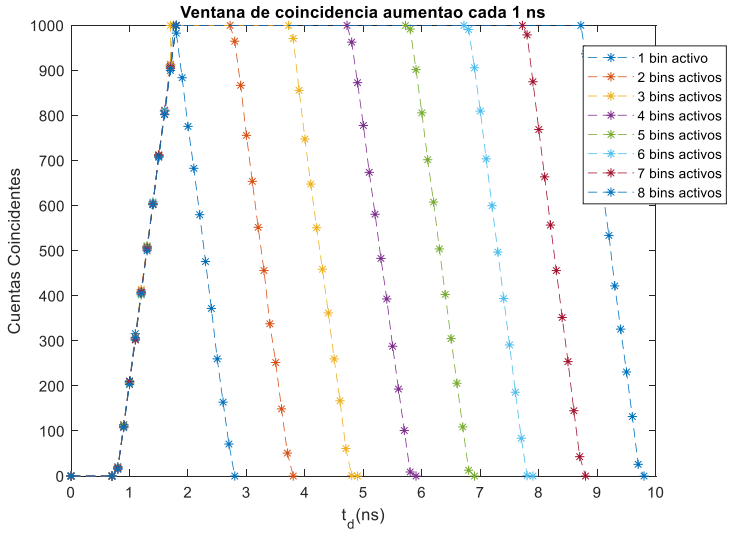
Fig. 4.12 Mediciones experimentales para τ=1 ns utilizando los bins activos

(a) Cuentas coincidentes separado cada un bin activo, **(b)** cuentas coincidentes con aumentos cada un bin activo.

Para el canal 0 con canal 2:



a)

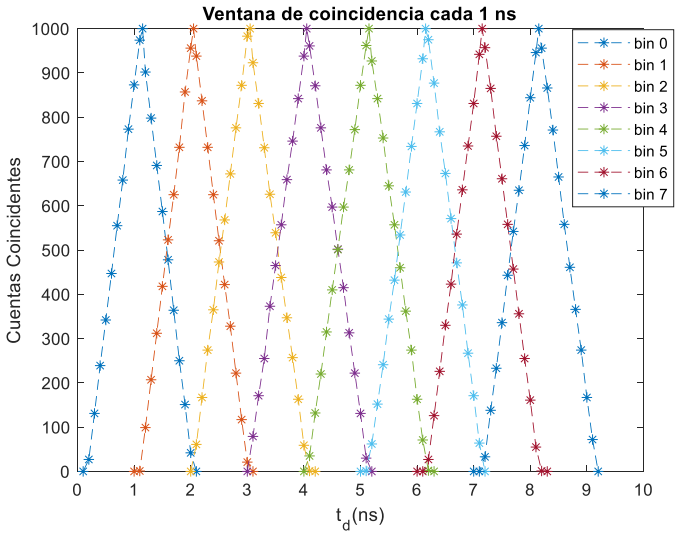


b)

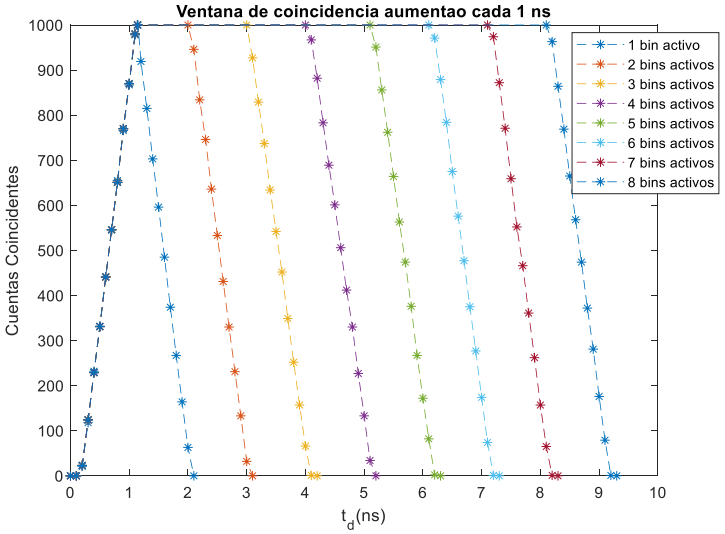
Fig. 4.13 Mediciones experimentales para τ=1 ns utilizando los bins activos

(a) Cuentas coincidentes separado cada un bin activo, **(b)** cuentas coincidentes con aumentos cada un bin activo.

Para el canal 0 con canal 3:



a)

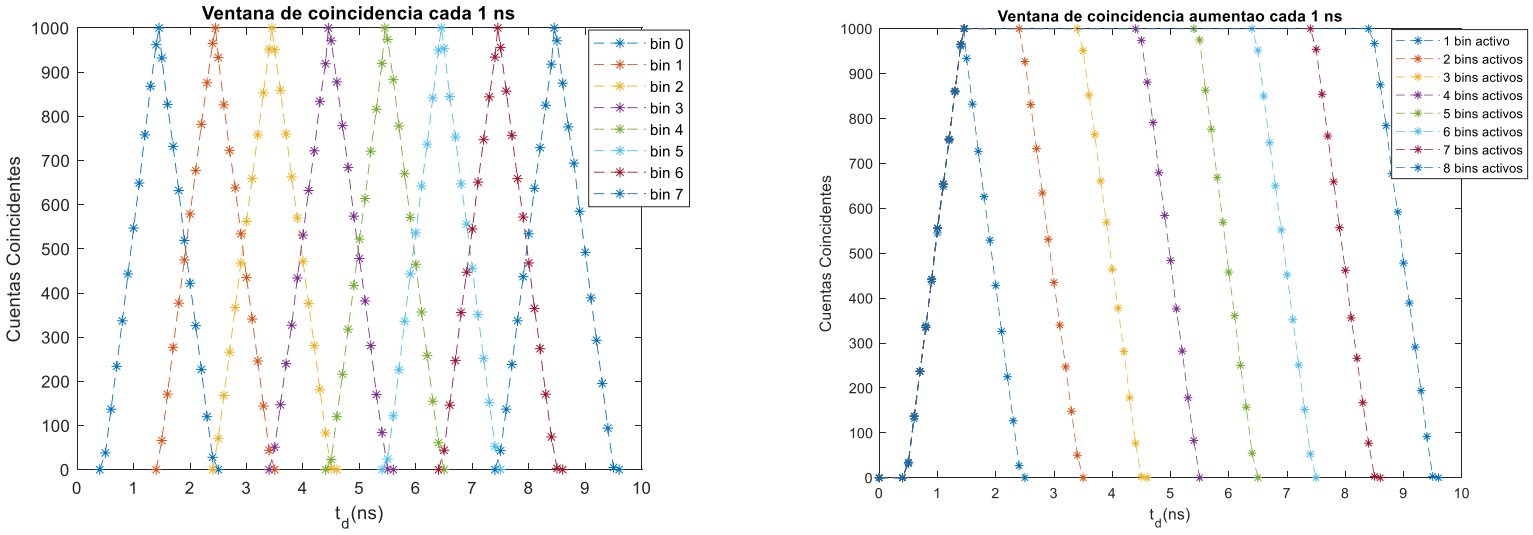


b)

Fig. 4.14 Mediciones experimentales para τ=1 ns utilizando los bins activos

(a) Cuentas coincidentes separado cada un bin activo, **(b)** cuentas coincidentes con aumentos cada un bin activo.

Para el canal 1 con canal 2:

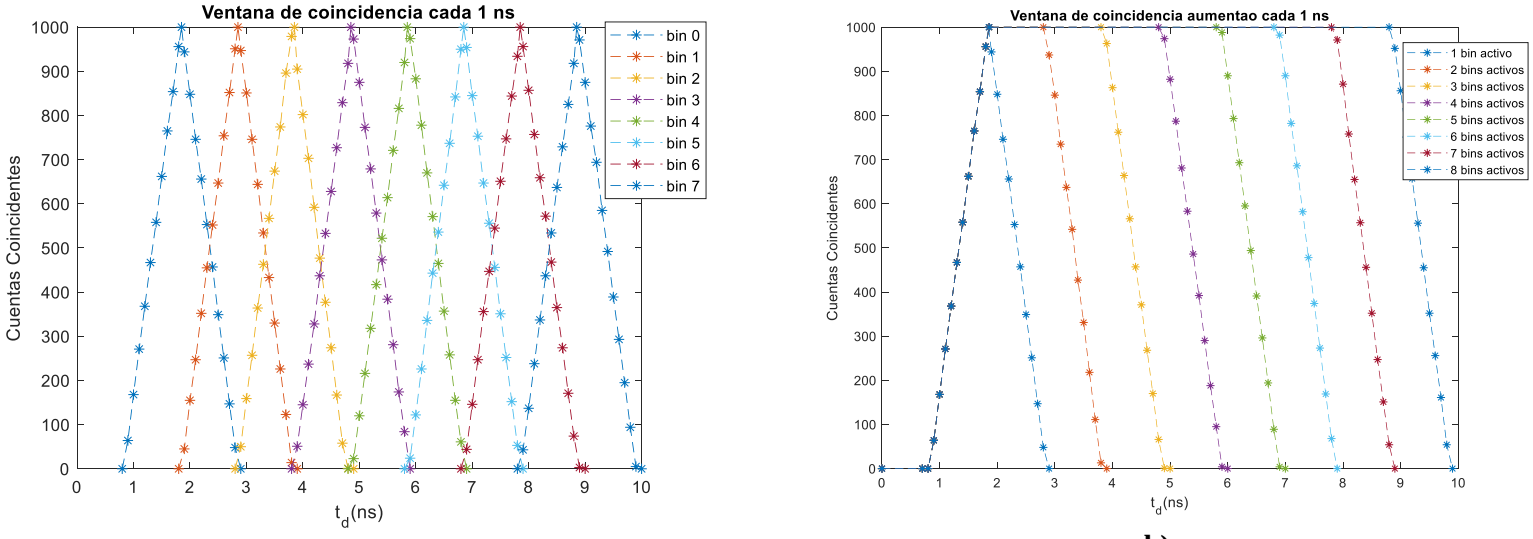


a) **b)**

Fig. 4.15 Mediciones experimentales para τ=1 ns utilizando los bins activos

(a) Cuentas coincidentes separado cada un bin activo, **(b)** cuentas coincidentes con aumentos cada un bin activo.

Para el canal 1 con canal 3:

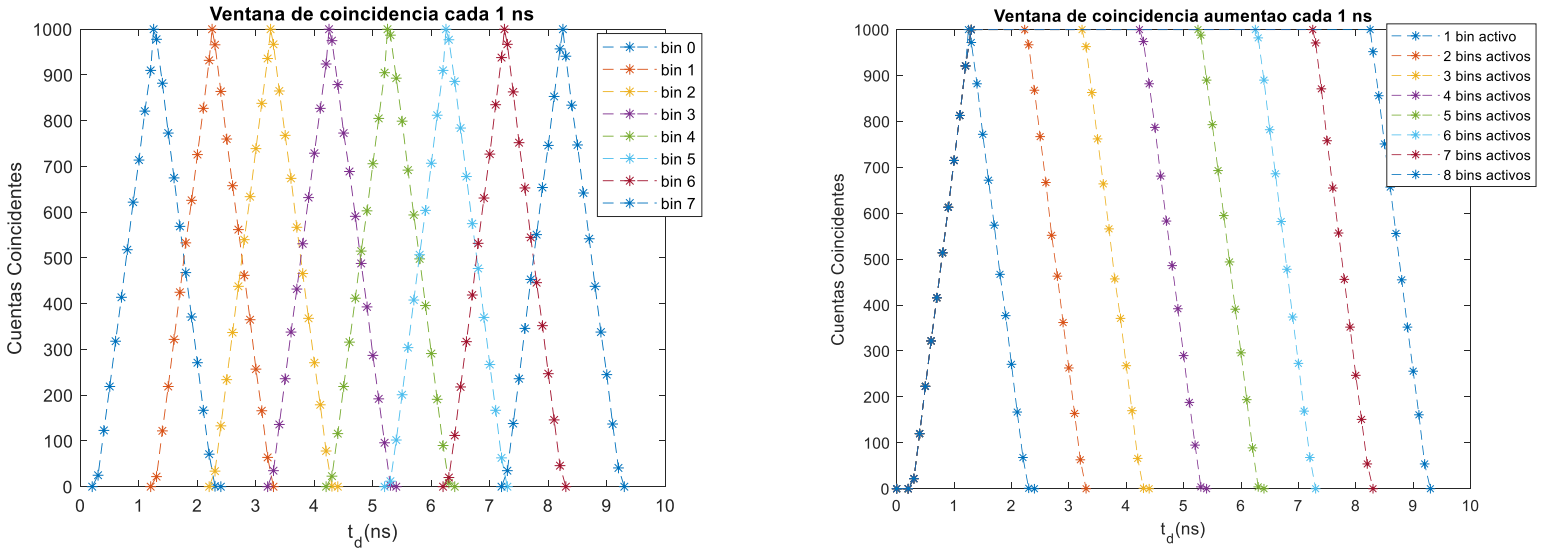


a) **b)**

Fig. 4.16 Mediciones experimentales para τ=1 ns utilizando los bins activos

(a) Cuentas coincidentes separado cada un bin activo, **(b)** cuentas coincidentes con aumentos cada un bin activo.

Para el canal 2 con canal 3:



a) **b)**

Fig. 4.17 Mediciones experimentales para τ=1 ns utilizando los bins activos

(a) Cuentas coincidentes separado cada un bin activo, **(b)** cuentas coincidentes con aumentos cada un bin activo.

En las figuras 4.12.a), 4.13.a), 4.14.a), 4.15.a), 4.16.a) y 4.17.a), se realizaron mediciones de la ventana de coincidencia con un intervalo de 1 ns, activando un bin a la vez. Se observa que esta ventana se desplaza a través del Delay cada 1 ns a medida que se alterna entre la activación y desactivación de los bins subsiguientes. Por lo que, la longitud de estos desplazamientos está directamente relacionada con el tamaño de la ventana de coincidencia utilizada en el proceso.

Mientras en la en las figuras 4.12.b), 4.13.b), 4.14.b), 4.15.b), 4.16.b) y 4.17.b), se puede observar cómo la ventana de coincidencia de 1 ns aumenta en tamaño a medida que se incrementa el número de bins activados. Este enfoque de medición es adecuado, ya que el aumento de los bins se ajusta de acuerdo al tamaño predefinido de la ventana de coincidencia. Es esencial destacar que, para obtener el valor real de la ventana, es necesario dividir su tamaño total entre la cantidad de bines activos.

Cabe mencionar que se aplicó este mismo enfoque de medición a las otras ventanas de coincidencia disponibles y los resultados correspondientes se encuentran detallados en la Tabla 4.3. En esta tabla, se pueden apreciar las mediciones que reflejan con precisión el tamaño de las ventanas de coincidencia.

Tabla 4.3: Parámetros obtenidos de las mediciones experimentales a un bin activo con la ecuación 7.

τ (ns)		σ	R^2
Teórico	**Experimental**		
1	1.0183 ± 0.0075	0.5083 ± 0.0354	0.9390
2	2.0333 ± 0.0121	1.0733 ± 0.0641	0.9393
4	4.0401 ± 0.0089	2.1683 ± 0.0248	0.9366
8	8.0617 ± 0.0194	4.3250 ± 0.0632	0.9311
16	16.0450 ± 0.0164	7.9480 ± 0.0497	0.9413

De acuerdo a la tabla 4.3 se puede obtener gracias a la ecuación 3 la resolución temporal empírica $t_r = 1.0111$ ns con $d_w = 0.0064$ ns.

Finalmente, a partir de todas las mediciones efectuadas mediante diversos métodos y para todas las posibles combinaciones entre pares de canales posibles, se puede concluir que cada una de las ventanas de coincidencia muestra una consistente concordancia en lo que respecta a sus tamaños. Estas mediciones experimentales coinciden de manera sólida con las expectativas teóricas, lo que respalda la precisión y la confiabilidad del sistema contador de coincidencias.

Capítulo 5. Conclusiones

En este proyecto se ha logrado diseñar un sistema basado en el FPGA Zedboard Zynq-7000 obteniendo una precisión excepcional en la detección de coincidencias entre dos pulsos eléctricos operando a una velocidad de muestreo de 1 GS/s. El sistema ha demostrado de manera consistente ser eficiente y confiable.

Lo que hace que el sistema sea verdaderamente especial es su flexibilidad, que permite la adaptación del ancho de la ventana de coincidencia incluso después de realizar la medición y ofrece la capacidad de combinar de a pares los cuatro canales disponibles. Esto es posible gracias al FPGA Zynq-7000.

La interfaz se ha desarrollado utilizando el software Visual C# permite visualizar las cuentas coincidentes en tiempo real y brinda la versatilidad de ajustar las ventanas de coincidencia y aplicar correcciones o retrasos de forma manual, independientemente para cada canal, para satisfacer las preferencias del usuario.

Los resultados obtenidos en cuanto al tamaño de las ventanas de coincidencia son altamente alentadores, ya que en todos los casos se asemejaron estrechamente a las ventanas teóricas, con una correlación que supera el 90%. Además, independientemente de la combinación de canales utilizada, se mantuvo una consistencia en el tamaño de las ventanas de coincidencia, lo que demuestra que el sistema está calibrado y listo para ser empleado en aplicaciones relacionadas con sistemas ópticos cuánticos.

5.1. Trabajos futuros

Como trabajo futuro, se propone llevar a cabo la caracterización del sistema para detecciones en coincidencia triples y cuádruples, lo que permitirá una comprensión más profunda de su comportamiento en situaciones de mayor complejidad. Esto abrirá el camino para aplicaciones más avanzadas en el campo de la óptica cuántica y la exploración de fenómenos cuánticos con un mayor número de fotones involucrados. Adicionalmente, se sugiere emplear el sistema en experimentos de óptica cuántica con el fin de evaluar su eficiencia y desempeño en condiciones reales de investigación.

Referencias

[1] E. Arabul, S. Paesani, S. Tancock, J. Rarity y N. Dahnoun, «A Precise High Count-Rate FPGA based Multi-Channel Coincidence Counting System for Quantum Photonics Applications,» *IEEE Photonics Journal,* vol. 12, nº 2, pp. 1-14, Abril 2020.

[2] A. Aspuru-Guzik y P. Walther, «Photonic quantum simulators,» *Nature Physics,* vol. 8, nº 4, pp. 285-291, 01 04 2012.

[3] S. Pirandola, U. L. Andersen, L. Banchi, M. Berta, D. Bunandar, R. Colbeck, D. Englund, T. Gehring, C. Lupo, C. Ottaviani, J. Pereira, M. Razavi, J. S. Shaari, M. Tomamichel, V. C. Usenko, C. Vallone, P. Villoresi y P. Wallden, «Advances in quantum cryptography,» *Adv. Opt. Photon,* vol. 12, nº 4, pp. 1012-1236, 12 2020.

[4] D. Deutsch y R. Jozsa, «Rapid Solution of Problems by Quantum Computation,» *Proc. R. Soc. Lond,* p. 553–558, 1992.

[5] S. P. Neumann, T. Scheidl, M. Selimovic, M. Pivoluska, B. Liu, M. Bohmann y R. Ursin, «Model for optimizing quantum key distribution with continuous-wave pumped entangled-photon sources,» *Phys. Rev. A,* vol. 104, nº 2, p. 022406, 05 08 2021.

[6] C. H. Bennett y S. J. Wiesner, «Communication via One- and Two-Particle Operators on Einstein-Podolsky-Rosen States,» *Physical Review Letters,* vol. 69, nº 20, pp. 2881-2884, 16 11 1992.

[7] J. Cariñe, S. A. Gómez, G. F. Obregón, E. S. Gómez, M. Figueroa, G. Lima y G. B. Xavier, «Post-Measurement Adjustment of the Coincidence Window in Quantum Optics Experiments,» *IEEE Access,* vol. 9, pp. 94010-94016, 2021.

[8] S. Gaertner, H. Weinfurter y C. Kurtsiefer, «Fast and compact multichannel photon coincidence unit for quantum information processing,» *Review of Scientific Instruments,* vol. 76, nº 12, p. 123108, 12 2005.

[9] R. C. Pooser, D. D. Earl, P. G. Evans, B. Williams, J. Schaake y T. S. Humble, «FPGA-based gating and logic for multichannel single photon counting,» *Journal of Modern Optics,* vol. 59, nº 17, pp. 1500-1511, 10 10 2012.

[10] G. Sportelli, N. Belcar, P. Guerra y A. Santos, «Low-resource synchronous coincidence processor for positrón emission tomography,» *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment,* vol. 648, pp. S199-S201, 21 08 2011.

[11] B. K. Park, Y.-S. Kim, O. Know, S.-W. Han y S. Moon, «High-performance reconfigurable coincidence counting unit based on a field programmable gate array,» *Applied Optics,* vol. 54, nº 15, pp. 4727-4731, 20 05 2015.

[12] Avnet, «ZedBoard Zynq Evaluation and Development Hardware user's Guide,» 2014.

[13] R. H. hadfield, «single-photon detectors for optical quantum information applications,» *Nature Photonics,* vol. 3, nº 12, pp. 696-705, 01 12 2009.

[14] W. Becker, Advanced Time-Correlated Single Photon Counting Techniques, vol. 81, Springer, 2005.

[15] M. Eisaman D., J. Fan, A. Migdall y S. Polyakov V., «Single-photon sources and detectors,» *The Review of scientific instruments,* p. 071101, 01 07 2011.

[16] IDQ, «ID210 ADVANCED SYSTEM FOR SINGLE PHOTON DETECTION WITH 100MHz GATED MODE AND FREE-RUNNING MODE,» 2014.

[17] Electronic, IAM, «Breakout board for Low-Pin Count FPGA Mezzanine Cards,» 2018.

[18] F. J. R. Serrano, "Diseño e implementacion de la correlacion y de la correntropia cruzada, utilizando FPGA," Santiago, Chile, 2017.

[19] S. Churiwala, Designing with Xilinx® FPGAs Using Vivado, Suiza, 2017.

[20] Digilent, «Digilent,» [En línea]. Available: https://digilent.com/reference/programmable-logic/zedboard/reference-manual..

[21] Xilinx, «Zynq-7000 SoC Technical Reference Manual,» 2021.

[22] G. T. B. Y. S. S. Ali Antucu M., «Real-Time System Implementation for Image Processing with Hardware/Software Co-design on the Xilinx Xynq Plataform,» *International Journal of Information and Electronics Engineering*, vol. 5, pp. 473-477, 06 10 2015.

[23] Xilinx, AMD, «Xilinx,» [En línea]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.

[24] R. S. Hasan, S. K. Tawfeeq, N. Q. Mohammed y A. I. Khaleel, «A true random number generator based on the photon arrival time registered in a coincidence window between two single-photon counting modules,» *Chinese Journal of Physics*, vol. 56, nº 1, pp. 385-391, 2018.

[25] M. Conti, «Method for reducing an electronic time coincidence window in positron emission tomography,» Knoxville, TN (US), 2005.

[26] Xilinx, «RAM-Based Shift Register v12.0 LogiCORE IP Product Guide,» 2021.

[27] AMD, «AMD,» 22 10 2021. [En línea]. Available: https://docs.xilinx.com/r/2021.2-English/ug953-vivado-7series-libraries/ISERDESE2.

[28] T. R. Kuphaldt, Lessons In Electric Circuits, Volume IV – Digital, vol. 4, 2007.

[29] Xilinx, «AMD,» 20 04 2022. [En línea]. Available: https://docs.xilinx.com/r/en-US/pg065-clk-wiz/Clocking-Wizard-v6.0-LogiCORE-IP-Product-Guide.

[30] Microsoft, «Microsoft,» 2022. [En línea]. Available: https://learn.microsoft.com/es-es/visualstudio/get-started/csharp/?view=vs-2022.

[31] J. Foxall, Visual C# 2008 in 24 hours, USA: Sams, 2008.

[32] «everythingRF,» 03 09 2018. [En línea]. Available: https://www.everythingrf.com/community/sma-connectors.

Anexo A. Códigos en Matlab

Código de Matlab para la señal del SPD (ID210):

Cabe menciar que se utilizaron archivos Excel para guardar los datos.

```
clear
clc
Data = csvread('tek0001ALL.csv',22);

Data_x = Data(:, 1); % Primera columna
Data_y = Data(:, 2); % Segunda columna


Data_Long=size(Data, 1);

Data_x_v = reshape(Data_x', 1, Data_Long);
Data_y_v = reshape(Data_y, 1, Data_Long);

plot(Data_x_v,Data_y_v,'red','LineWidth',2)
xlim([-20*10^-9 30*10^-9])
legend('Señal del SPD')
xlabel('t (s)')
ylabel('V')
title('Señal arrojada por SPD')
```

Códigos para observar las ventanas de coincidencia:

Cabe mencionar que se utilizaron archivos Excel para guardar los datos.

```
clear all
clc
close all

t0=xlsread('tw0.xlsx','A2:A25');
cc0=xlsread('tw0.xlsx','B2:B25');

t1=xlsread('tw1.xlsx','A2:A43');
cc1=xlsread('tw1.xlsx','B2:B43');

t2=xlsread('tw2.xlsx','S2:S32');
cc2=xlsread('tw2.xlsx','T2:T32');

t3=xlsread('tw3.xlsx','S2:S45');
cc3=xlsread('tw3.xlsx','T2:T45');

t4=xlsread('tw4.xlsx','S2:S37');
cc4=xlsread('tw4.xlsx','T2:T37');

figure(1)
subplot(2,1,1)
plot(t0,cc0,'--*')
title('Ventana de coincidencia de 1 ns')
legend('tw= 1ns')
xlabel('t_d(ns)')
ylabel('Cuentas Coincidentes')
xlim([0 35])
subplot(2,1,2)
plot(t1,cc1,'--*')
title('Ventana de coincidencia de 2 ns')
legend('tw= 2ns')
xlabel('t_d(ns)')
ylabel('Cuentas Coincidentes')
xlim([0 35])

figure(2)
subplot(2,1,1)
plot(t2,cc2,'--*')
title('Ventana de coincidencia de 4 ns')
legend('tw= 4ns')
xlabel('t_d(ns)')
```

```
ylabel('Cuentas Coincidentes')
xlim([0 35])
subplot(2,1,2)
plot(t3,cc3,'--*')
title('Ventana de coincidencia de 8 ns')
legend('tw= 8ns')
xlabel('t_d(ns)')
ylabel('Cuentas Coincidentes')
xlim([0 35])

figure(3)
plot(t4,cc4,'--*')
title('Ventana de coincidencia de 16 ns')
legend('tw= 16ns')
xlabel('t_d(ns)')
ylabel('Cuentas Coincidentes')
xlim([0 35])
```

Códigos para ajustar la ventana de coincidencia con ecuación 7:

```
clear all
clc
close all

load tw4-cc23
f=tw0;

g=diff(f);
plot(g)
td=tw0(:,1);
f=tw0(:,2);
g=[diff(f);0];
sigma=8.24;
A=77;
t0=6.6;
k=1;

for tw=14:0.01:17
g2=A*(exp(-(td-t0).^2/(2*sigma.^2))-exp(-(td-t0-
tw).^2/(2*sigma.^2)));
subplot(2,1,1)
plot(td,g,'--*',td,g2)
title('Comparación de ventana de 16 ns experimetal con la
teórica')
legend('tw_e_x_p','tw_t_e_o')
```

```
xlabel('t_d(ns)')
ylabel('Diferencia de cuentas')
R2(k,1)=corr(g,g2);
R2(k,2)=tw;
k=k+1;
pause(0.001);
end

subplot(2,1,2)
plot(R2(:,1))
```

Códigos para observar ventana de 1 ns utilizando los bins activos:

```
clear all
clc
close all

t0=xlsread('tw0-1bin.xlsx','A2:A25');
cc0=xlsread('tw0-1bin.xlsx','B2:B25');

t1=xlsread('tw0-1bin.xlsx','D2:D24');
cc1=xlsread('tw0-1bin.xlsx','E2:E24');

t2=xlsread('tw0-1bin.xlsx','G2:G25');
cc2=xlsread('tw0-1bin.xlsx','H2:H25');

t3=xlsread('tw0-1bin.xlsx','J2:J25');
cc3=xlsread('tw0-1bin.xlsx','K2:K25');

t4=xlsread('tw0-1bin.xlsx','M2:M25');
cc4=xlsread('tw0-1bin.xlsx','N2:N25');

t5=xlsread('tw0-1bin.xlsx','P2:P24');
cc5=xlsread('tw0-1bin.xlsx','Q2:Q24');

t6=xlsread('tw0-1bin.xlsx','S2:S24');
cc6=xlsread('tw0-1bin.xlsx','T2:T24');

t7=xlsread('tw0-1bin.xlsx','V2:V24');
cc7=xlsread('tw0-1bin.xlsx','W2:W24');


figure(1)
plot(t0,cc0,'--*')
hold on
plot(t1,cc1,'--*')
hold on
```

```
plot(t2,cc2,'--*')
hold on
plot(t3,cc3,'--*')
hold on
plot(t4,cc4,'--*')
hold on
plot(t5,cc5,'--*')
hold on
plot(t6,cc6,'--*')
hold on
plot(t7,cc7,'--*')
hold off
title('Ventana de coincidencia cada 1 ns')
legend('bin 0','bin 1','bin 2','bin 3','bin 4','bin 5','bin
6','bin 7')
xlabel('t_d(ns)')
ylabel('Cuentas Coincidentes')


clear all
clc
close all

t0=xlsread('tw-aumentobins.xlsx','A2:A26');
cc0=xlsread('tw-aumentobins.xlsx','B2:B26');

t1=xlsread('tw-aumentobins.xlsx','D2:D26');
cc1=xlsread('tw-aumentobins.xlsx','E2:E26');

t2=xlsread('tw-aumentobins.xlsx','G2:G27');
cc2=xlsread('tw-aumentobins.xlsx','H2:H27');

t3=xlsread('tw-aumentobins.xlsx','J2:J27');
cc3=xlsread('tw-aumentobins.xlsx','K2:K27');

t4=xlsread('tw-aumentobins.xlsx','M2:M27');
cc4=xlsread('tw-aumentobins.xlsx','N2:N27');

t5=xlsread('tw-aumentobins.xlsx','P2:P26');
cc5=xlsread('tw-aumentobins.xlsx','Q2:Q26');

t6=xlsread('tw-aumentobins.xlsx','S2:S26');
cc6=xlsread('tw-aumentobins.xlsx','T2:T26');

t7=xlsread('tw-aumentobins.xlsx','V2:V26');
cc7=xlsread('tw-aumentobins.xlsx','W2:W27');
```

```
figure(1)
plot(t0,cc0,'--*')
hold on
plot(t1,cc1,'--*')
hold on
plot(t2,cc2,'--*')
hold on
plot(t3,cc3,'--*')
hold on
plot(t4,cc4,'--*')
hold on
plot(t5,cc5,'--*')
hold on
plot(t6,cc6,'--*')
hold on
plot(t7,cc7,'--*')
hold off
title('Ventana de coincidencia aumentao cada 1 ns')
legend('1 bin activo','2 bins activos','3 bins activos','4
bins activos','5 bins activos','6 bins activos','7 bins
activos','8 bins activos')
xlabel('t_d(ns)')
ylabel('Cuentas Coincidentes')
```

Anexo B. Anchos de las diferentes ventanas de coincidencia

Se presentan las ventanas de coincidencia con los distintos anchos para cada combinación en par de canales

Para el canal 0 con el canal 1:

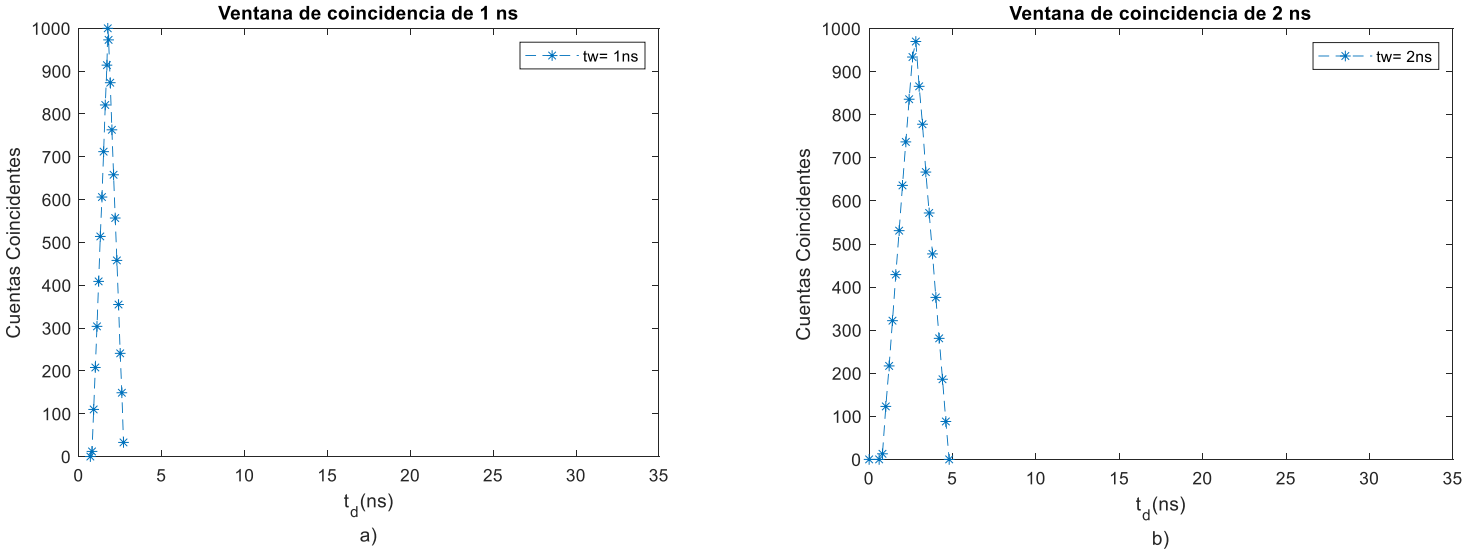


Fig. B.1 Ventanas de coincidencia con un bin activo
(a) Ventana de coincidencia de 1 ns, **(b)** Ventana de coincidencia de 2 ns.

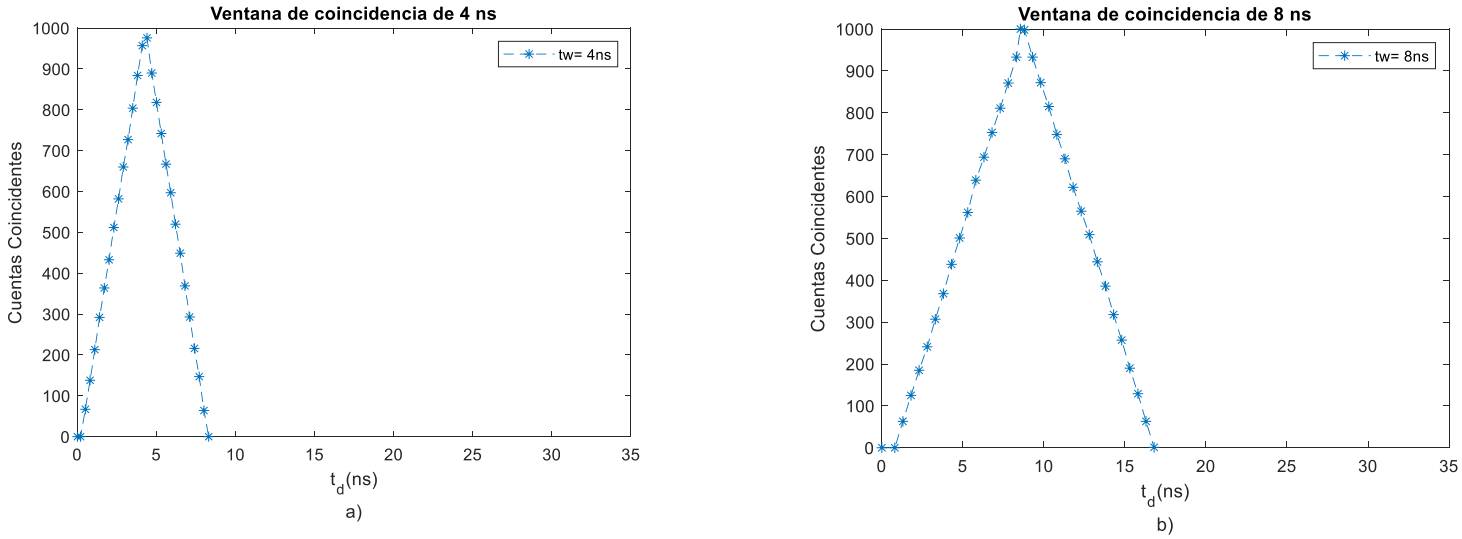


Fig. B.2 Ventanas de coincidencia con un bin activo
(a) Ventana de coincidencia de 4 ns, **(b)** Ventana de coincidencia de 8 ns.

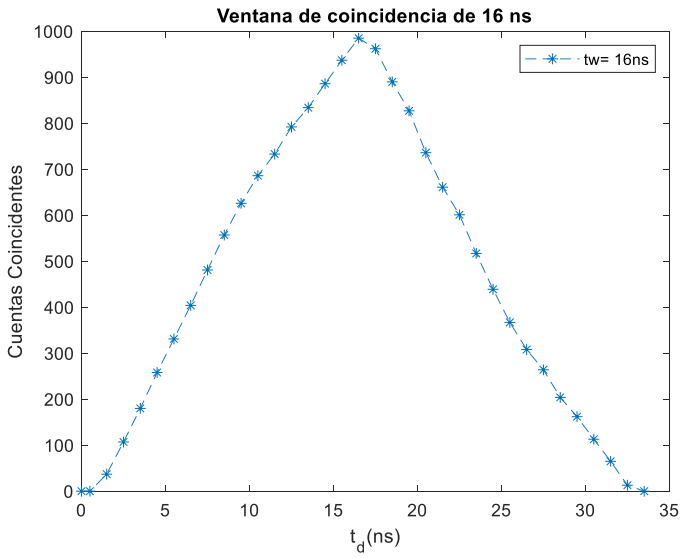


Fig. B.3 Ventana de coincidencia de 16 ns a un bin activo

Para el canal 0 con el canal 2:

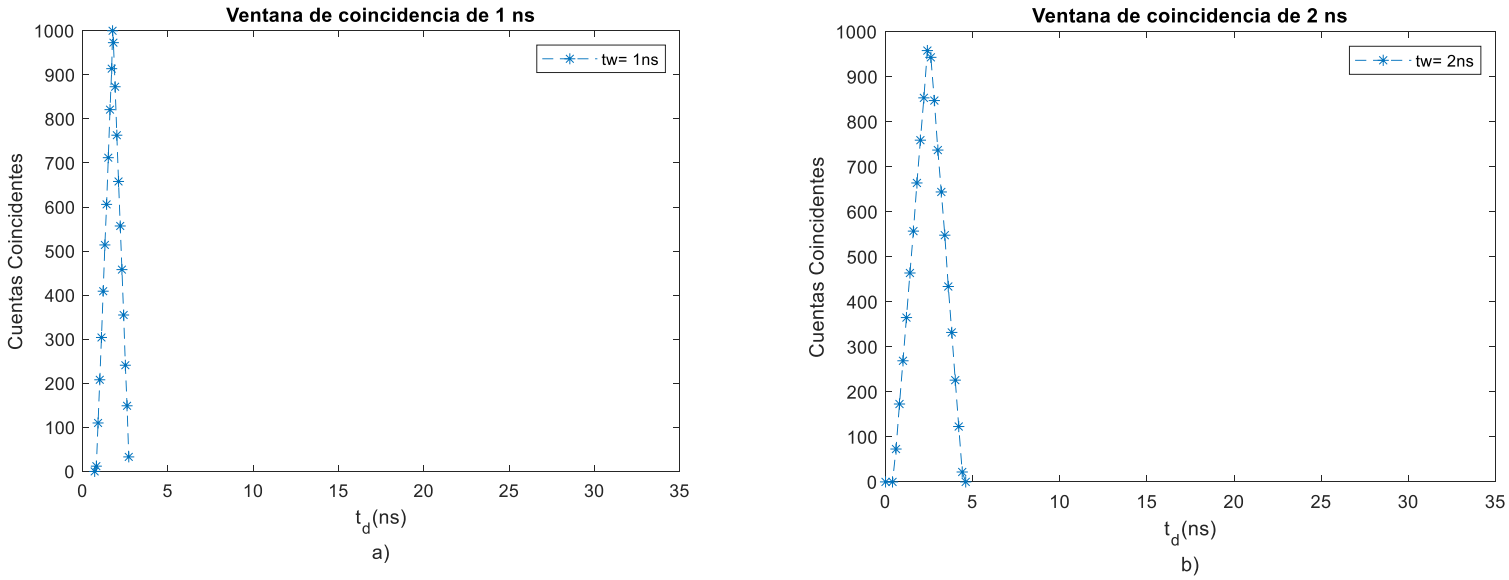


Fig. B.4 Ventanas de coincidencia con un bin activo
(a) Ventana de coincidencia de 1 ns, **(b)** Ventana de coincidencia de 2 ns.

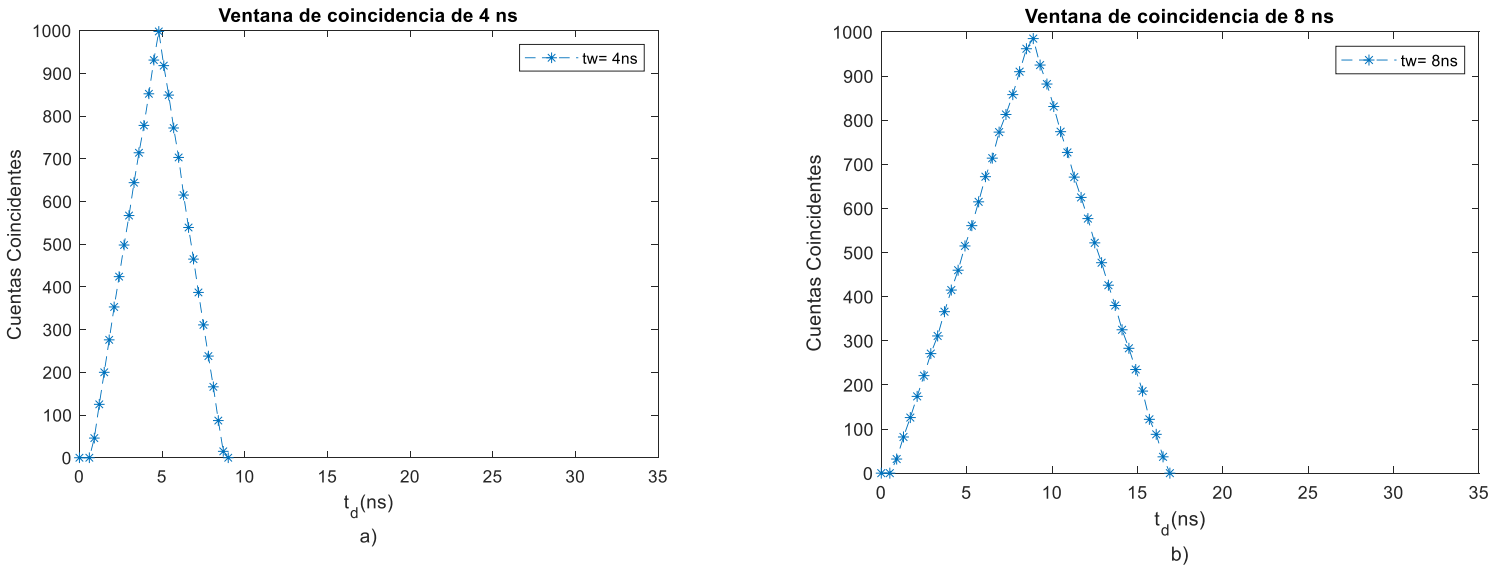


Fig. B.5 Ventanas de coincidencia con un bin activo
(a) Ventana de coincidencia de 4 ns, **(b)** Ventana de coincidencia de 8 ns.

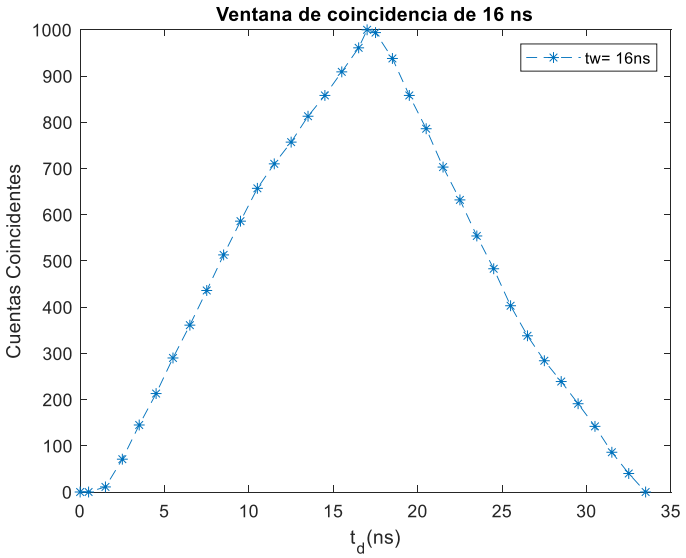


Fig. B.6 Ventana de coincidencia de 16 ns a un bin activo

Para el canal 0 con el canal 3:

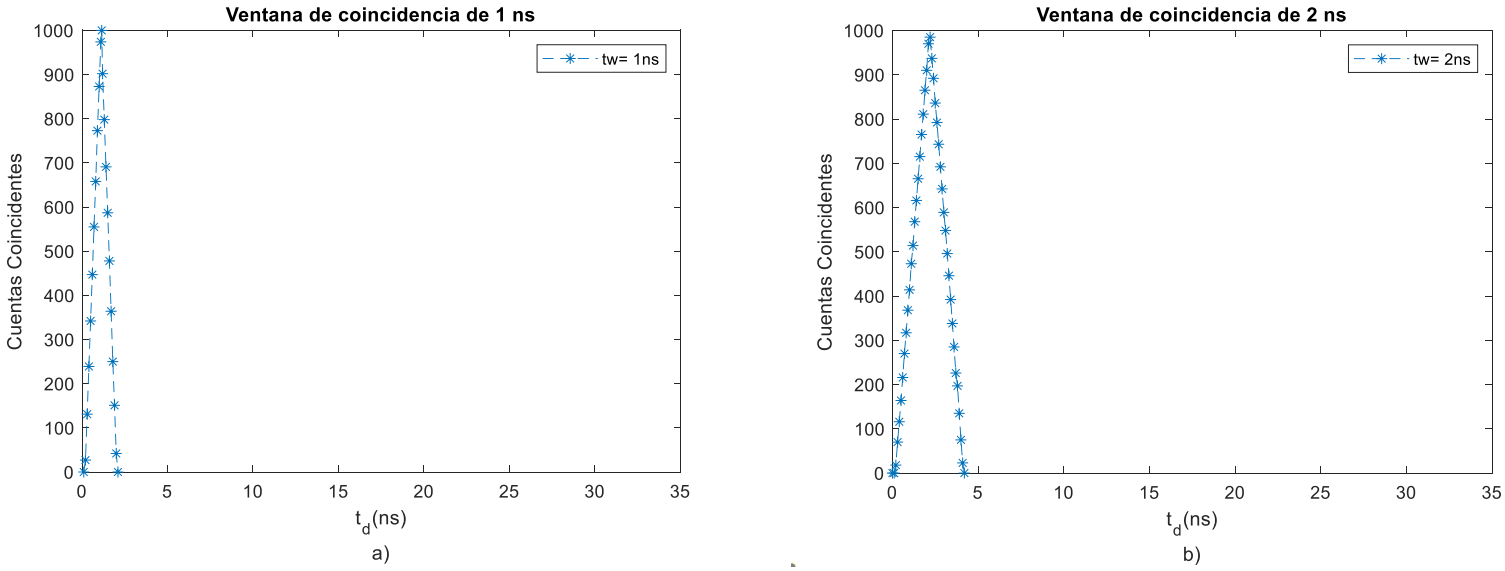


Fig. B.7 Ventanas de coincidencia con un bin activo
(a) Ventana de coincidencia de 1 ns, **(b)** Ventana de coincidencia de 2 ns.

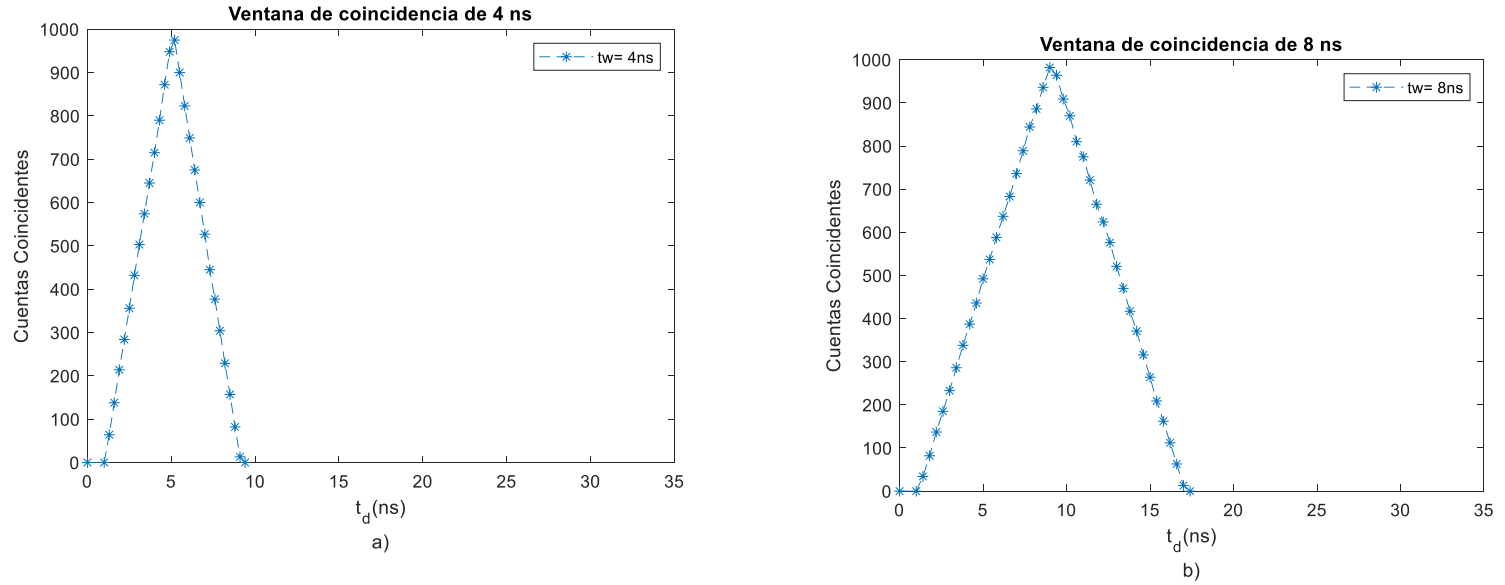


Fig. B.8 Ventanas de coincidencia con un bin activo
(a) Ventana de coincidencia de 4 ns, **(b)** Ventana de coincidencia de 8 ns.

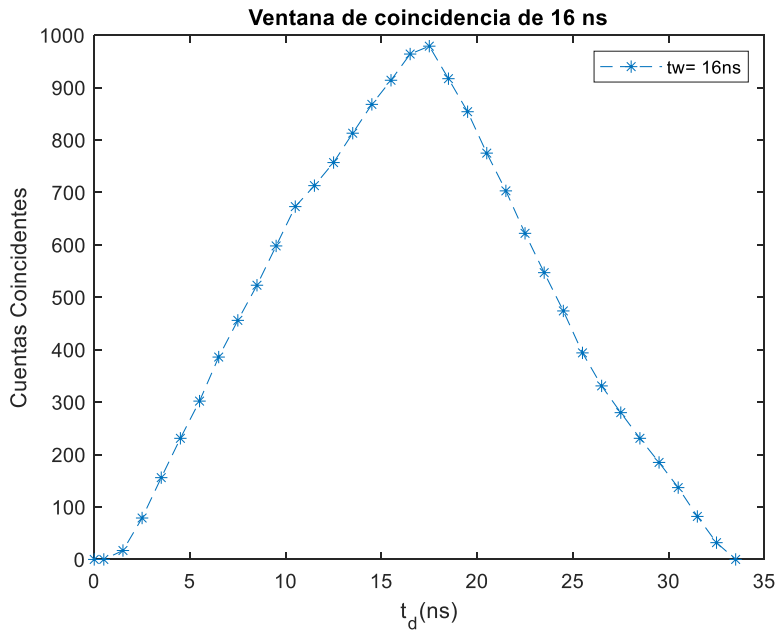


Fig. B.9 Ventana de coincidencia de 16 ns a un bin activo

Para el canal 1 con el canal 2:

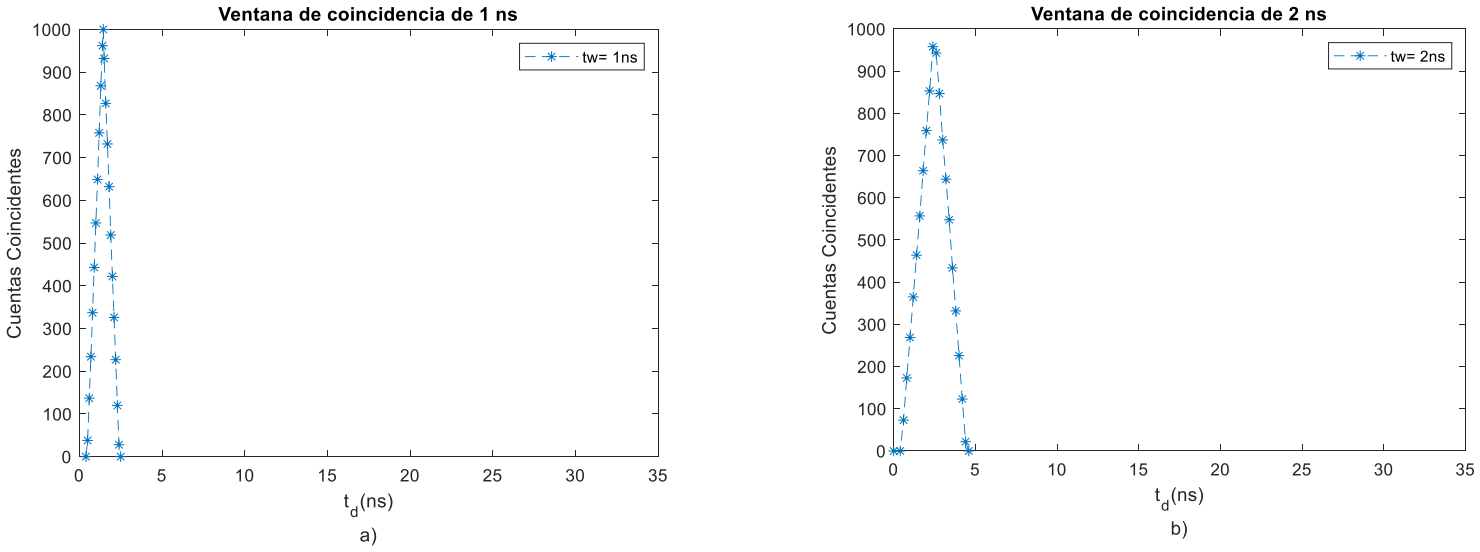


Fig. B.10 Ventanas de coincidencia con un bin activo
(a) Ventana de coincidencia de 1 ns, **(b)** Ventana de coincidencia de 2 ns.

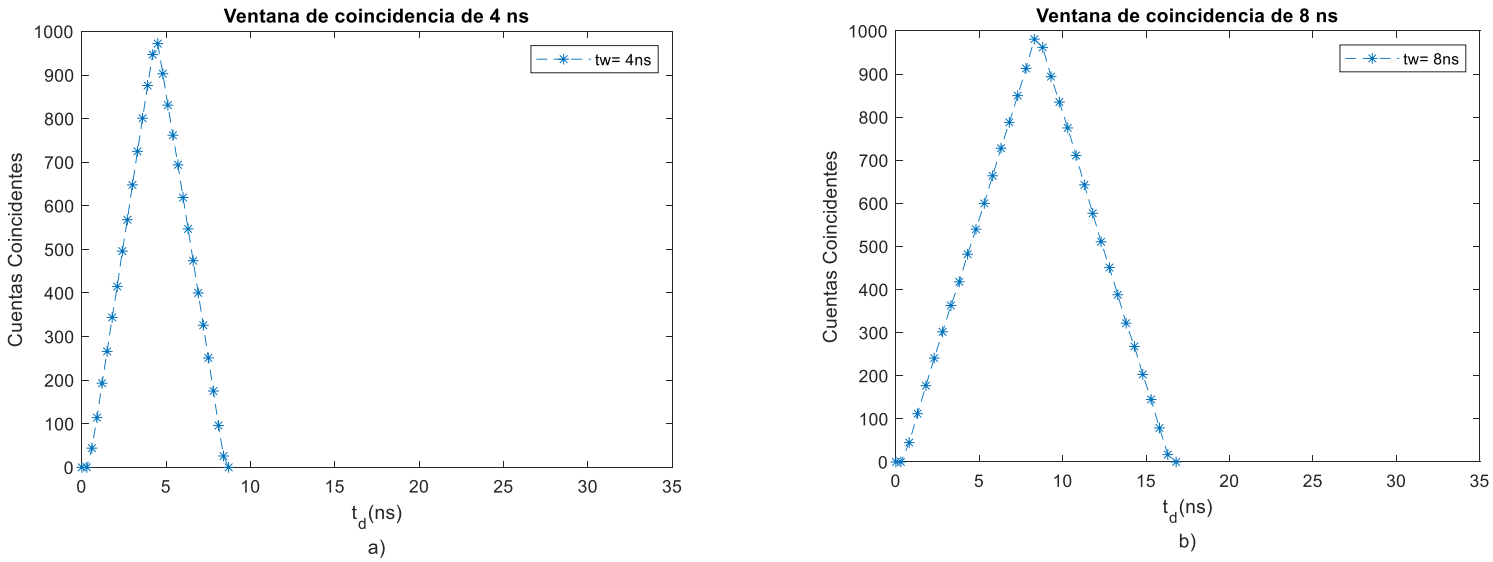


Fig. B.11 Ventanas de coincidencia con un bin activo
(a) Ventana de coincidencia de 4 ns, **(b)** Ventana de coincidencia de 8 ns.

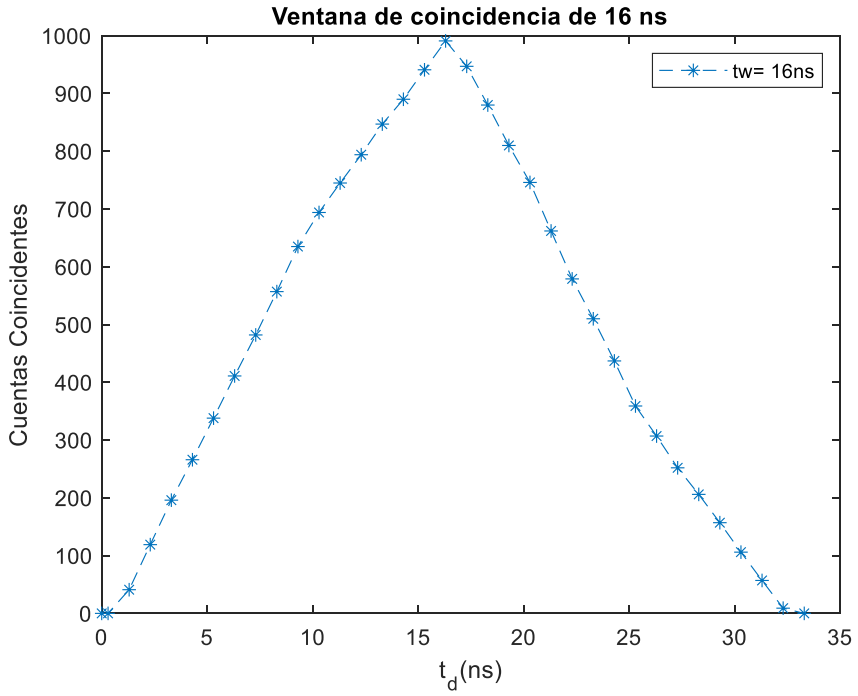


Fig. B.12 Ventana de coincidencia de 16 ns a un bin activo

Para el canal 1 con el canal 3:

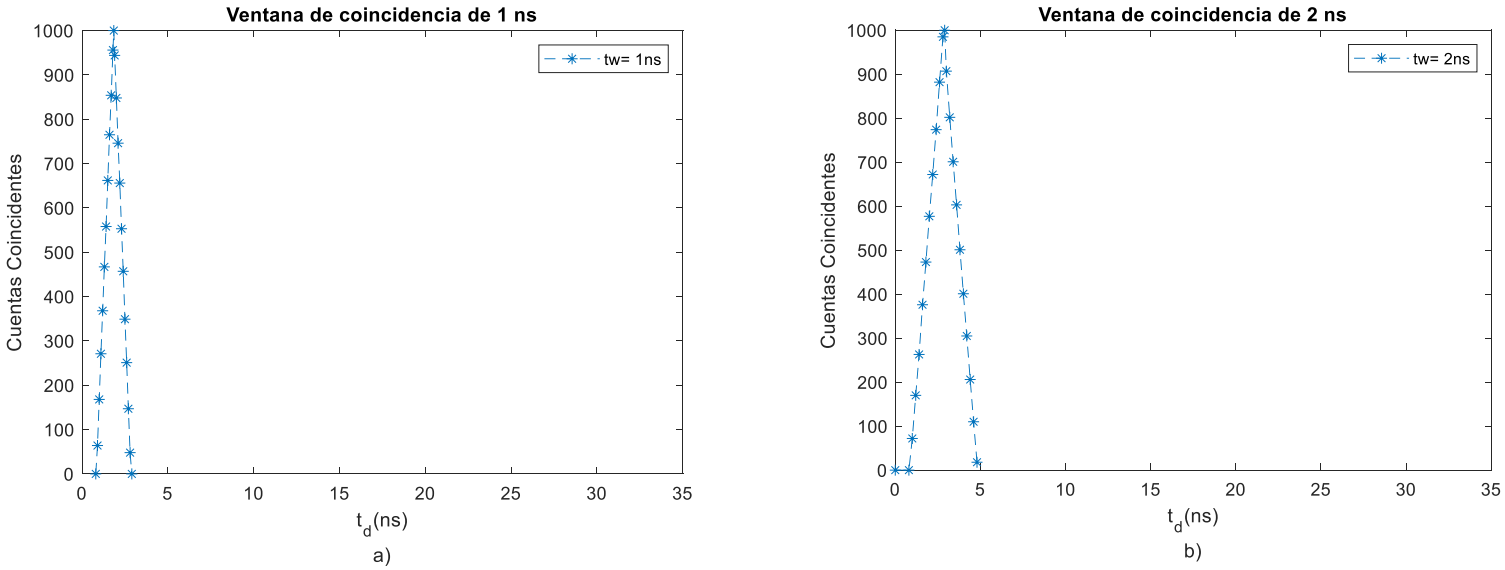


Fig. B.13 Ventanas de coincidencia con un bin activo
(a) Ventana de coincidencia de 1 ns, **(b)** Ventana de coincidencia de 2 ns.

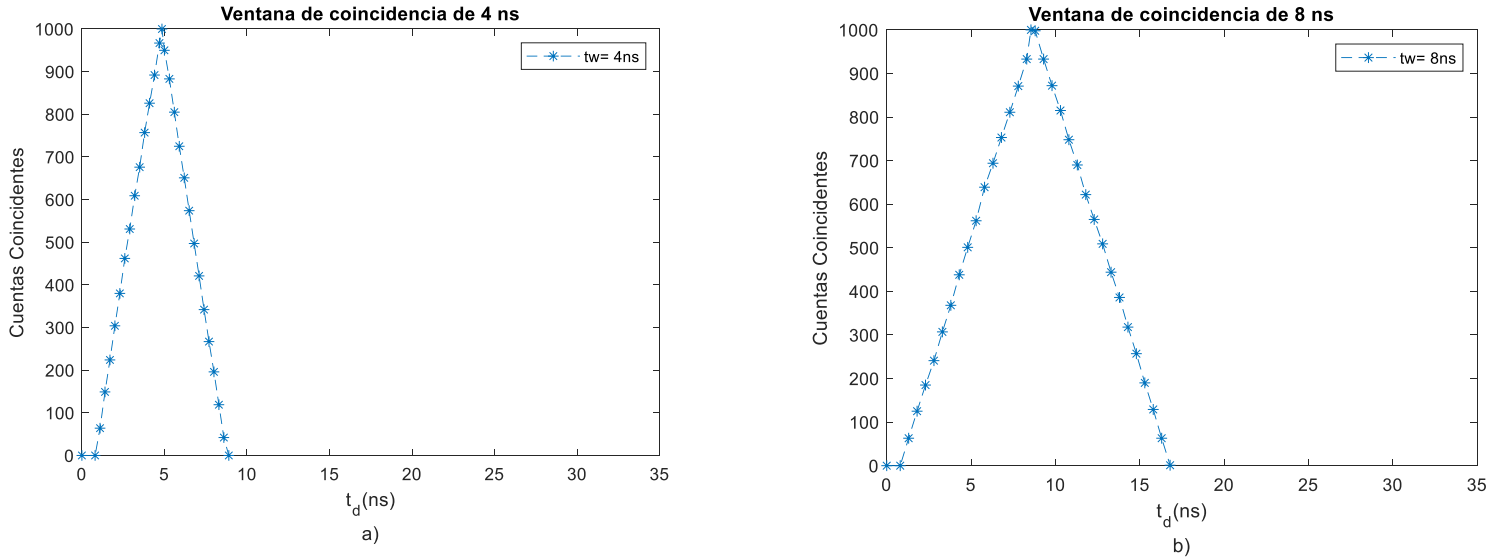


Fig. B.14 Ventanas de coincidencia con un bin activo
(a) Ventana de coincidencia de 4 ns, **(b)** Ventana de coincidencia de 8 ns.

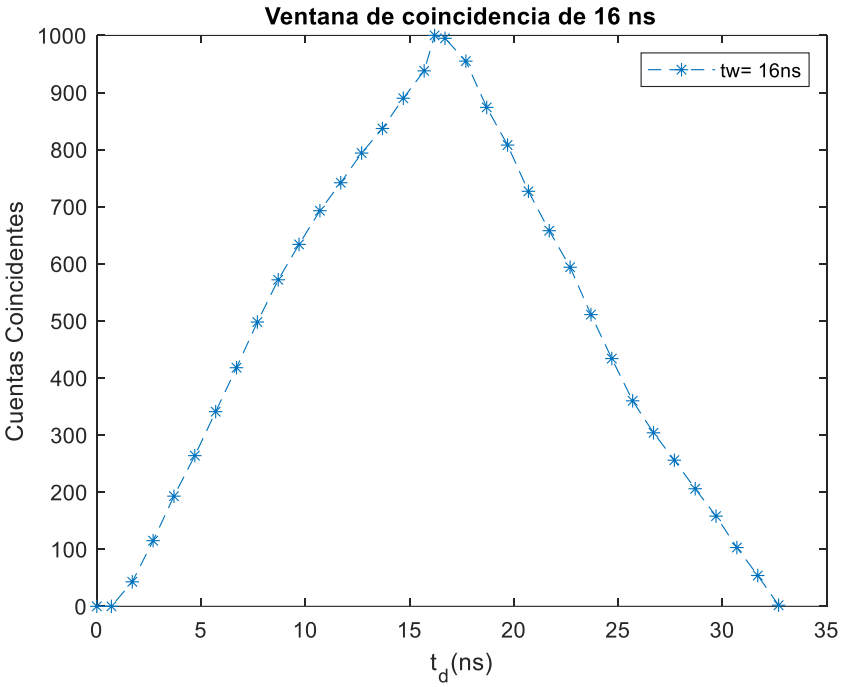


Fig. B.15 Ventana de coincidencia de 16 ns a un bin activo

Para el canal 2 con el canal 3:

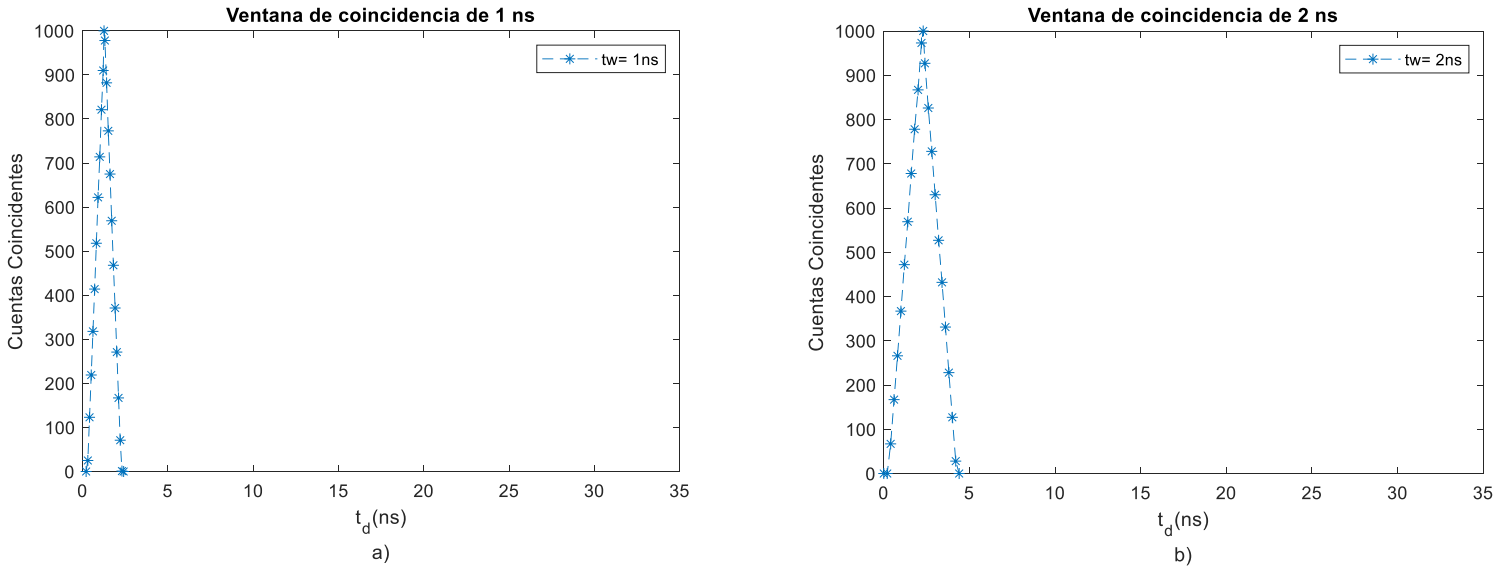
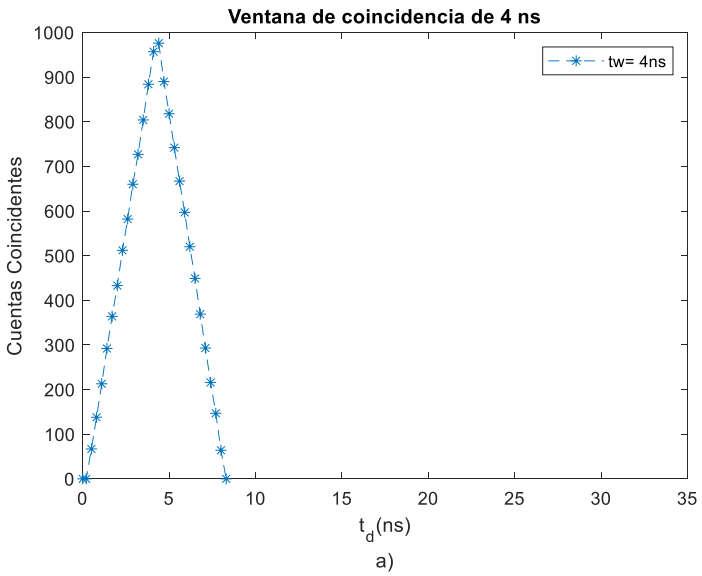


Fig. B.16 Ventanas de coincidencia con un bin activo
(a) Ventana de coincidencia de 1 ns, **(b)** Ventana de coincidencia de 2 ns.



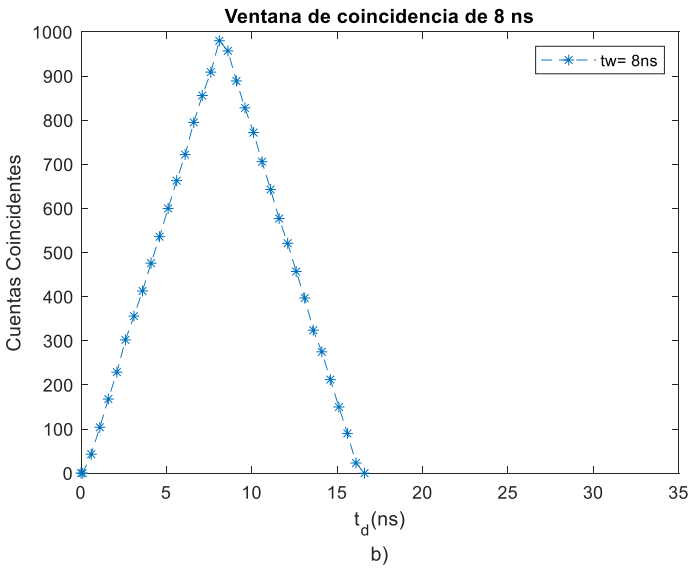


Fig. B.17 Ventanas de coincidencia con un bin activo
(a) Ventana de coincidencia de 4 ns, **(b)** Ventana de coincidencia de 8 ns.

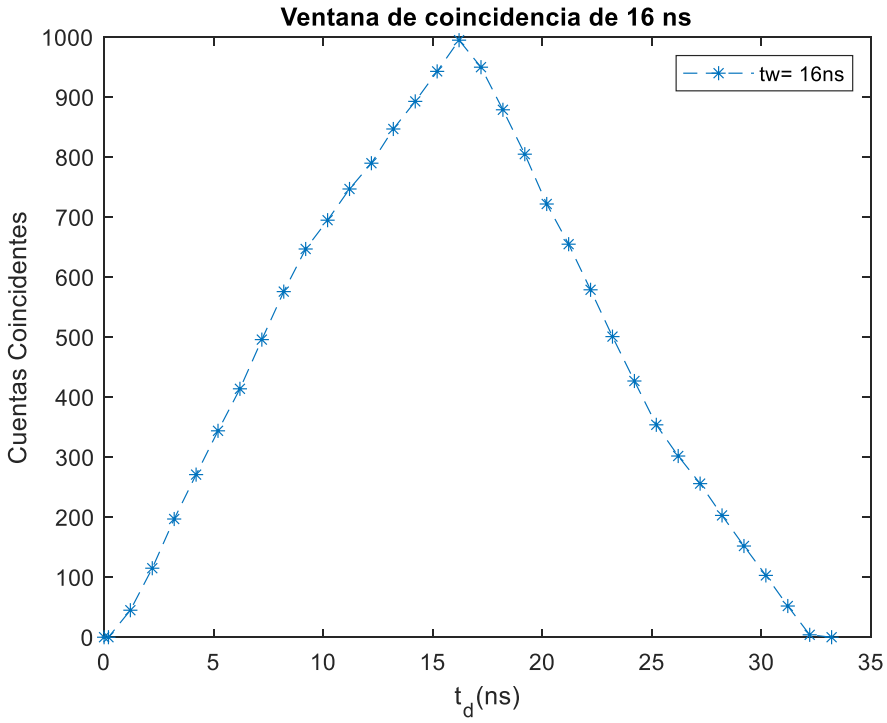


Fig. B.18 Ventana de coincidencia de 16 ns a un bin activo