UNIVERSIDAD NACIONAL "PEDRO RUIZ GALLO"
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
ESCUELA PROFESIONAL DE ELECTRÓNICA

"Sistema en FPGA para proporcionar marcas de tiempo basadas en GPS a sistemas de adquisición en el Radio Observatorio de Jicamarca"

TESIS

Para optar el título profesional de Ingeniero Electrónico

Presentado por:

Bach. Vásquez Ortiz Víctor Eduardo

Asesor:

Mg. Romero Cortez Oscar Ucchelly

Co-Asesor:

Dr. Milla Bravo Marco Antonio

LAMBAYEQUE – PERÚ

2018

UNIVERSIDAD NACIONAL "PEDRO RUIZ GALLO"

FACULTAD DE CIENCIAS FÍSICAS Y MATEMÍTICAS

ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA

"Sistema en FPGA para proporcionar marcas de tiempo basadas en GPS a sistemas de adquisición en el Radio Observatorio de Jicamarca"

Bach. Vásquez Ortiz Víctor Eduardo

Autor

Mg. Romero Cortez Oscar Ucchelly

Asesor

Dr. Milla Bravo Marco Antonio

Co-Asesor

Lambayeque – Perú

Noviembre-2018

UNIVERSIDAD NACIONAL" PEDRO RUIZ GALLO"

FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS

ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA

Los firmantes, por la presente certifican que han leído y recomiendan a la Facultad de Ciencias Físicas y Matemáticas la aceptación de la tesis titulada **"Sistema en FPGA para proporcionar marcas de tiempo basadas en GPS a sistemas de adquisición en el Radio Observatorio de Jicamarca"**, presentada por el bachiller en ingeniería electrónica, Vásquez Ortiz Víctor Eduardo, en el cumplimiento parcial de los requisitos necesarios para la obtención del título profesional de ingeniero electrónico.

Ing. Ramírez Castro Manuel Javier

Presidente Jurado de Tesis

Ing. Oblitas Vera Carlos Leonardo

Secretario Jurado de Tesis

Mg. Nombera Lossio Martín Augusto

Vocal Jurado de Tesis

Fecha de Defensa: 23 de Noviembre - 2018

Agradecimientos

A mis padres y hermano por su apoyo incondicional a lo largo de mi vida.

Al Radio Observatorio de Jicamarca, sede científica del Instituto Geofísico del Perú financiada por la Fundación Nacional de Ciencias de Estados Unidos mediante un convenio de colaboración entre el IGP y la Universidad de Cornell.

A Danny Scipión por involucrarme en el ROJ como tesista, a Joaquín Verástegui por su apoyo en el desarrollo del proyecto, a Marco Milla por su paciencia y guía, y al personal del observatorio que siempre estuvo dispuesto a ayudarme.

A Óscar Romero por su asesoría.

Dedicatoria

A mis padres Eduar y Gloria.

Presentación

En la presente tesis se expone la investigación y trabajos realizados con el objetivo de solucionar el problema de cronometraje de datos obtenidos por los radares del Radio Observatorio de Jicamarca, lo cual se logró gracias al desarrollo de un módulo hardware que a partir de información del Sistema de Posicionamiento Global registra el tiempo de transmisión o adquisición de señales radar.

Resumen

En este trabajo se presenta el diseño e implementación del sistema GCTS (GPS Controlled Timing System) el cual es un módulo hardware que responde a la necesidad del Radio Observatorio de Jicamarca de cronometrar los eventos registrados por los radares de sus instalaciones. El GCTS se desarrolló en un FPGA usando el lenguaje VHDL con un diseño estructural que facilitó una organización modular de los distintos bloques de funcionamiento. El sistema permite extraer información de tiempo UTC a partir de un receptor GPS Trimble, con dicha información se forma un reloj con una precisión basada en la señal de referencia del receptor GPS, con lo cual es posible cronometrar una señal de excitación externa (en este caso la señal de transmisión de pulso de radar) asignándole una marca de tiempo de 8 bytes. Estas marcas son almacenadas en una memoria junto con registros de configuración y operación a los cuales se accede mediante una interfaz SPI. Las pruebas realizadas en conjunto con los equipos de adquisición del ROJ validaron la confiabilidad y portabilidad del sistema, donde se obtuvo una precisión inferior a 50ns, menor que la precisión de 1µs solicitada, y una velocidad de transferencia SPI de hasta 15MHz.

Palabras clave: cronometraje, GPS, marca de tiempo, sincronización, FPGA

Abstract

This work presents the design and implementation of the GCTS (GPS Controlled Timing System) which is a hardware module that answers the Jicamarca Radio Observatory necessity of timing the events registered by the radars on its facilities. The GCTS was developed on a FPGA using VHDL language under a structural design that enabled a modular organization of the various functional blocks. The system allows the extraction UTC time information from a GPS Trimble receiver, with such information it creates a clock with a precision based on the reference signal from the GPS receiver, thus making it possible to timestamp an external excitation signal (in this case the radar transmission pulse) assigning it a timestamp of 8 bytes. These timestamps are stored in a memory along with configuration and operation registers which are accessed by an SPI interface. The tests made together with JRO's acquisition equipment validated the reliability and portability of the system. A 50ns accuracy, less than the microsecond accuracy requested, and a SPI transfer rate as high as 15MHz were achieved.

Keywords: timing, GPS, timestamp, synchronization, FPGA

Índice

1 Introducción

Desde su creación en 1961, el Radio Observatorio de Jicamarca (ROJ) ha estado a la vanguardia en los estudios de la atmósfera, contando con una ubicación única al estar situado en el ecuador magnético (S 11.95°, O 76.87°), y con uno de los radares más grandes y potentes del mundo compuesto por cuatro transmisores de 1.5 MW y 18,432 antenas dipolo en un área total de 85,000m^2.

A lo largo de los años, con desarrollo de hardware y software propio, el ROJ ha producido una gran cantidad de artículos de investigación, ofreciendo los datos obtenidos a la comunidad científica. Es por esta razón que gracias al afán de seguir mejorando la calidad de la información brindada, se llevó acabo este proyecto, el cual desarrolla el sistema GCTS (GPS Controlled Timing System) que es capaz de cronometrar a una escala precisa y de uso global cada evento registrado por los radares del ROJ.

1.1 Planteamiento del problema

Actualmente, las señales obtenidas por los radares del ROJ son digitalizadas por un sistema de adquisición para luego ser enviadas como datos a una computadora, la cual les asigna una marca de tiempo.

El problema radica en que:

- El tiempo asignado a los datos corresponde al tiempo de llegada a la computadora y no al tiempo de adquisición.
- Sincronizar el tiempo de la computadora con una escala de tiempo estándar a través de internet no es una opción, pues la máxima precisión que se puede alcanzar va

desde 1ms a 1s ([1] y [2]), sin contar los ajustes y saltos que se vuelven necesarios cuando el reloj local se desincroniza.

- La resolución normal que ofrece la computadora es en milisegundos y como en muchos experimentos realizados en el ROJ los periodos entre cada pulso de transmisión del radar están en el orden de unos pocos milisegundos (2.5ms por ejemplo), se necesita una escala con una resolución mayor.
- El periodo trascurrido entre dos marcas de tiempo consecutivas cualesquiera no es el mismo pues la frecuencia del cristal de cuarzo dentro del reloj de tiempo real (RTC) de la computadora no es estable.
- Los datos obtenidos no se pueden comparar con los de otras estaciones pues no se cuenta con una escala de referencia.

Ante esta situación, se debía utilizar un reloj capaz de proporcionar el tiempo en una escala estándar y con una precisión en microsegundos, para que un sistema de cronometraje pueda extraer esta información y provea marcas de tiempo a los datos obtenidos por los radares.

Como en el ROJ se cuenta con receptores GPS que ofrecen señales de referencia que alcanzan una precisión de hasta 15ns y ofrecen el tiempo en las escalas GPS o UTC, desde el principio se decidió que estos equipos serían la fuente de tiempo y la escala estándar elegida debería ser UTC pues es la escala establecida a nivel global.

Se decidió diseñar e implementar el sistema de cronometraje en un FPGA pues este dispositivo ofrece una gran versatilidad en el control y manejo de señales externas, capacidad de sincronismo y comunicación entre múltiples procesos trabajando en paralelo, y una arquitectura que permite trabajar a altas frecuencias.

Por lo tanto, todo lo previamente mencionado conduce a la formulación de la siguiente interrogante:

¿De qué manera un sistema en FPGA permite proporcionar marcas de tiempo basadas en GPS a sistemas de adquisición de radar?

1.2 Antecedentes

Abeysekara et al. (2014), en su artículo "GPS Timing and Control System of the HAWC Detector" (Sistema GPS de sincronización y control del detector HAWC [3]) describen el diseño y desempeño del sistema de Sincronización y Control GPS (GTC) del Observatorio de Rayos Gamma HAWC (High Altitude Water Cerenkov). El sistema, implementado en FPGA, proporciona marcas de tiempo totalmente sincronizadas con GPS con una precisión menor a $1\mu s$ para cada evento registrado en HAWC. Estas marcas son introducidas al sistema de adquisición de datos principal inmediatamente después de la etapa de recepción para evitar desfases con los datos registrados.

En el monitoreo continuo del sistema hecho en el 2013 se determinó que la precisión de las marcas de tiempo producidas por el sistema tenía un límite máximo de $25ns$, que era más que suficiente para la precisión de $1\mu s$ requerida por el observatorio.

Włodarczyk et al. (2013) en su artículo "Multi-Channel Data Acquisition System with Absolute Time Synchronization" (Sistema multicanal de adquisición de datos con sincronización de tiempo absoluta [2]) describen un sistema de adquisición de datos con marcas de tiempo asignadas automáticamente, donde el tiempo es proporcionado por el receptor GPS: "Resolution T" del fabricante Trimble.

El sistema está basado en un microcontrolador y logra una precisión con respecto al tiempo GPS o UTC mejor que $\pm 0.2\mu s$ con un retraso constante de aproximadamente $1.5\mu s$ para mediciones simultáneas de hasta 4 canales analógicos y 2 canales digitales con una frecuencia máxima de 1KS/s.

El sistema GCTS tiene características similares al sistema GTC, pues está basado en FPGA y ofrece una precisión menor a $1\mu s$, pero a diferencia de este, proporciona las marcas de tiempo a través de una interfaz externa.

La similitud con el trabajo de Włodarczyk et al. (2013), está en que ambos sistemas usan un receptor GPS del fabricante Trimble utilizando el mismo protocolo de comunicación, y también alcanzan una precisión menor a $1\mu s$.

Asimismo, los sistemas mencionados anteriormente son parte de un sistema de adquisición más complejo, que es lo que se quiere lograr en un futuro con el sistema GCTS.

1.3 Hipótesis

Si se diseña e implementa un sistema en FPGA capaz de proporcionar marcas de tiempo a partir de un receptor GPS a una señal de control externa, y se realizan las pruebas correspondientes para verificar su operación, entonces se logrará proporcionar dichas marcas con exactitud y precisión a sistemas de adquisición de radar.

1.4 Objetivos

1.4.1 Objetivo General

Diseñar e implementar un sistema en FPGA para proporcionar marcas de tiempo basadas en GPS a sistemas de adquisición de radar.

1.4.2 Objetivos Específicos

- Diseñar e implementar un sistema capaz de proporcionar marcas de tiempo, con una precisión en el orden de microsegundos, a una señal de control externa.
- Construir un sistema de reloj en base a la escala UTC.
- Diseñar e implementar un módulo de comunicación de alta velocidad para proporcionar la marca de tiempo a aplicaciones externas.
- Diseñar e implementar un sistema modular y portable para que pueda ser usado como parte de otros sistemas de adquisición.
- Realizar pruebas reales y verificar el comportamiento del sistema en conjunto con los equipos de adquisición del ROJ.

1.5 Marco Teórico

1.5.1 Sistema de Posicionamiento Global (GPS)

En [4] se define: el Sistema Global de Posicionamiento Navstar como un sistema de radio posicionamiento y transferencia de tiempo. GPS provee una información precisa de posición,

velocidad y tiempo (PVT) a un número ilimitado de usuarios en tierra, mar, aire y espacio. Esta información está disponible mundialmente en cualquier condición climática para rastreadores pasivos. Normalmente el sistema incluye funciones que permiten el acceso a la precisión completa del servicio solo usuarios autorizados.

GPS incluye tres segmentos: Espacial, Control y Usuario (Figura 1.1). El segmento espacial consiste en una constelación nominal formada por 24 satélites Navstar. Cada satélite envía códigos de distancia y mensajes de datos de navegación. El segmento de control está formado por una red de estaciones de seguimiento y control que son usadas para maniobrar la constelación de satélites y actualizar los mensajes de navegación de éstos. El segmento del usuario consiste en una variedad de equipos receptores de radio navegación diseñados específicamente para recibir, decodificar y procesar los códigos de distancia de los satélites GPS y los mensajes de datos de navegación.

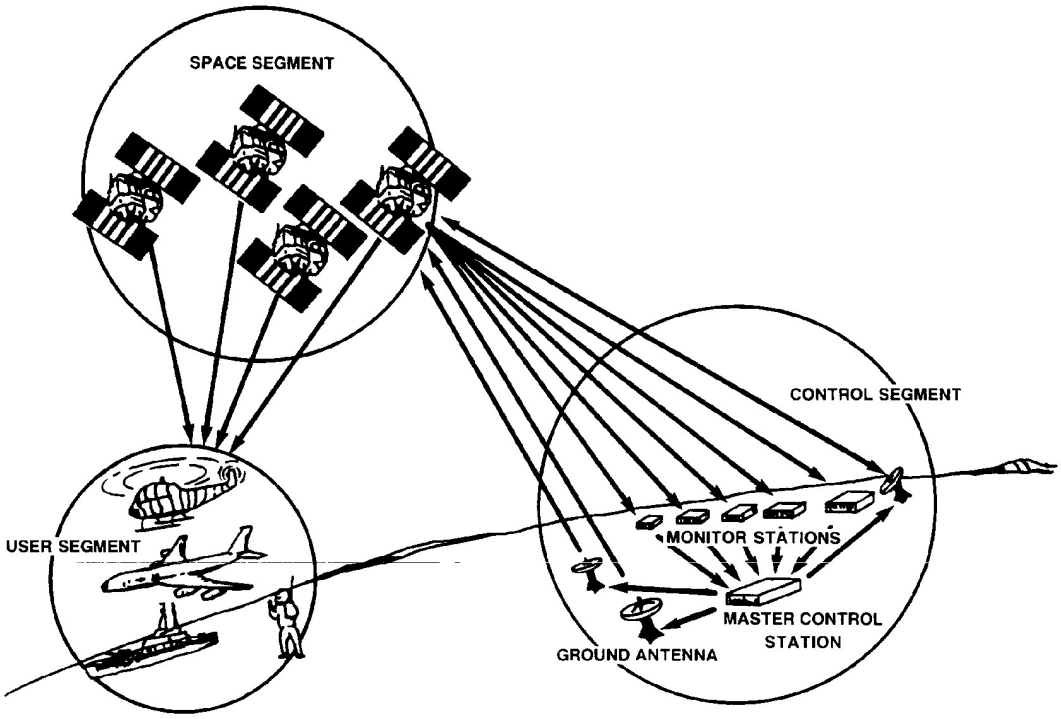


Figura 1.1 Segmentos GPS. Imagen tomada de Navstar GPS: User Equipment Introduction [4].

1.5.1.1 Receptor GPS

Según [5] la mayoría de receptores GPS pueden:

- Rastrear de 8 a 12 satélites.

- Proveer señales de tiempo y frecuencia derivadas del promedio de todos los satélites en línea de vista.
- Proveer información de fecha y hora en un formato legible por computadora, usualmente a través de RS-232 o interfaces similares.
- Proveer una señal eléctrica de 1 pulso por segundo (PPS), la cual puede ser fácilmente configurada para estar sincronizada hasta 100ns UTC o menos.

1.5.2 Tiempo

1.5.2.1 Escalas de tiempo

En [6] se define una escala de tiempo como una manera inequívoca de ordenar eventos. Es creada midiendo unidades de tiempo pequeñas (como el segundo) y luego contando el número de segundos transcurridos para establecer intervalos de tiempo más largos, como minutos, horas y días. El dispositivo que hace el conteo es llamado reloj.

Según la Oficina Internacional de Pesas y Medidas [7], el segundo es la duración de 9 192 631 770 oscilaciones de la radiación emitida en la transición entre los dos niveles hiperfinos del estado fundamental del isótopo 133 del átomo de cesio.

A continuación se describen las escalas de tiempo UT, TAI y UTC según el Anexo 1 de la Recomendación ITU-R TF.460-6 (Emisiones de frecuencias patrón y señales horarias) [8], después se describe la escala GPS.

1.5.2.2 Tiempo Universal (UT)

El UT es la designación general de las escalas de tiempo basadas en la rotación de la Tierra. En las aplicaciones en las que no se puede tolerar una imprecisión de unas centésimas de segundo, es necesario especificar las formas de UT que deben utilizarse:

- **UT0** es el tiempo solar medio, del meridiano origen, obtenido a partir de observaciones astronómicas directas.
- **UT1** es el UT0 con correcciones de los ligeros movimientos de la Tierra con relación al eje de rotación (variación polar). Corresponde directamente a la posición angular de la Tierra en torno a su eje de rotación diurna.
- **UT2** es el UT1 con corrección de los efectos de las pequeñas fluctuaciones estacionales en la velocidad de rotación de la Tierra.

1.5.2.3 Tiempo Atómico Internacional (TAI)

La escala de referencia internacional de timepo atómico (TAI), basada en el segundo (SI), la forma la Oficina Internacional de Pesas y Medidas (BIPM) con la información de reloj facilitada por establecimientos colaboradores. Tiene forma de escala continua, es decir, en días, horas, minutos y segundos, desde el 1 de enero de 1958. Fue aprobada por la Conferencia General de Pesas y Medidas (CGPM) en 1971.

1.5.2.4 Tiempo Universal Coordinado (UTC)

El UTC es la escala de tiempo mantenida por la BIPM, con la participación del Servicio Internacional de Rotación de la Tierra y Sistemas de Referencia (IERS), y constituye la base de una difusión coordinada de frecuencias patrón y señales horarias. Corresponde exactamente en cuanto al régimen de transcurso con el TAI aunque difiere de él en un número entero de segundos.

La escala de UTC se ajusta mediante inserción u omisión de segundos (segundos intercalares positivos o negativos) necesarios para asegurar una concordancia aproximada con UT1.

Desde el 1 de Enero de 1958 UT1, cuando la escala fue introducida, hasta 1972, fue ajustada con saltos de $\pm 0.1s$ o 0.05s TAI, con un último salto de +0.107758s TAI en el año nuevo de 1972 con el cual coincidió con las 00:00:10 del 1 de Enero de 1972 TAI. A partir de esa fecha es que UTC solo difiere de TAI en un número entero de segundos, además de mantenerse en concordancia de $\pm 0.9s$ TAI con la escala UT1.

Hasta la fecha (Agosto 2018) se han añadido 27 segundos intercalares positivos y más los 10 segundos de desfase inicial hacen un total de 37 segundos de retraso con respecto a TAI.

1.5.2.5 Tiempo GPS

En [9] se define el tiempo GPS como una escala de tiempo continua mantenida por el Segmento de Control del GPS, que empezó en la medianoche del 5/6 de Enero de 1980 en la escala del Tiempo Universal Coordinado según lo establecido por el Observatorio Naval de los Estados Unidos (USNO).

Como esta escala no puede tolerar la introducción de segundos intercalares, cuando en 1980 el Departamento de Defensa de los Estados Unidos comenzó a mantener el tiempo en

los satélites GPS, su sistema y frecuencia fueron fijados a coincidir con el tiempo UTC. En ese momento la diferencia del tiempo TAI y el tiempo UTC era de 19 segundos. Desde entonces, el tiempo UTC ha sido retrasado varios segundos intercalares mientras que el tiempo GPS no. Por lo tanto, el tiempo GPS sigue estando muy cerca del tiempo TAI - 19s. La especificación en el tiempo GPS es de que no se debe desviar más de un microsegundo del módulo de un segundo del tiempo UTC. En la práctica el rendimiento es mayor, llegando a una desviación máxima de 40 nanosegundos. [10].

1.5.2.6 Tiempo POSIX

También conocido como tiempo UNIX. POSIX es el acrónimo de Interfaz de Sistema Operativo Portable donde X viene de UNIX.

En el estándar [11] se detalla lo siguiente: POSIX.1-2008 es un conjunto de normas, especificadas por la IEEE en IEEE Std 1003.1™-2008 y The Open Group Technical Standard Base Specifications - Issue 7, para sistemas basados en UNIX. Dicha norma define una interfaz estándar del sistema operativo y el entorno, incluyendo un intérprete de comandos (o "shell"), y programas de utilidades comunes para apoyar la portabilidad de las aplicaciones a nivel de código fuente.

La noción de "tiempo" para la mayoría de sistemas es la de un valor en constante crecimiento, este valor debería incrementar incluso durante los segundos intercalares. Sin embargo, no solo la mayoría de sistemas no llevan registro de segundos intercalares, cuya adición o sustracción no es predecible, sino que tampoco están sincronizados a un estándar de tiempo referencial. Mientras el UTC incluye los segundos intercalares, en sistemas UNIX y derivados la interpretación de tiempo se toma como los segundos transcurridos desde la época UNIX (00:00:00 del 1 de Enero de 1970 UTC) sin tomar en cuenta los segundos intercalares, por esto cada día cuenta con exactamente 86400 segundos. Así, el tiempo POSIX desglosado no es necesariamente UTC a pesar de parecerlo y brinda un método fácil y compatible en el cómputo de diferencias entre instantes de tiempo.

La representación UNIX es válida ya que es inapropiado requerir que un tiempo representado como los segundos transcurridos desde la época UNIX represente con precisión el número de segundos entre ese instante de tiempo y la época, pues es suficiente que las aplicaciones del sistema traten ese valor como si representará el número de segundos, siendo responsabilidad del administrador del sistema asegurar la utilidad de este valor para

dichas aplicaciones. Por eso, es importante que todos los sistemas que hagan uso del tiempo POSIX, interpreten 536457599 segundos desde la época como 59 segundos, 59 minutos, 23 horas del 31 de Diciembre de 1986, sin importar la precisión del sistema.

A pesar de que últimamente se ha hecho un esfuerzo en migrar a arquitecturas de 64-bits, la mayoría de sistemas UNIX aún representan el tiempo usando un tipo de datos *time_t* (entero con signo de 32-bits), por lo cual se espera que para el año 2038, *time_t* se desborde ocasionando fallos en dichos sistemas.

1.5.3 Protocolos de comunicación

1.5.3.1 Estándar RS-232

Citando [12]: RS-232 es un estándar de comunicación serial, que ha sido desarrollado por la Alianza de Industrias Electrónicas (EIA) y la Asociación de la Industria de Telecomunicaciones (TIA), es comunmente conocido como "RS-232" donde RS significa estándar recomendado. En años recientes este sufijo se ha reemplazado por "EIA/TIA" para ayudar a identificar la fuente del estándar.

RS-232 es un estándar completo, ya que asegura compatibilidad entre el anfitrión y sistemas periféricos, especificando:

1. Voltaje común y niveles de señal. Ver Tabla 1.1.
2. Mínimo control de información entre el host y los sistemas periféricos. El estándar define 4 categorías de señales: común, data, control y sincronismo, en total hacen 24 señales de las cuales 8 son las más usadas y se pueden ver en la Tabla 1.2.
3. Configuración de cableado común a través de una interfaz mecánica. Aunque el estándar solo especifica un conector tipo D de 25 pines (DB25), se debe aclarar que a menudo este conector no es usado, debido a que la mayoría de aplicaciones no necesitan todas las señales definidas en el estándar, por eso es común el uso de otros conectores, entre ellos el más popular DB9. Ambas conexiones se ven en la Fig. 1.2

1.5.3.2 Protocolo SPI

Del inglés Serial Peripheral Interface, el protocolo SPI se define en [13] como un estándar de datos seriales sincronizados, principalmente usado para permitir a un microprocesador

Tabla 1.1 RS-232: Niveles de señal.

	Transmisor	**Receptor**
Nivel Alto (0)	de +5V a +15V	de +3V a +13V
Nivel Bajo (1)	de -5V a -15V	de -3V a -13V

Tabla 1.2 Principales señales definidas en estándar RS-232.

abrev.	**Señal**	**Categoría**
TD	Transmitted Data	Data
RD	Received Data	
RTS	Request to Send	Control
CTS	Clear to Send	
DSR	DCE Ready	
DTR	DTE Ready	
RI	Ring Indicator	
DCD	Data Carrier Detect	

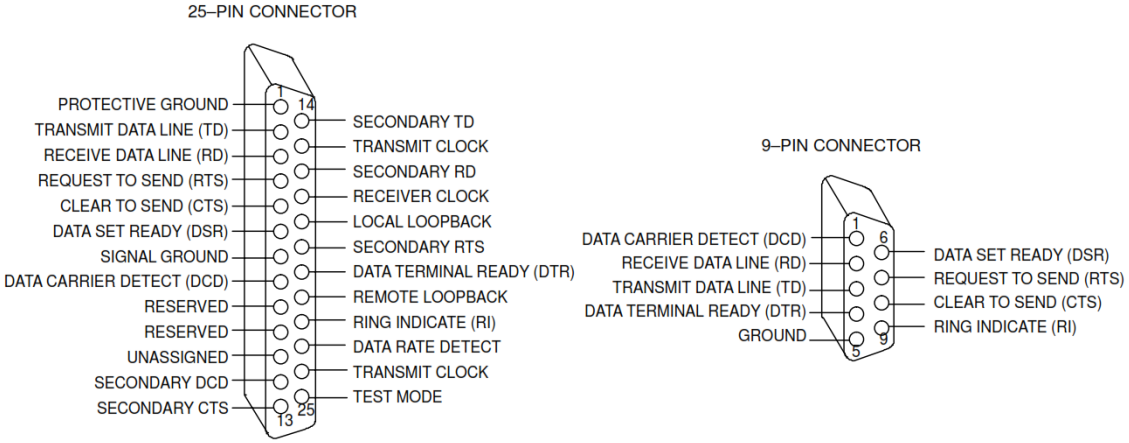


Figura 1.2 Conexiones RS-232. Imagen tomada de Fundamentals of RS-232 Serial Communications [12].

comunicarse con otros microprocesadores o circuitos integrados como memorias, LCD, ADC, etc.

SPI es un sencillo protocolo maestro/esclavo basado en 4 líneas:

- Línea de reloj (SCLK)
- Salida serial (MOSI)
- Entrada serial (MISO)
- Seleccionar esclavo (SS)

Cada sistema SPI consiste de un maestro y uno o más esclavos, donde un maestro inicia la comunicación manteniendo activa la línea SS. Cuando un dispositivo esclavo es seleccionado, el maestro empieza a enviar la data de salida a través de la línea MOSI al esclavo seleccionado. El maestro envía y recibe un bit por cada ciclo del reloj. Un byte puede ser intercambiado en 8 ciclos de reloj. El maestro termina la comunicación desactivando la línea SS.

SPI es un protocolo primitivo sin un mecanismo de confirmación para revisar la información enviada o recibida. Para una comunicación segura, un control de flujo debe ser implementado en el protocolo de comunicación o en una capa más alta.

1.5.4 Radar

Por [14] se puede decir que un radar usa ondas de radio para detectar la presencia de objetos y hallar su posición. La palabra radar, usada por primera vez por la Marina de los Estados Unidos en 1940, representa: ***RA****dio* ***D****etection* ***A****nd* ***R****anging*, indicando así sus dos propósitos: detección y ubicación.

En [15] se presenta la Figura 1.3, la cual muestra los elementos básicos implicados en el proceso de transmisión, propagación, reflección y recepción de la señal (el sistema puede ser más simple o más complejo).

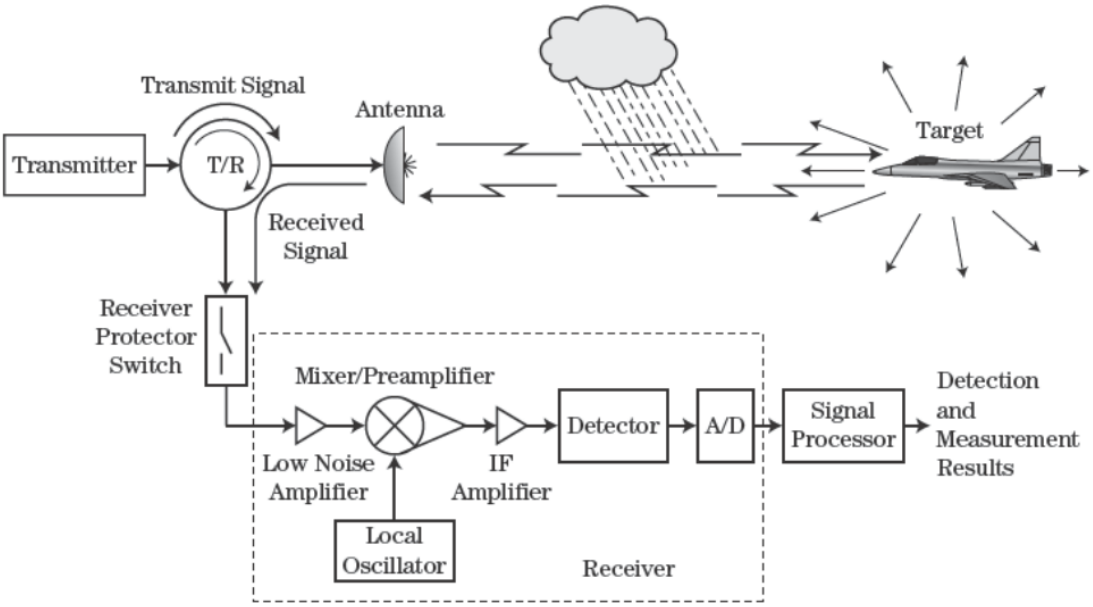


Figura 1.3 Elementos básicos en la operación de radar. Imagen tomada de [15].

El transmisor genera las ondas electromagnéticas y se conecta a la antena través de un

transmisor/receptor (T/R), el cual tiene la función de proveer puntos de conexión para que el transmisor y el receptor puedan acoplarse a la antena. La señal transmitida se propaga en el ambiente hasta alcanzar su blanco. Algunas de estas señales se reflejan de vuelta hacia el radar que las captura con la antena y las pasa a los circuitos receptores. Los componentes en el receptor amplifican la señal y la transforman de Radio Frecuencia (RF) a Frecuencia Intermedia (IF), posteriormente pasan esta señal a un conversor analógico-digital (ADC) para finalmente enviar la data a un procesador de señales.

En [16] se detallan los dos tipos básicos de radares: radar de pulso y radar de onda continua.

El radar de pulso es el más convencional, transmite una señal de radio y luego espera por la energía (o eco) reflejada de vuelta hacia la antena. Después de un periodo de tiempo específico (que depende de que tan lejos el radar esta escaneando) se envía otro pulso seguido de un periodo de escucha. Como las ondas de radar viajan a la velocidad de la luz, la distancia de la señal de retorno puede ser calculada facilmente.

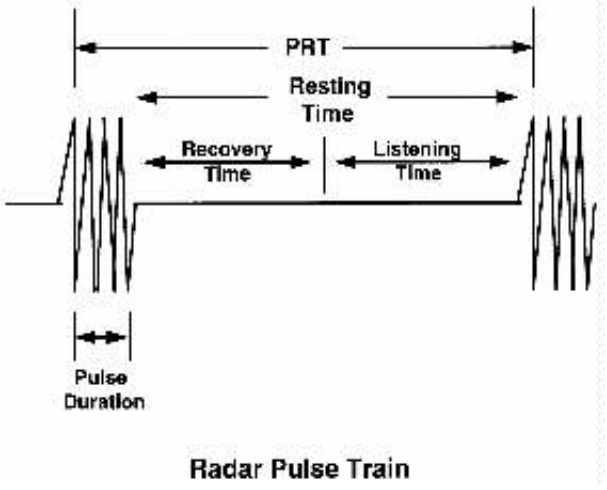
Los términos básicos para definir las características de un radar de pulso son:

- **Duración de pulso** (PD): El tiempo en el que el radar está transmitiendo una señal de radio frecuencia (RF). También se llama ancho de pulso (PW).
- **Tiempo de Repetición de Pulso** (PRT): Es el tiempo requerido para completar un ciclo de transmisión. Es el tiempo desde el inicio de un puslo de radar hasta el inicio del próximo. Representa el periodo de transmisión para un ciclo.
- **Frecuencia de Repetición de Pulso** (PRF): Es igual al número de pulsos por segundo que transmite el radar. Si se desea que el radar "observe" a largas distancias, es necesario un PRF bajo; para distancias más cortas, se puede usar un PRF más alto.

  La relación entre PRF y PRT es: PRF = 1/PRT
- **Tiempo de reposo**: Es el tiempo entre el final de una transmisión y el inicio de otra. Se divide en dos secciones: tiempo de recuperación y tiempo de recepción.
- **Tiempo de Recuperación** (RT): Representa el tiempo inmediato que sigue a la transmisión de la señal de RF. Debido a las leyes de la física, el radar no es capaz de procesar ecos de retorno durante este tiempo.

- **Tiempo de Recepción** (LT): Es la parte del tiempo de reposo en la que el radar puede recibir y procesar los ecos de retorno.

Un diagrama de los tiempos mencionados se puede ver en la Fig 1.4.



.

Figura 1.4 Tiempos en la operación de un radar de pulsos. Imagen tomada de [16].

1.5.5 Sistemas Digitales

1.5.5.1 Matrices de Puertas Programable por Campo (FPGA)

Según [17]: Un FPGA consiste en un arreglo de bloques lógicos programables, bloques de entrada/salida (E/S) y una red de interconexiones programable que puede ser usada para conectar un elemento lógico a cualquier otro. Cada bloque lógico contiene la circuitería necesaria para implementar lógica combinacional arbitraria, además de flip-flops D y multiplexores para manejo de señales. Esta arquitectura implementa eficazmente una macro-celda lógica de salida (OLMC) dentro de cada bloque, ofreciendo así máxima flexibilidad y mayores recursos para lógica secuencial. La Figura 1.5 muestra la arquitectura genérica de un FPGA.

1.5.5.2 VHDL

En [18] se define VHDL como un lenguaje de descripción de hardware, el cual describe el comportamiento de un circuito electrónico o sistema, del cual se puede implementar un circuito físico.

Sus siglas significan Lenguaje de descripción de Hardware VHSIC. VHSIC a su vez es una abreviación de Circuitos Integrados de Muy Alta Velocidad, una iniciativa fundada

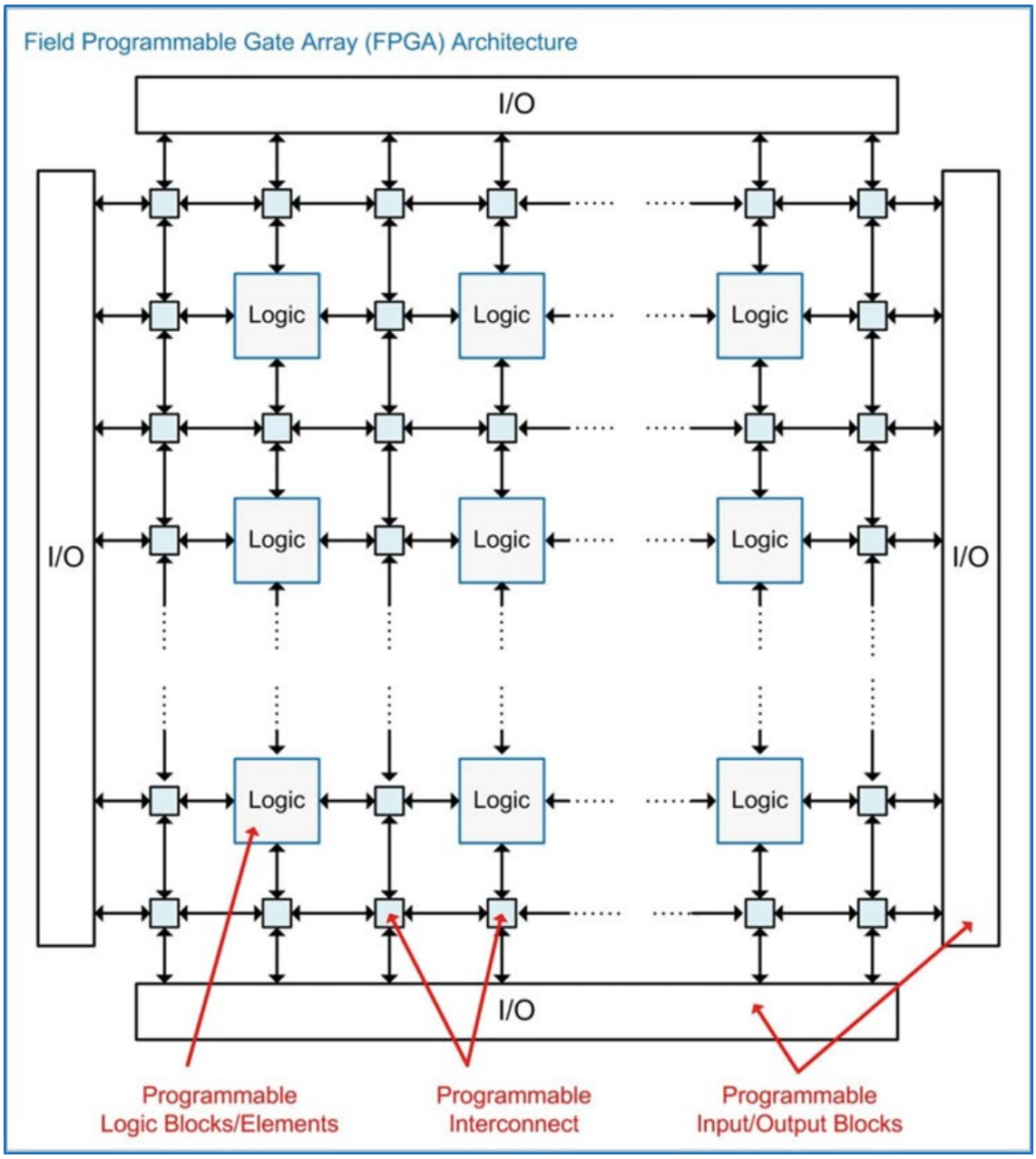


Figura 1.5 Arquitectura de un FPGA. Imagen tomada de [17].

por el Departamento de Defensa de los Estados Unidos en los años 1980 que llevó a la creación de VHDL. Su primera versión fue VHDL 87, luego mejorada a VHDL 93. VHDL fue el primer, y original, lenguaje de descripción de hardware en ser estandarizado por el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) en el estándar IEEE 1076.

VHDL esta diseñado para sintetizar y simular circuitos, sin embargo, aunque es completamente simulable, no todos los circuitos son sintetizables.

VHDL proporciona tres métodos básicos para describir un circuito digital por software: comportamental, flujo de datos y estructural.

1.5.5.3 Método estructural de programación en VHDL

De acuerdo a [19]:

El método estructural para escribir una descripción VHDL de una función lógica puede compararse con el montaje de circuitos integrados en una tarjeta de circuito y el establecimiento de las interconexiones entre ellos mediante cables. Con el método estructural, se describen las funciones lógicas y se especifica cómo se conectan entre sí. El componente VHDL es una forma de predefinir una función lógica para poder emplearla repetidas veces en un mismo programa o en otros programas. El componente puede utilizarse para describir cualquier cosa, desde una simple puerta lógica a una función lógica compleja. La señal VHDL es una forma de especificar una conexión mediante un "cable" entre componentes. Ver Figura 1.6.

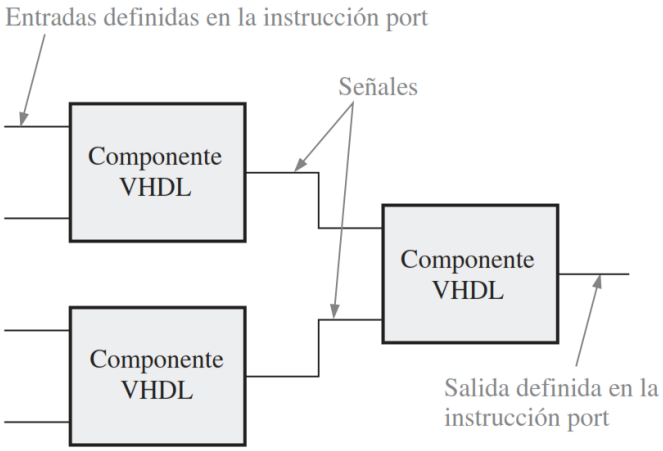


Figura 1.6 Implementación estructural VHDL. Imagen tomada de [19].

1.5.5.4 Metaestabilidad

En [20] se describe:

La salida de un flip-flop (FF) disparado por flancos tiene 2 estados válidos: alto y bajo. Para asegurar una operación segura, los diseños deben cumplir los requisitos temporales del FF. La entrada del FF debe estar estable por un tiempo mínimo antes del flanco de reloj (tiempo de establecimiento de registro o t_{SU}) y un tiempo mínimo después del flanco de reloj (tiempo de mantenimiento de registro o t_H). Valores específicos para t_{SU} y t_H se presentan en las hojas técnicas de cada familia de dispositivos.

En sistemas no-síncronos, si la señal asíncrona de entrada viola los requerimientos

temporales del FF, la salida de este puede volverse metaestable. Salidas metaestable oscilan entre alto y bajo por un período corto de tiempo, el cual puede causar fallas del sistema. Por lo tanto, se debe analizar las características de metaestabilidad de un dispositivo para determinar su fiabilidad en diseños de sistemas no-síncronos. En sistemas síncronos, las señales de entrada siempre cumple con los requerimientos temporales del FF; así pues, no se presenta la metaestabilidad.

Cuando un FF está en un estado metaestable, causa que la transición a un estado estable demore más de lo especificado por el retardo de propagación (t_{CO}). El tiempo adicional después de t_{CO} que le toma a una salida metaestable en pasar a un estado estable se llama tiempo de estabilización (t_{MET}).

Para reducir los efectos de metaestabilidad, comúnmente se usa un sincronizador multietapa en el cual dos o más FF en cascada forman un circuito de sincronización (ver Figura 1.7). Si el primer FF produce una salida metaestable, esta se puede estabilizar antes de que el segundo flanco de activación dispare el segundo FF. Este método no garantiza que el segundo FF no producirá un valor indefinido, pero incrementa dramáticamente la probabilidad de que la señal de entrada vaya a un estado válido antes de pasar al resto del circuito.

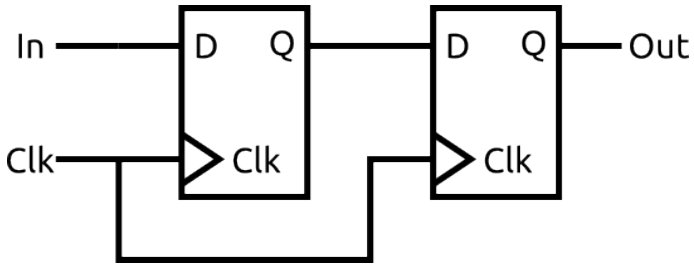


Figura 1.7 Sincronizador multietapa.

1.5.5.5 Máquina de estado finito (FSM)

En [17] se define:

Una FSM o máquina de estado, es un circuito que contiene un número predeterminado de estados, en el cual la máquina solo puede existir en un estado a la vez. El circuito transita entre estados a partir de un evento de disparo, comúnmente un flanco de reloj, y los valores de las entradas de la máquina; todas las posibles transiciones también están predeterminados. A través del uso de estados y las secuencias de transición pasadas, el circuito es capaz de decidir su siguiente estado, esto le permite crear salidas que son más

inteligentes comparadas a un circuito de lógica combinacional simple que solo tiene salidas basadas en el valor actual de sus entradas.

Hay dos tipos diferentes de condiciones de salida para una máquina de estado. El primero es cuando la salida solo depende del estado actual de la máquina, este tipo de sistema es llamado Máquina de Moore. El segundo es cuando las salidas dependen del estado actual y de las entradas al sistema, se lo conoce como Máquina de Mealy.

Las máquinas de estado tienen tres componentes principales: la memoria de estado, la lógica del siguiente estado, y la lógica de salida. El bloque "lógica del siguiente estado" es un grupo de lógica combinacional que produce el siguiente estado en base al estado actual y las entradas del sistema. El bloque "memoria de estado" contiene el estado actual del sistema y es actualizado en cada flanco de subida del reloj a través de FF tipo D (D-FF), los cuales deciden a que estado pasar. El bloque "lógica de salida" es otro grupo de lógica combinacional que crea las salidas del sistema en función del estado actual (y las entradas en la Máquina Mealy). Ver Figura 1.8

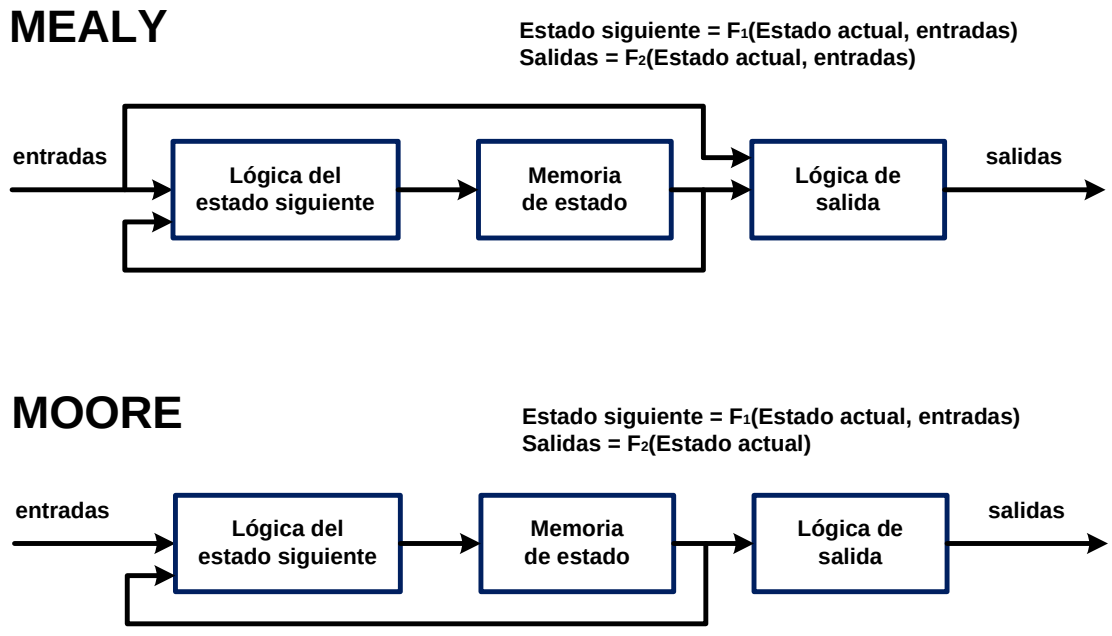


Figura 1.8 Tipos y componentes principales de una máquina de estado finito.

2 Materiales y Métodos

2.1 Metodología

Para lograr los objetivos de la investigación:

1. Se analizó las especificaciones técnicas de receptores GPS y componentes electrónicos para seleccionar los equipos y materiales idóneos a ser usados en el diseño del sistema.
2. Se diseñó un circuito VHDL para obtener fecha y hora exacta UTC a partir del receptor GPS.
3. Se diseñó un circuito VHDL de reloj en tiempo real con una resolución de microsegundos.
4. Se diseñó un circuito VHDL que utilice los datos de fecha y hora UTC y el reloj en microsegundos para generar una marca de tiempo sincronizada con una señal de control externa, que sería el pulso de transmisión del radar proporcionado por el Controlador de Radar del ROJ.
5. Se seleccionó una interfaz de transmisión de datos digital y se diseñó un circuito VHDL que usando dicha interfaz sea capaz de proporcionar la marca de tiempo a los sistemas de adquisición.
6. Se probó los circuitos diseñados usando plataformas de simulación y luego se implementaron independientemente en las tarjetas y se observó su comportamiento real.

7. Se integraron los circuitos para configurar el sistema que cumpla con el objetivo de la investigación.

8. Se probó el sistema en conjunto con los equipos de adquisición del ROJ para verificar su correcto funcionamiento.

9. Se configuraron y probaron 2 sistemas GCTS independientes para validar la portabilidad del sistema.

El diagrama de la solución planteada se muestra en la Fig. 2.1.

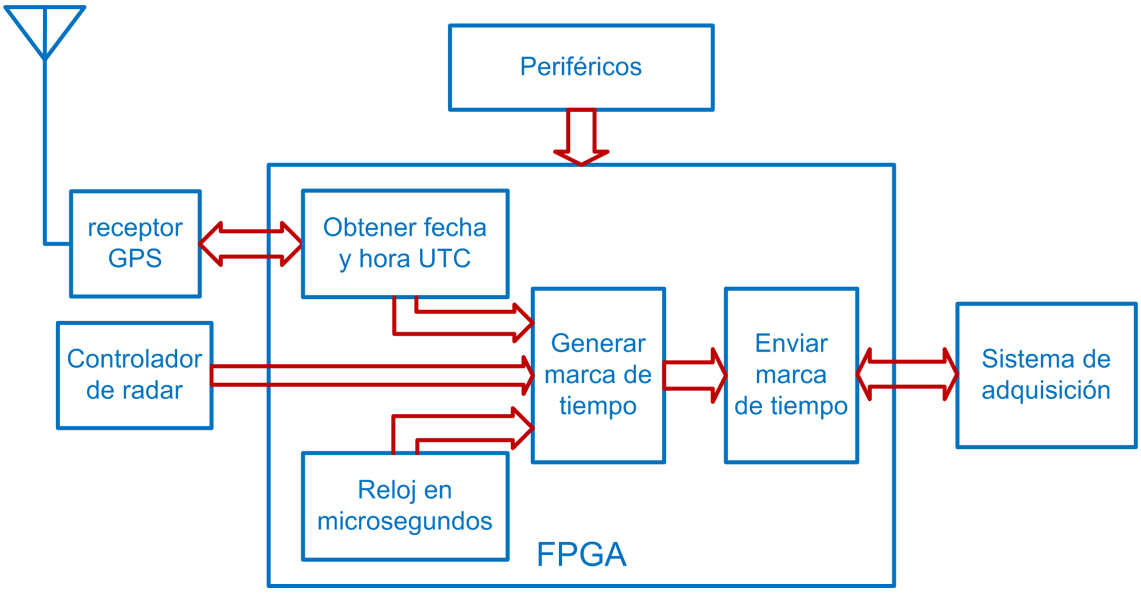


Figura 2.1 Modelo metodológico para el diseño e implementación del sistema GCTS.

2.2 Hardware y Software utilizados

- **Receptores GPS:**

  Se examinaron 3 receptores de GPS usados en el ROJ, éstos fueron: ThunderBolt E, módulo GPSDO para USRP N200 (en adelante GPSDO) y GPS 15xH-W, se pueden ver sus características en la Tabla 2.1.

  La primera propiedad que se tomó en cuenta a la hora de seleccionar el receptor GPS fue su comportamiento en el estado *holdover*.

  En [21] se define *holdover* como una condición de operación de un reloj que ha perdido su entrada de control y está usando data adquirida en el modo de operación

Tabla 2.1 Receptores GPS.

	ThunderBolt E	**GPSDO para USRP**	**GPS 15xH-W**
Fabricante	Trimble	Ettus Research	Garmin
Precisión PPS	15 ns (5V)	50 ns (3.3V)	$1\mu s$ (5V)
Holdover	<$4\mu s$ / día	$11\mu s$ / 3 h	X
Control RS-232	TSIP	NMEA, SCPI-99	NMEA
Precio ($)	∼1600	∼830	40 - 70
Reloj de referencia	10 MHz	10 MHz	X
Estabilidad de reloj	1.16E-12	2.5E-08	X

locked para controlar su salida. La información almacenada es usada para controlar las variaciones de fase y frecuencia, permitiendo reproducir la condición *locked* dentro de las especificaciones. El modo *holdover* empieza cuando la señal de salida del reloj ya no refleja la influencia de una referencia externa conectada y termina cuando el reloj vuelve a la condición *locked*.

En el caso especifico de los receptores GPS, la señal de control es la data obtenida de la red de satélites GPS que van corrigiendo al reloj mientras un filtro Kalman va aprendiendo el comportamiento de un Oscilador de Cristal Controlado por Horno (OXCO) a lo largo del tiempo y en respuesta a la temperatura. Una vez que se dejan de detectar señales GPS se entra en el modo *holdover*, del cual se retorna cuando se detectan las señales nuevamente.

Por esto se descartó el GPS 15xH-W pues no posee capacidad de *holdover*, además de que su precisión ya es de $1\mu s$, lo que iba a reducir el margen de error del sistema, pero sobretodo no contaba con una señal de reloj de referencia como si lo tienen los otros dos equipos, la cuál además de ser necesaria para el Controlador de Radar (equipo desarrollado en el ROJ), ayudaría en la sincronización de los módulos contadores del sistema.

A pesar de que el GPSDO cuenta con una salida PPS al nivel deseado de voltaje para las tarjetas FPGA (3.3V) y usa un protocolo de comunicación estándar establecido por la Asociación Nacional de Electrónica Marina (NMEA), se decidió usar el ThunderBolt E porque es el que se usa en los experimentos que se realizan en el ROJ, y tiene mejores características de precisión, *holdover* y estabilidad.

El receptor ThunderBolt E se usó para proveer al sistema GCTS con:

  - La señal PPS que reinicia los contadores internos.
  - Una señal de reloj apropiada con la que funcione el sistema y el Controlador de Radar, a partir de la señal sinusoidal de 10 MH.
  - Extraer la información de tiempo GPS y UTC.

- **Tarjetas FPGA:**

  Tarjetas de desarrollo sobre la cual se implementó la lógica del sistema y se realizaron las pruebas en las distintas etapas del proyecto. Como el sistema es un módulo hardware puede ser adaptado a cualquier dispositivo FPGA, las siguientes tarjetas solo se utilizaron para probar su desempeño, la primera es parte de los equipos a disposición en el ROJ y la segunda es adquisición de la UNPRG:

  - **ZedBoard:** Esta tarjeta está basada en el dispositivo XILINX Zynq-7000 All Programmable SoC (AP SoC), ver [22].

    Dispositivo: XC7Z020.

    Empaquetado: CLG484.

    Al ser un producto oficial, información adicional puede ser encontrada en la web oficial de XILIX.

  - **X-SP6-X9 Board:** Esta tarjeta está basada en un FPGA de XILINX de la familia Spartan-6, ver [23].

    Dispositivo: XC6SLX9.

    Empaquetado: TQG144.

    Está diseñada y fabricada por un vendedor de AliExpress de China (FPGA Board Store; Store No.620372) que puede ser encontrado en la siguiente dirección: https://www.aliexpress.com/store/620372/, la tarjeta se puede conseguir aquí. Para mayor descripción e información de la tarjeta también se pueden consultar los siguientes links: https://artofcircuits.com/product/spartan-6-fpga-development-board-xc6slx9-tqg144, http://www.microsolution.com.pk/product/xc6slx9-spartan6-fpga-development-board-pakistan/.

- **Tarjetas de desarrollo basadas en Microcontroladores y Microprocesadores:**

Se usaron para extraer las marcas de tiempo del sistema GCTS a través de un bus SPI y enviarlas a una computadora, en la cual serían almacenadas.

Se probaron distintas tarjetas y plataformas como Arduino Pro Mini, Arduino Zero, Teensy 3.2, TIVA C Series tm4c1294 y la minicomputadora Raspberry Pi. En la elección se buscó siempre la capacidad para comunicarse a un nivel de voltaje de 3.3 V, pues es el nivel al que se manejan las entradas y salidas de las dos tarjetas FPGA, y una mayor velocidad de trabajo para poder procesar rápidamente los datos adquiridos.

Las pruebas finales se realizaron con el Arduino Zero y el Teensy 3.2.

- **Osciloscopio:**

  Se utilizaron osciloscopios Tektronix de 4 canales con 200MHz de frecuencia de muestreo, para:

  – Comparar el desfase de la señal PPS entre los 3 modelos de receptores GPS y entre dos equipos ThunderBolt E para corroborar las especificaciones detalladas en los datasheet.

  – Observar el sincronismo y desfase entre la señales *CLK* (60 MHz) y *SYNC* que entran al sistema y la señal de 10 MHz del receptor GPS. Ver Figura 2.2.

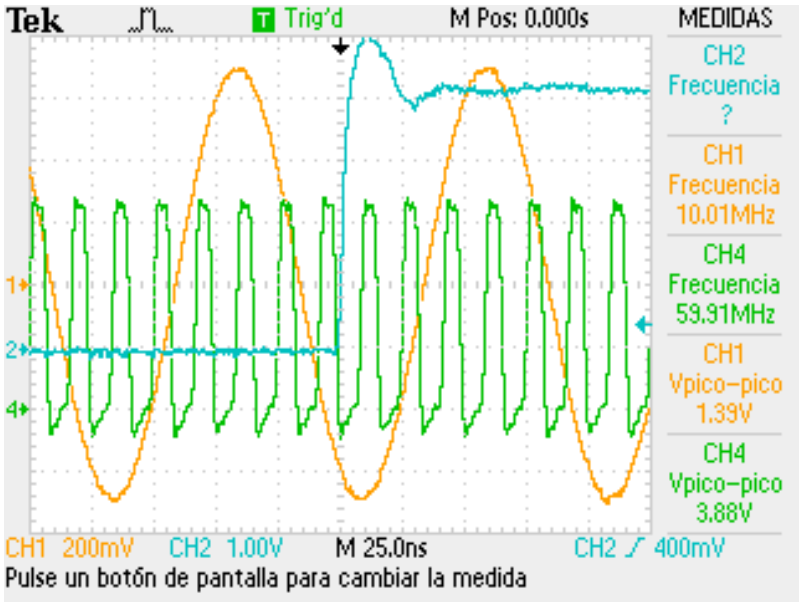


Figura 2.2 Sincronismo de señales: 10MHz (receptor GPS), *CLK* de 60MHz y *SYNC* (Controlador de Radar).

- Observar el jitter entre la señal CLK y la señal PPS.
- Comprobar que la señal SYNC fue programada adecuadamente para las pruebas.
- Ver el nivel de voltaje, la frecuencia y la forma de onda de las señales mencionadas anteriormente.
- Mediante la función "Persistence" del menú "Display" monitorear diversas señales a lo largo del tiempo durante las pruebas.
- Detectar crosstalk entre señales. En la Figura 2.3 se puede ver como la señal *SCLK* (en magenta) del módulo SPI esclavo afecta la señal *SS* (en cyan) causando un reinicio no deseado en la transmisión de las marcas de tiempo. La señal en amarillo es la misma señal *SS* pasada a través del FPGA y sirve para ver como el crosstalk causa los niveles altos de la línea que son los que detecta el FPGA.

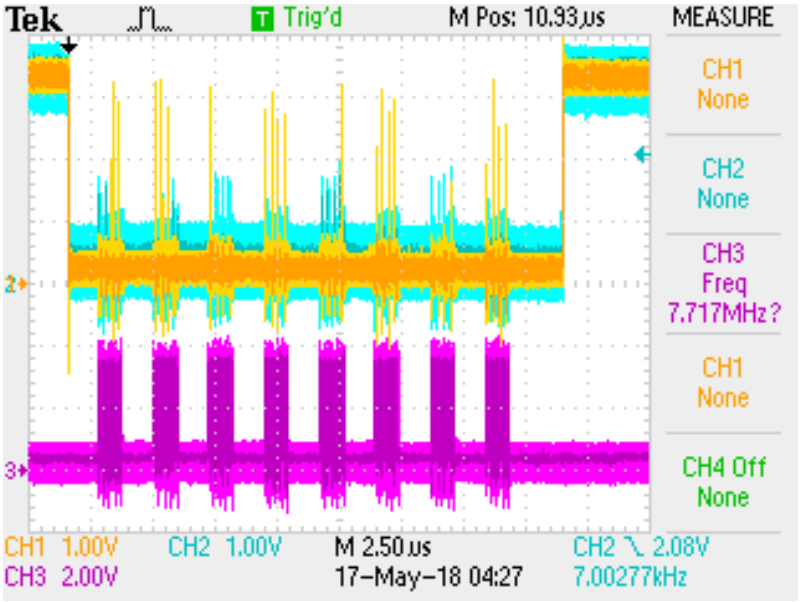


Figura 2.3 Crosstalk detectado en comunicación SPI.

- Ver el funcionamiento de las señales de comunicación SPI. En el siguiente ejemplo se observaron las líneas de comunicación entre el Arduino Zero y la tarjeta ZedBoard:

  En la Figura 2.4 se pueden ver las líneas *MOSI* y *SCLK*, el dispositivo maestro está enviando el código 0xAA para una lectura automática de las marcas de

tiempo. En la Figura 2.5 se pueden ver las líneas *MISO* y *SCLK*, el dispositivo esclavo está enviando el byte 0x5B que corresponde a la marca *UNIX_3*.

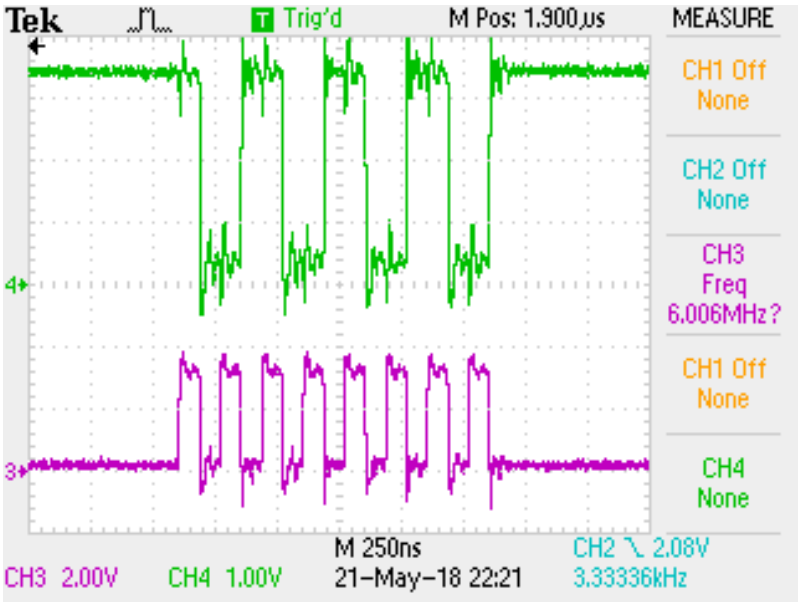


Figura 2.4 Envío de código 0xAA para lectura automática. *MOSI* en verde, *SCLK* en magenta.

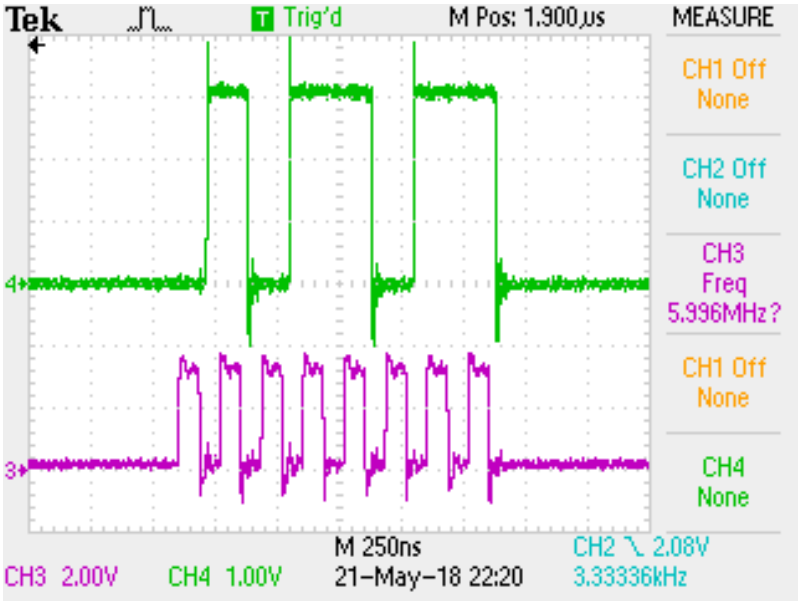


Figura 2.5 Recepción del byte 0x5B correspondiente al registro *UNIX_3*. *MISO* en verde, *SCLK* en magenta.

– Detectar el jitter entre 2 señales *CLK* de 60MHz durante las pruebas de validación, cada señal fue obtenida a partir del reloj de referencia de un receptor ThunderBolt E distinto. En la Figura 2.6 se utiliza la función "Persistence" del

osciloscopio y se puede ver el desfase de estas señales a lo largo del tiempo, prácticamente se ha formado una franja en el canal 2 lo que indica el desfase ocurrido con respecto a la señal del canal 1.

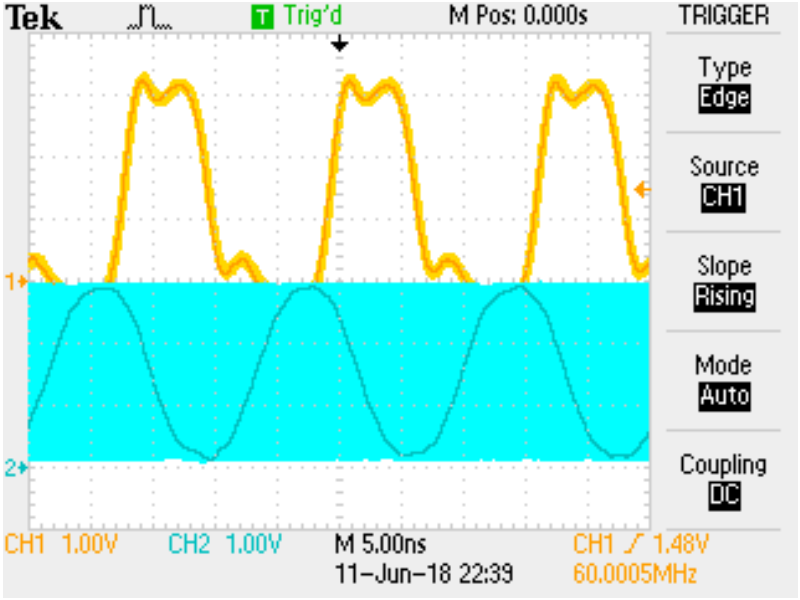


Figura 2.6 Jitter detectado entre 2 señales *CLK*. La franja color cyan representa el desplazamiento de la señal del canal 2 (cyan) con respecto a la señal del canal 1 (amarillo).

- **Generador de Señales:**

  Usado para generar, a partir de la señal de 10 MHz del receptor GPS Thunderbolt E, una señal sinusoidal de 60 MHz que utiliza el Controlador de Radar.

- **Controlador de Radar (CR):**

  Equipo desarrollado en el ROJ el cual genera las señales que operan el radar. Opera a partir de una señal de 60MHz, mayormente obtenida del reloj de 10MHz del receptor ThunderBolt E a través del generador de señales aunque también cuenta con oscilador interno, además puede activarse una opción de sincronismo externo a partir de la señal PPS del receptor GPS para reiniciar las cuentas internas de su sistema.

  A partir de la señal sinusoidal de 60MHz genera una señal de reloj aproximadamente cuadrada apta para lógica de 5V o 3.3V que se usa como señal CLK en los sistemas FPGA de adquisición y en el sistema GCTS.

De todas las señales que se pueden programar y usar del CR, en las pruebas del sistema GCTS se utilizó como entrada *SYNC* las señales TR (señal de conmutación entre modos de transmisión y recepción) y TXA (señal de pulso de transmisión).

En la Figura 2.7 se ve la señal TR, tiene una frecuencia de 8KHz y un nivel alto de voltaje de poco más de 4V, por lo que dicha señal pasó a través de un adaptador de nivel antes de llegar al GCTS.

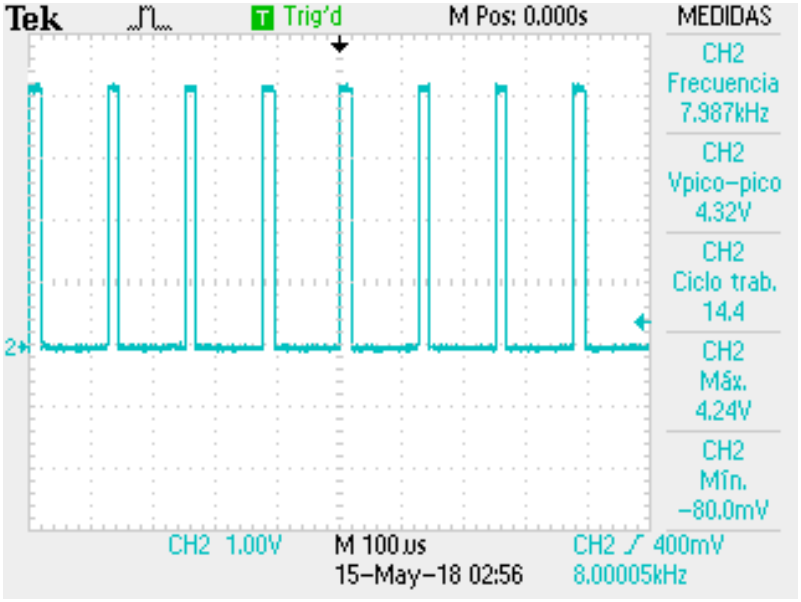


Figura 2.7 Ejemplo de pulso TR del Controlador de Radar.

- **Generador de Funciones:**

  Usado para generar señales de reloj aptas para los niveles lógicos de las tarjetas FPGA, a partir de la señal de 10 MHz del receptor GPS "Thunderbolt E".

- **Fuentes de voltaje:**

  Usadas para proveer energía a las tarjetas y módulos electrónicos y al receptor GPS.

- **Computadora:**

  Usada para recibir por un puerto USB 2.0 los datos extraídos por las tarjetas Teensy 3.2 y Arduino Zero y almacenarlos en archivos para su posterior análisis.

  También se utilizó para ejecutar las distintas aplicaciones con las que se desarrolló el sistema.

- **Lenguaje VHDL:**

VHDL-93. Utilizado para describir la arquitectura del circuito.

- **ISE Design Suite:**

  Versión 14.7. Software en el cual se desarrolló el sistema, utilizado para escribir el código para la programación del FPGA, sintetizar los bloques lógicos, diseñar la estructura del circuito y simular su comportamiento.

- **RS-232**

  Interfaz usada para comunicarse con el receptor GPS.

- **SPI**

  Interfaz usada para configurar el sistema a través de un dispositivo SPI maestro y poder extraer los datos.

  Seleccionada por permitir velocidades en el orden de los Mbps a cortas distancias y estar presente en la mayoría de microcontroladores de gama media o alta.

- **Protocolo de interfaz estándar de Trimble (TSIP)**

  Es un protocolo de interfaz diseñado para facilitar a desarrolladores gran flexibilidad al momento de interactuar con productos de Trimble.

  En [24] se detalla que el equipo ThunderBolt E tiene un puerto serial E/S de comunicación usado para el control y transmisión de datos usando la interfaz TSIP, la cuál esta basada en la transmisión de paquetes de información con la siguiente estructura:

  <DLE> <id> <data string bytes> <DLE> <ETX>

  donde:

  - <DLE> es el byte 0x10
  - <ETX> es el byte 0x03
  - <id> es un byte que identifca al paquete, puede tener cualquier valor excepto <DLE> y <ETX>

  Como los bytes en la cadena de datos (data string bytes) pueden tener cualquier valor, para evitar confusiones con las secuencias <DLE> <id> y <DLE> <ETX>, cada byte <DLE> debe ser precedido por un byte extra <DLE> (de relleno). Este byte extra debe ser añadido antes de enviar un paquete y removido cuando se recibe un paquete.

Una secuencia <DLE> <ETX> no significa necesariamente el final del paquete ya que pueden ser bytes en el medio de la cadena de datos. El final del paquete está establecido por <DLE> <ETX>, donde <ETX> está precedido por un número impar de bytes <DLE>.

- **Putty:**

  Consola de comunicación serial, en la cual se verificó si los datos se transmitían y recibían de forma correcta desde el FPGA y desde el receptor GPS.

- **Lenguaje C++:**

  Utilizado para describir el comportamiento de un dispositivo SPI maestro en un microprocesador que se encargó de controlar al sistema GCTS.

- **Arduino IDE y Energia:**

  Programas usados para programar las tarjetas Arduino, Teensy y TIVA en lenguaje C++.

- **Minicom:**

  Aplicación usada para leer el tráfico a través de los puertos USB en una computadora con sistema operativo Linux. Usado para leer y almacenar los datos proporcionados por el dispositivo maestro SPI que obtenía las marcas de tiempo a partir del GCTS.

- **Python y MATLAB:**

  Lenguaje de programación y aplicación de procesamiento usados para analizar y hacer seguimiento de las señales y datos generados por el sistema. Útiles para detectar errores y patrones en grandes cantidades de datos.

3 Diseño del Sistema

El sistema GCTS desarrollado es un **módulo hardware** capaz de extraer información de tiempo de un receptor GPS Thunderbolt E a través de una interfaz RS-232 y formar un reloj UTC. La precisión del reloj está basada en la señal sinusoidal de referencia de 10MHz del receptor GPS, si esta señal pasa previamente por un multiplicador de frecuencia se puede aumentar la precisión (el CR la multiplica a 60MHz). El sistema es configurable a través de un bus SPI esclavo que además de permitir un reinicio del sistema y elegir su comportamiento frente a segundos intercalares, da la opción de leer registros que indican el modo de configuración del receptor GPS, la condición de operación del reloj, el estado del sistema y la transición de un segundo intercalar. Se puede configurar en dos modos: *request* y *auto*, el primero escribe y lee registros usando direcciones y el segundo puede ser usado para obtener consecutivamente las marcas de tiempo mediante un código.

Cuando el sistema detecta un flanco de subida en la señal de excitación, le asigna una marca de tiempo de 8 bytes dividida en 3 secciones:

- *UNIX*: (4 bytes) representación POSIX del tiempo (número de segundos),
- *MS*: (2 bytes) número de milisegundos,
- *US*: (2 bytes) número de microsegundos,

la cual puede ser extraída en cualquier momento antes de la llegada de un nuevo flanco de subida en la señal de excitación.

El sistema se diseñó en VHDL usando una metodología estructural formada por varios componentes, por lo que también puede ser usado como parte de un sistema de adquisición digital más complejo.

La Figura 3.1 muestra los componentes del sistema y la comunicación entre ellos. A continuación se ofrece una breve descripción del funcionamiento de los componentes:

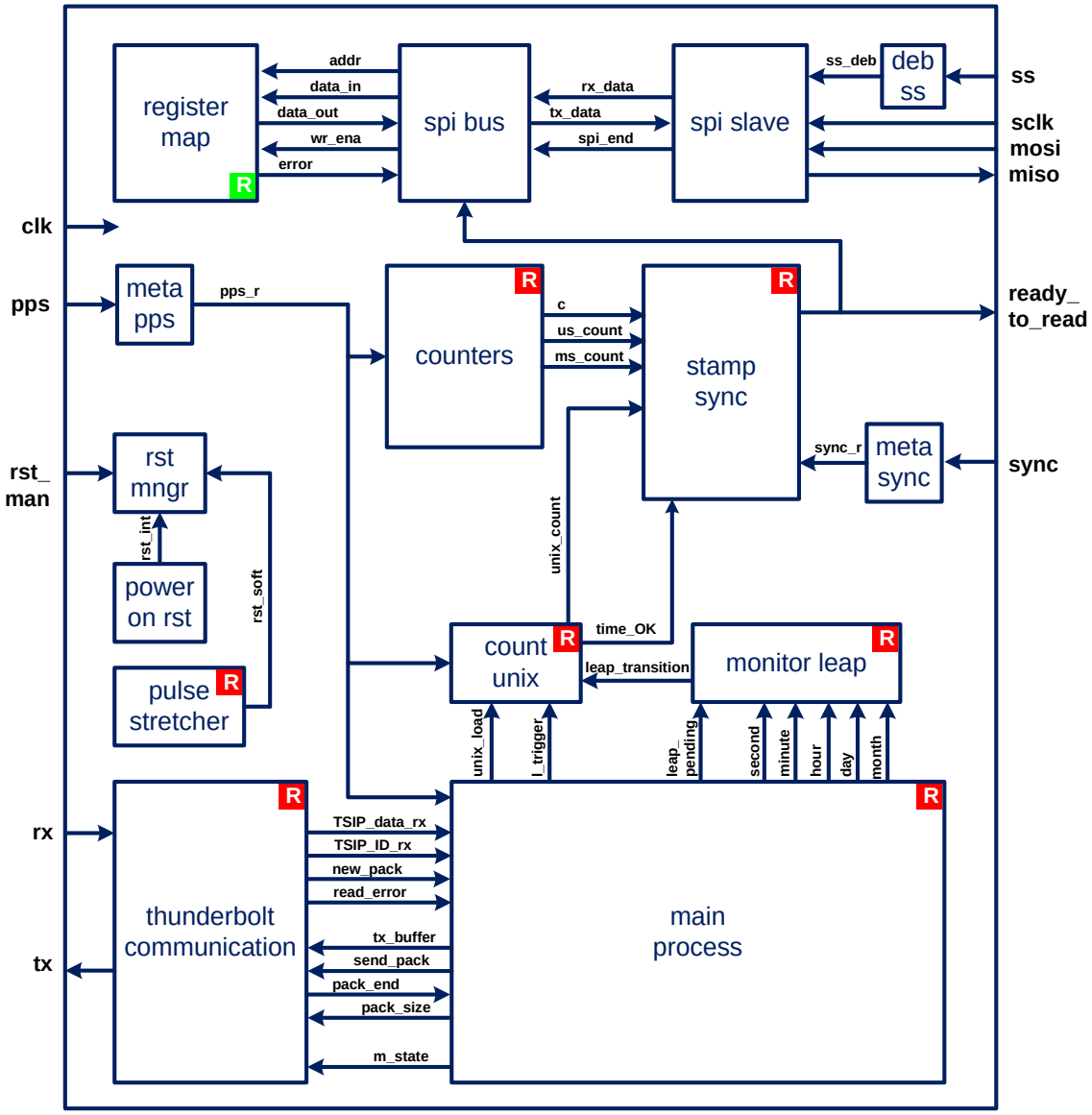


Figura 3.1 Diagrama general del sistema y sus componentes.

Los procesos ejecutados dentro de los componentes trabajan a partir de cada flanco de subida de la señal *clk* (el reloj del sistema) y la mayoría pueden ser reiniciados con la señal *rst_man* llevando los valores de sus señales internas y externas a 0, salvo que se indique lo contrario.

La entrada *pps* pasa al dominio de reloj del sistema a través del componente **meta pps**, el cual entrega una señal *pps_r* para los componentes **main process**, **count unix** y **counters**, los cuales la utilizan respectivamente para:

- disparar los procesos principales de operación,
- incrementar cuenta de la señal *unix_count* y
- reiniciar las cuentas de las señales *c*, *us_count* y *ms_count*.

El componente **main process** maneja a los componentes **thunderbolt communication**, **count unix** y **monitor leap**, respectivamente, de la siguiente manera:

- Espera un cambio de estado en la señal *pps* para configurar que mensajes de difusión transmitirá el receptor GPS, luego espera nuevamente el cambio de estado para configurar las opciones de formato de los datos que recibirá, para finalmente esperar por los mensajes y actualizar la información de tiempo del GCTS.
- Carga el componente contador **count unix** con la señal **unix load** mediante el disparo de la señal *l_trigger*.
- Le indica el tiempo UTC actual a través de 5 señales y le avisa cuando hay un segundo intercalar pendiente (*leap_pending*).

El componente **thunderbolt communication** es el encargado de comunicarse directamente con el receptor GPS a través de la interfaz RS-232 mediante las señales *rx* y *tx*. Tiene dos procesos: el de recepción y el de transmisión, los cuales interactúan con el componente **main process** por medio de 9 señales. Este componente utiliza el protocolo TSIP.

El componente **monitor leap** controla la señal *leap_transition* la que a su vez indica al componente **coun unix** cuando ocurre un segundo intercalar, este último componente maneja la señal *time_OK* que avisa cuando el tiempo ha sido actualizado a un valor válido después de cada carga.

Mientras la señal *time_OK* lo indique, el componente **stamp sync** registra las señales de los contadores (*c*, *us_count*, *ms_count* y *unix_count*) al detectar un cambio de estado de 0 a 1 en la señal *sync_r*, la cual sale del componente **meta sync** que a su vez recibe la entrada *sync* y la adecua al dominio de reloj del sistema. Cuando los registros están listos (aproximadamente dos ciclos de reloj después de un flanco de subida en la entrada *sync*) se activa la señal *ready_to_read*, que se ofrece para ser usada como señal de interrupción en un dispositivo maestro SPI, el cual podrá iniciar una transferencia para extraer las marcas de tiempo respectivas.

El bloque **counters** tiene 3 componentes, los cuales llevan cuentas del número de pulsos de reloj (*c*), los microsegundos (*us_count*) y los milisegundos (*ms_count*) transcurridos. A parte del reinicio normal, la señal *pps_r* también devuelve a cero los valores de las cuentas.

En la Figura 3.1, también se observa que varios componentes tienen un cuadrado rojo con una R en su esquina superior derecha, esto denota que dicho componente se comunica con una memoria almacenada en el componente **register map** (marcado con un cuadrado verde en su esquina inferior derecha), dicha memoria contiene 16 direcciones de un byte cada una, esta data guarda información del sistema y configuración de funcionamiento. La data se puede escribir o leer mediante 5 señales de operación.

El componente **spi bus** se encarga de realizar la lectura y escritura de los registros en **register map** a partir de los datos del componente **spi slave**. Se ejecuta en los dos modos definidos:

- *auto*: este modo actualiza la data a leer (*tx_data*) en base a una nueva marca de tiempo (8 bytes), desplazándose byte por byte en cada activación de la señal *spi_end*. La señal *ready_to_read* indica cuando debe posicionarse el desplazamiento nuevamente al primer byte de la marca de tiempo. Se usa para leer automáticamente las marcas de tiempo.
- *request*: en este modo se pueden leer y escribir los registros en cualquier momento. Se usa para configurar, reiniciar el sistema y leer registros con información específica.

Las 4 líneas SPI se comunican con el componente **spi slave**, el cual como su nombre lo indica implementa una módulo SPI esclavo que ofrece al sistema una interfaz

lógica sencilla de solo 3 señales: *rx_data*, *tx_data* y *spi_end*, a través de estas señales ofrece la data recibida, capta la data a transmitir, y avisa cuando se completo una transferencia SPI. Ante el cross-talk generado en las conexiones con el dispositivo maestro SPI se decidió añadir un filtro antirebote en la línea *ss*, implementado en el componente **deb ss** que ofrece una salida filtrada *ss_deb*.

Finalmente, el reinicio del sistema fue manejado en 3 maneras distintas de la siguiente manera:

- Reinicio inicial: mediante el componente **power on rst** y la señal *rst_init*.
- Reinicio manual: a través de la entrada **rst_man**.
- Reinicio por software: mediante la escritura de un registro se genera un pulso con duración de un ciclo de reloj el cual va hacia el componente *pulse_stretcher* para generar una señal de reinicio mas larga (**rst_soft**).

Estas 3 señales van al componente **rst mngr**, el cual es una compuerta lógica de 3 entradas que puede ser OR o AND dependiendo del nivel de reinicio configurado, este componente ofrece un reinicio síncrono en la activación de cualquiera de sus 3 entradas

3.1 Componentes del Sistema

A continuación se presenta una descripción detallada de cada componente del sistema, la mayoría de secciones esta estructurada de la siguiente manera:

- Un bloque donde se puede apreciar las entradas del componente a la izquierda y sus salidas a la derecha.
- Una tabla donde se describen las señales de entrada y salida.
- Una tabla que contiene los parámetros genericos de configuración del componente.
- Un diagrama de funcionamiento o una máquina de estados.
- Un gráfica del comportamiento de las señales en simulación.

El sistema utiliza una librería propia creada para proveer dos tipos de señales a los componentes *main_logic* y *thunderbolt_communication*:

- *byte_vector_std*: utilizada para para trabajar con arrays de datos, usada en las señales que se encargan de la lectura y envío de paquetes TSIP. Consiste en un array de rango natural en la que cada elemento es un byte (*std_logic_vector*).
- *m_state*: utilizada para comunicar el estado en que se encuentra el componente *main_logic* mediante un vector de 5 bits que va al componente *thunderbolt_communication*.

3.1.1 Componente de sincronización de señales externas

El componente *metastable_input* (Figura 3.2, Tabla 3.1) se diseñó para pasar una señal externa al dominio del reloj con el que funciona el sistema. Este componente soluciona el problema de metaestabilidad haciendo pasar la señal de entrada *input* a través de un circuito que consiste en dos Flip-Flop tipo D ofreciendo una salida *output* que tiene el registro de los dos últimos estados de la señal *input* captados en el flanco de subida del reloj *clk*, ver Figura 3.3.

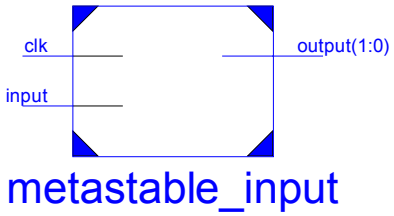


Figura 3.2 Componente *metastable_input*.

Tabla 3.1 Puertos de componente *metastable_signal*.

Puertos	Tipo	Modo	Descripción
clk	*std_logic*	in	Reloj
input	*std_logic*	in	Señal externa
output	*std_logic_vector*	out	Señal estable

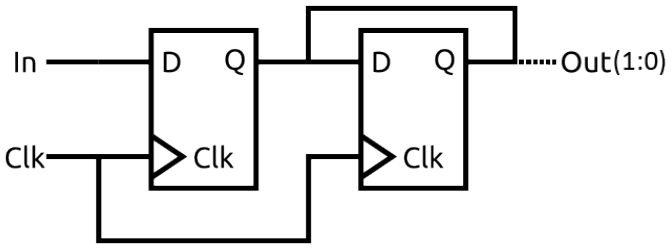


Figura 3.3 Diagrama de componente *metastable_input*.

Su funcionamiento se puede apreciar en la Figura 3.4, donde la señal *output* se actualiza durante los flancos de subida de *clk*, pudiendo ser leída al siguiente pulso de reloj de *clk*.

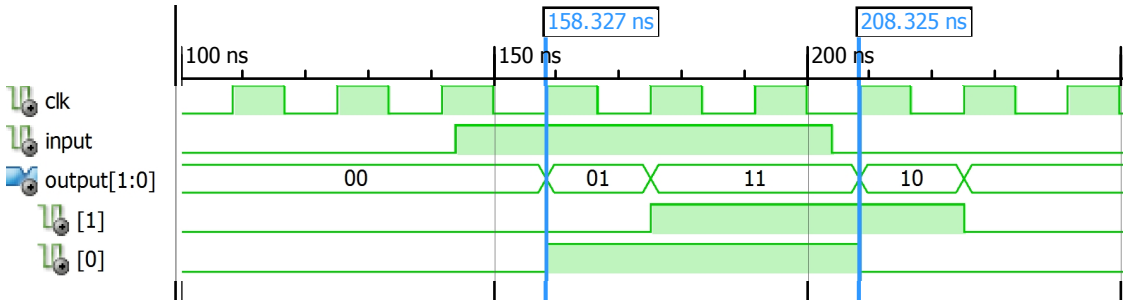


Figura 3.4 Testbench de componente *metastable_signal*.

Éste es el único bloque que se utiliza dos veces dentro del sistema, y toma las señales *pps* y *sync*, ver Figura 3.5.

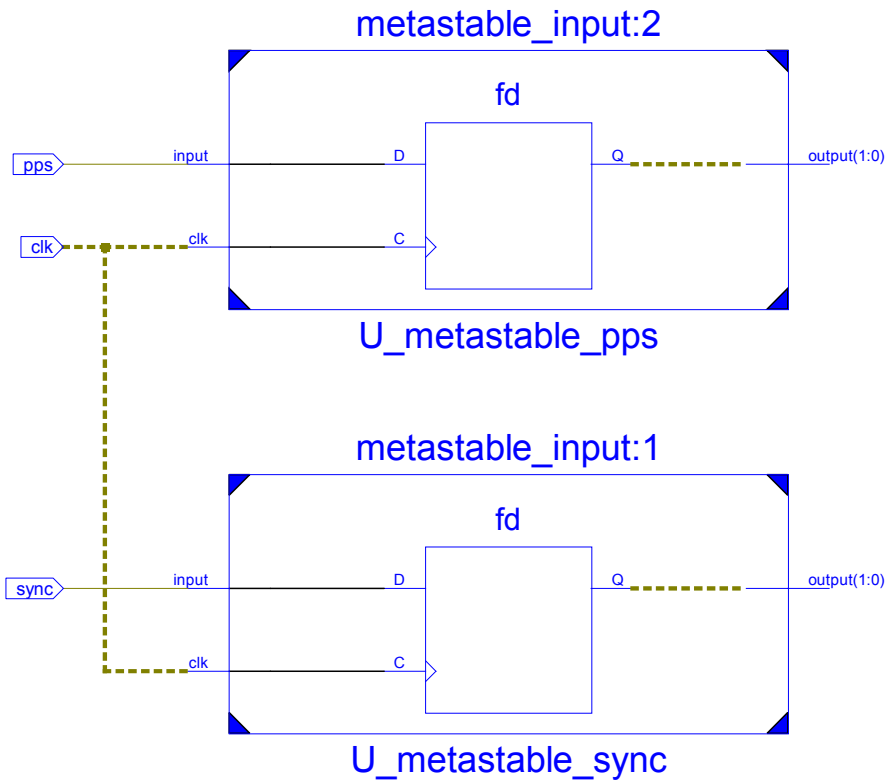


Figura 3.5 Circuito de bloques para sincronización de señales externas.

3.1.2 Componente de comunicación serial UART

El componente *UART* (Figura 3.6) fue implementado para establecer un bloque de comunicación serial, para esto se tomó como base un módulo hardware de *OpenCores.com* el cual consiste en dos máquinas de estado Mealy del mismo tipo usadas para la recepción y para la transmisión de datos.

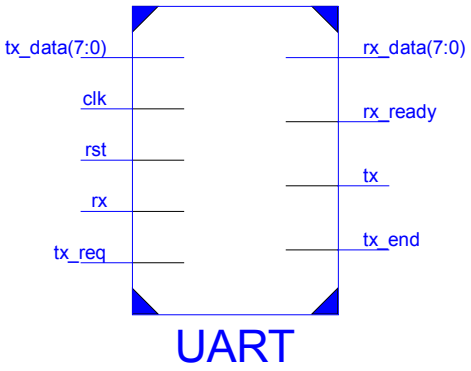


Figura 3.6 Componente *UART*.

Este componente toma los parámetros de configuración de la Tabla 3.2 definidos con la declaración *GENERIC* y cuenta con las entradas y salidas descritas en la Tabla 3.3.

Tabla 3.2 Parámetros de componente *UART*.

Parametros	Tipo	Descripción
CLK_FREQ	*integer*	Frecuencia del reloj en MHz
RST_LVL	*std_logic*	Nivel lógico de señal de reinicio
BAUDS	*integer*	Tasa de baudios
PARITY_EN	*std_logic*	Activación de bit de paridad
N_STOP	*std_logic*	Número de bits de parada

Tabla 3.3 Puertos de componente *UART*.

Puertos	Tipo	Modo	Descripción
clk	*std_logic*	in	Reloj
rst	*std_logic*	in	Reinicio
rx	*std_logic*	in	Pin RX de interfaz
tx	*std_logic*	out	Pin TX de interfaz
tx_req	*std_logic*	in	Solicitud de transmisión
tx_end	*std_logic*	out	Finalización de transmisión
tx_data	*std_logic_vector*	in	Datos a transmitir
rx_ready	*std_logic*	out	Datos recibidos listos
rx_data	*std_logic_vector*	out	Datos recibidos

PARITY_EN igual a 0 indica que no hay bit de paridad, si es igual a 1 habrá bit de paridad. *N_STOP* igual a 0 indica que habrá un bit de parada, caso contrario habrán 2.

El componente puede ser analizado en dos partes: recepción y transmisión. La recepción cuenta con 4 procesos: *rx_debouncer*, *rx_start_detect*, *rx_clk_generator* y *rx_process*, mientras que la transmisión solo con 2: *tx_clk_generator* y *tx_process*. (Figura 3.7).

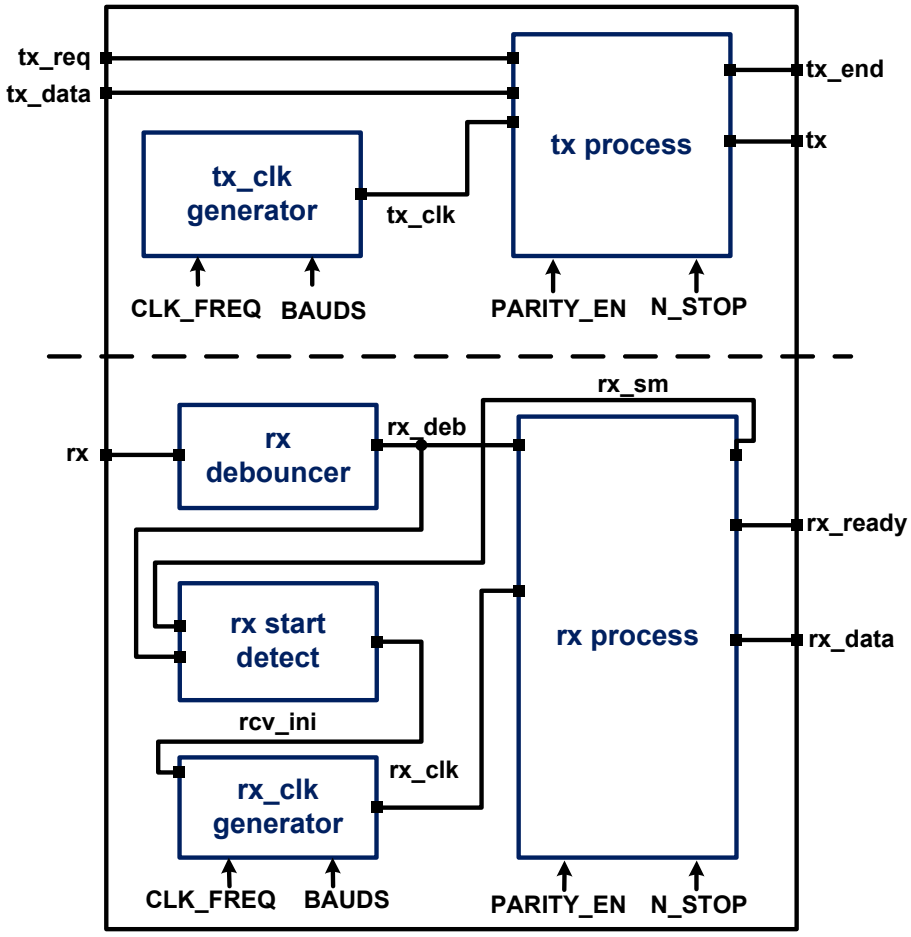


Figura 3.7 Diagrama de componente *UART*.

La lógica de la parte de transmisión del componente se basa en la máquina de estados *tx_fsm* (ver Figura 3.8) dentro del proceso *tx_process*, que se actualiza cada vez que el reloj *tx_clk* generado dentro de *tx_clk_generator* es 1, debido a esto el estado en la línea *tx* está retrasado un periodo respecto a *tx_fsm*.

En la figura 3.9a se aprecia que *tx_fsm* cambia de *idle* a *start* cuando se capta un 1 en *tx_req*, en este momento se carga el valor de *tx_data*, en este caso 0b10101100, y se empieza a transmitir a través de *tx* el bit de inicio (nivel bajo), luego se continua con los 8 bits

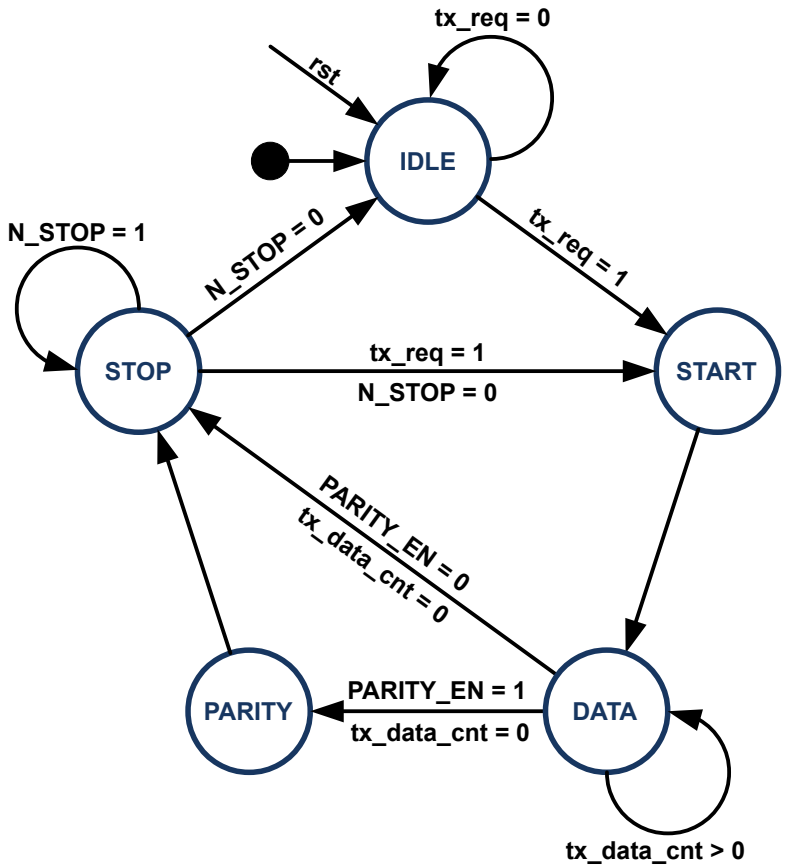
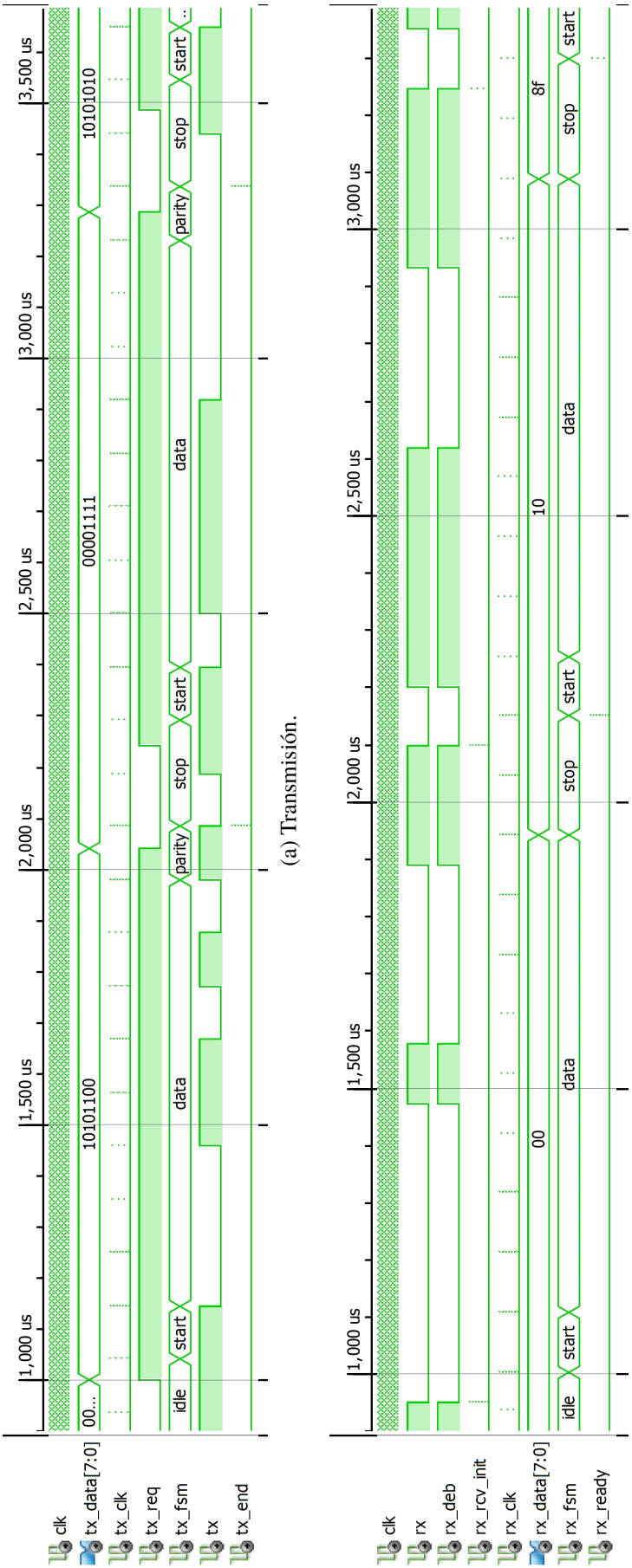


Figura 3.8 Máquina de estado de *tx_process*.

de datos, empezando por el bit de menor peso (LSB), en base al contador decreciente *tx_data_cnt*, dependiendo de la configuración se envía el bit de paridad calculado y finalmente los bits de parada. De encontrar a *tx_req* activo al finalizar la transmisión se pasará a cargar el nuevo byte a transmitir, caso contrario *tx_fsm* pasará al estado de reposo (*idle*).

Al inicio del envio del último bit de datos o el bit de paridad (si es que está configurado) se activará la señal *tx_end* para indicar que la transmisión ha finalizado y de ser necesario poder cargar un nuevo byte y solicitar una nueva transmisión con *tx_req*. El proceso recién enviará el nuevo byte cuando finalice la transmisión previa.

El comportamiento de la parte de recepción depende de 4 procesos (Figura 3.7), el filtro antirebote del proceso *rx_debouncer* evita falsos cambios de estado en la señal de entrada externa *rx* ofreciendo la señal filtrada *rx_deb*, el proceso *rx_start_detect* detecta un flanco de bajada en *rx_deb* y si la máquina de estado *rx_fsm* se encuentra en *idle* o *stop* activa la señal interna *rx_rcv_init* que resetea a la mitad el contador que maneja la señal *rx_clk* dentro del proceso *rx_clk_generator* asegurándose de que se adquiera el estado



(a) Transmisión.

(b) Recepción.

Figura 3.9 Testbench de componente *UART*.

lógico de *rx_deb* a la mitad del periodo del bit, es por esto que *rx_fsm* está retrasado medio periodo con respecto a la línea *rx_deb*.

Cuando *rx_clk* es 1, cada vez que la cuenta llega a su máximo valor, se produce un cambio en el estado *rx_fsm* y se lee el valor de la línea *rx_deb*, así se van detectando los bits de inicio, datos y parada de acuerdo a la máquina de estado *rx_fsm*. Al recibir el último bit de parada se activará la señal *rx_ready* para indicarle al usuario que los datos almacenados en *rx_data* están listos para leerse. De detectarse una transición de alto a bajo cuando *rx_fsm* está en *stop* significa que una nueva transferencia está ocurriendo y *rx_fsm* pasa a *start*, caso contrario *rx_fsm* pasa al estado de reposo (*idle*).

En el ejemplo de la Figura 3.9b se ve que después de la transición de alto a bajo en la línea *rx_deb* la señal *rx_clk* ha detectado la siguiente cadena de valores 0b00000100011 que se traducen en "bit inicio, bit0, bit1, bit2, ... bit7, bit stop1, bit stop2" dando como resultado *rx_data* igual a 0x10, mientras que en la siguiente transacción se recibe 0x8F.

3.1.3 Componente de comunicación con receptor GPS Timble

El componente *thunderbolt_communication* (Figura 3.10) fue implementado para contener y controlar el componente *UART* descrito en la sección 3.1.2 y establecer la comunicación con el receptor GPS "Thunderbolt E" de Trimble a través del protocolo TSIP descrito en la sección 2.2.

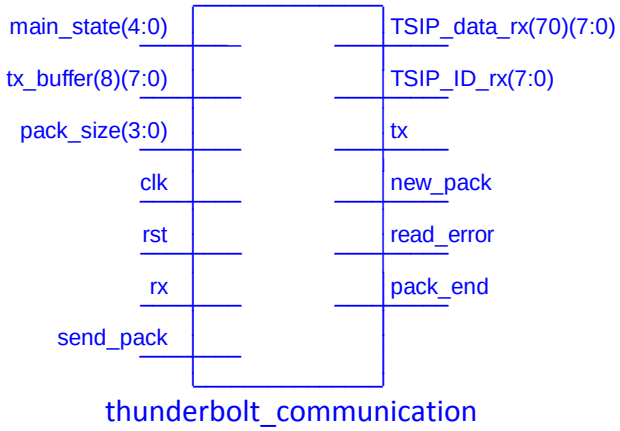


Figura 3.10 Componente *thunderbolt_communication*.

El componente toma los parámetros de configuración de la Tabla 3.4 definidos con la declaración *GENERIC* que también incluyen 3 parámetros para configurar el componente *UART*, además cuenta con las entradas y salidas descritas en la Tabla 3.5.

Tabla 3.4 Parámetros de componente *thunderbolt_communication*.

Parámetros	Tipo	Descripción
CLK_FREQ	*integer*	Frecuencia del reloj en MHz
RST_LVL	*std_logic*	Nivel lógico de señal de reinicio
TX_BUFF_SIZE	*integer*	Tamaño de buffer de transmisión
RX_BUFF_SIZE	*integer*	Tamaño de buffer de recepción
BAUDS	*integer*	Tasa de baudios
PARITY_EN	*std_logic*	Activación de bit de paridad
N_STOP	*std_logic*	Núммero de bits de parada

Tabla 3.5 Puertos de componente *thunderbolt_communication*.

Puertos	Tipo	Modo	Descripción
CONTROL			
clk	*std_logic*	in	Reloj
rst	*std_logic*	in	Reinicio
LÍNEAS DE COMUNICACIÓN			
rx	*std_logic*	in	RX de compoente UART
tx	*std_logic*	out	TX de componente UART
ESTADO DE LÓGICA PRINCIPAL			
main_state	*m_state*	in	Estado del sistema GCTS
PROCESO *read_TSIP_package*			
TSIP_data_rx	*byte_vector_std*	out	Data del paquete TSIP extraído
TSIP_ID_rx	*std_logic_vector*	out	Byte de identificación de paquete
new_pack	*std_logic*	out	Indica recepción de nuevo paquete
read_error	*std_logic*	out	Bandera de error en lectura
PROCESO *send_TSIP_package*			
tx_buffer	*byte_vector_std*	in	Almacena paquete TSIP a enviar
send_pack	*std_logic*	in	Solicita envío de paquete
pack_end	*std_logic*	out	Transmisión de paquete finalizada
pack_size	*unsigned*	in	Tamaño del paquete a enviar

Este componente interactúa con el bloque *main_logic* para enviar y recibir basándose en los procesos *read_TSIP_package* y *send_TSIP_package*.

El proceso de recepción (*read_TSIP_package*) evalúa si la señal *main_state* corresponde a *wait_for_string* y si no hubo un error en la lectura del primer byte recibido (debe ser 0x10), de cumplir con estas condiciones espera a cada activación de la señal *rx_ready* para asignar el byte de *rx_data* a un buffer interno. En este proceso es donde se decodifica el paquete TSIP extrayendo la data válida, cuando se termina de leer un paquete se activa la señal *new_pack*, y se cargan las señales del buffer interno a las señales *TSIP_ID_rx* (identificador de paquete)y *TSIP_data_rx* (data de paquete).

El proceso de transmisión (*send_TSIP_package*) empieza a enviar los bytes de la señal *tx_buffer* cuando se activa la señal *send_pack* y al terminar la transmisión de n-bytes (tamaño del paquete determinado por *pack_size*) activa la señal *pack_end* para indicar el final de la transmisión al componente *main_logic*.

3.1.4 Componentes contadores

En el sistema se utilizan 3 contadores basados en la señal *clk*: *count_pulses*, *count_microseconds* y *count_milliseconds*, los cuales tienen dos parámetros de configuración como se puede ver en la Tabla 3.6.

Tabla 3.6 Parámetros de componentes contadores..

Parámetros	Tipo	Descripción
CLK_FREQ	*integer*	Frecuencia del reloj en MHz
BIN_COUNT	*integer*	Bit más significativo para cuenta (0 a CLK_FREQ)

El componente *count_pulses* (Figura 3.11, Tabla 3.7) consiste en un contador simple *c* de rango 0 a *CLK_FREQ*-1 que se incrementa a cada flanco de subida de *clk* y se reinicia cuando llega a *CLK_FREQ*-1 o cuando se detecta una transición de 0 a 1 en la señal *pps_r*. Este componente es el límite de precisión que se puede alcanzar y se puede acceder a su cuenta en el momento de cronometraje de la señal *sync* solicitando el registro *PULSES*. En la implementación final del sistema *CLK_FREQ* es 60, pues a esa frecuencia opera el Controlador de Radar.

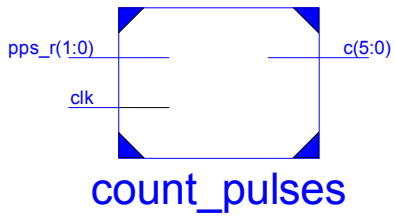


Figura 3.11 Componente *count_pulses*.

Tabla 3.7 Puertos de componente *count_pulses*.

Puertos	Tipo	Modo	Descripción
clk	std_logic	in	Reloj
pps_r	std_logic_vector	in	Registro de señal pps
c	unsigned	out	Contador de pulsos por microsegundo

El componente *count_microseconds* (Figura 3.12, Tabla 3.8) está formado por un

contador *us_count* de rango 0 a 999, el cual se incrementa hasta llegar a 999 cuando el flanco de subida de *clk* coincide con *c* = *CLK_FREQ*-1 y solo se reiniciará cuando detecte una transición de 0 a 1 en la señal *pps_r*.

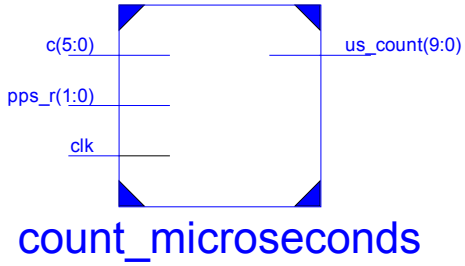


Figura 3.12 Componente *count_microseconds*.

Tabla 3.8 Puertos de componente *count_microseconds*.

Puertos	Tipo	Modo	Descripción
clk	std_logic	in	Reloj
pps_r	std_logic_vector	in	Registro de señal pps
c	unsigned	in	Contador de pulsos por microsegundo
us_count	unsigned	out	Contador de microsegundos

El componente *count_milliseconds* (Figura 3.13, Tabla 3.9) formado por un contador *ms_count* de rango 0 a 999, el cual se incrementa hasta llegar a 999 cuando el flanco de subida de *clk* coincide con *c* = *CLK_FREQ*-1 y con *us_count* = 999 y solo se reiniciará cuando detecte una transición de 0 a 1 en la señal *pps_r*.

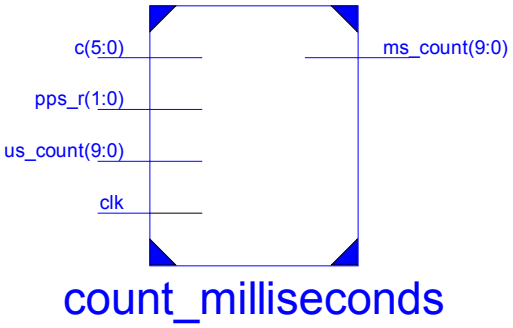


Figura 3.13 Componente *count_milliseconds*.

Un diagrama completo de las conexiones entre los 3 contadores se puede apreciar en la Figura 3.14.

Tabla 3.9 Puertos de componente *count_milliseconds*.

Puertos	Tipo	Modo	Descripción
clk	std_logic	in	Reloj
pps_r	std_logic_vector	in	Registro de señal pps
c	unsigned	in	Contador de pulsos por microsegundo
us_count	unsigned	in	Contador de microsegundos
ms_count	unsigned	out	Contador de milisegundos

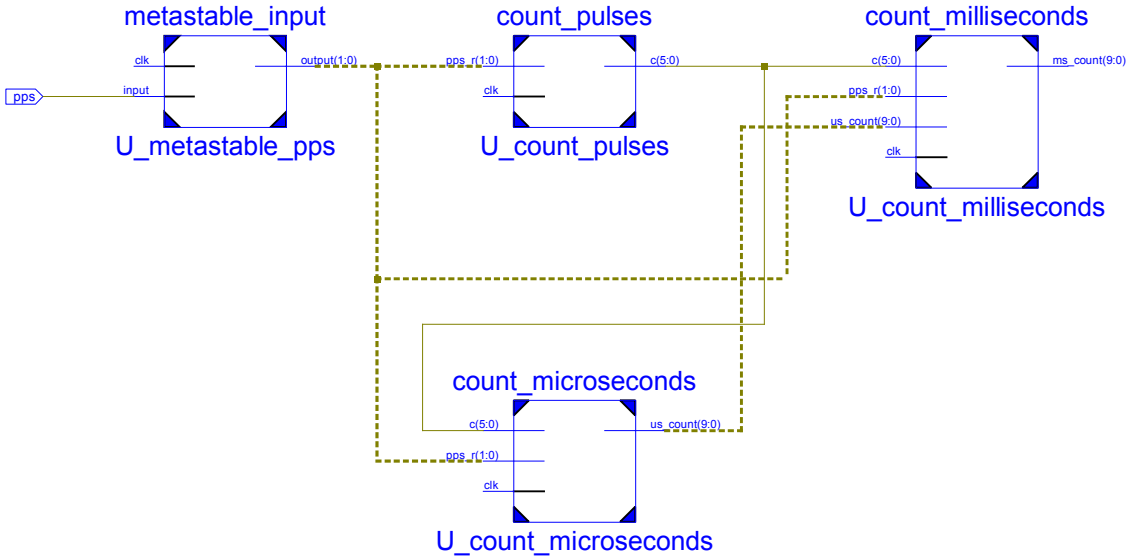


Figura 3.14 Circuito de contadores.

3.1.5 Componente contador de segundos

El componente *coun_unix* (Figura 3.15, Tabla 3.10) está formado por un contador creciente de 32 bits basado en la representación de las marcas de tiempo POSIX. Tiene opción de reseteo tiene opción de carga, pausa y salto.

Tabla 3.10 Puertos de componente *count_unix*.

Puertos	Tipo	Modo	Descripción
clk	*std_logic*	in	Reloj
rst	*std_logic*	in	Reinicio
pps_r	*std_logic_vetor*	in	Registro de señal pps
unix_load	*unsigned*	in	Carga para contador
l_trigger	*std_logic*	in	Bandera para cargar
leap_sign	*std_logic*	in	Signo de segundo intercalar
leap_transition	*std_logic*	in	Indicador de segundo intercalar
unix_count	*unsigned*	out	Contador de segundos
time_OK	*std_logic*	out	Bandera que indica tiempo válido

El contador empieza desde 0 y consigue el valor real de tiempo cuando *l_trigger*

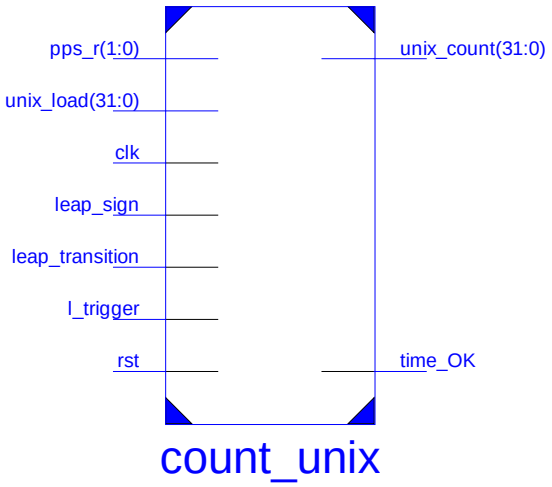


Figura 3.15 Componente *count_unix*.

se activa y carga al contador con *unix_load*, una vez ocurrido esto la bandera *time_OK* pasa a estado alto. A partir de este momento el contador aumentará en 1 cada vez que detecte una transición de 0 a 1 en la señal *pps_r* salvo que la señal *leap_transition* se active, ya que dependiendo del signo del segundo intercalar a considerar (*leap_sign*) la cuenta parará por un segundo o aumentará en 2.

Normalmente el componente *main_logic* proporcionará los datos de tiempo después de cada pulso PPS gracias al paquete 0x8F-AB, debido a esto el contador se mantendrá siempre actualizado, pero como esto pasa después del pulso PPS la actualización no cambiará la cuenta. En el único caso que se podrá notar la actualización será si se configura el registro *LEAP CONFIG* para que no tome en cuenta el segundo intercalar, de manera que al llegar el PPS correspondiente al segundo la cuenta se incrementará en 1 y recién cuando llegue la información del paquete se actualizará al valor real.

3.1.6 Componente de lógica principal

El componente *main_logic* (Figura 3.16) toma los parámetros de configuración de la Tabla 3.11 definidos con la declaración *GENERIC* y cuenta con las entradas y salidas descritas en la Tabla 3.12.

Tabla 3.11 Parámetros de componente *main_logic*.

Parámetros	Tipo	Descripción
RST_LVL	*std_logic*	Nivel de reset
TX_BUUF_SIZE	*integer*	Tamaño de buffer de transmisión
RX_BUFF_SIZE	*integer*	Tamaño de buffer de recepción

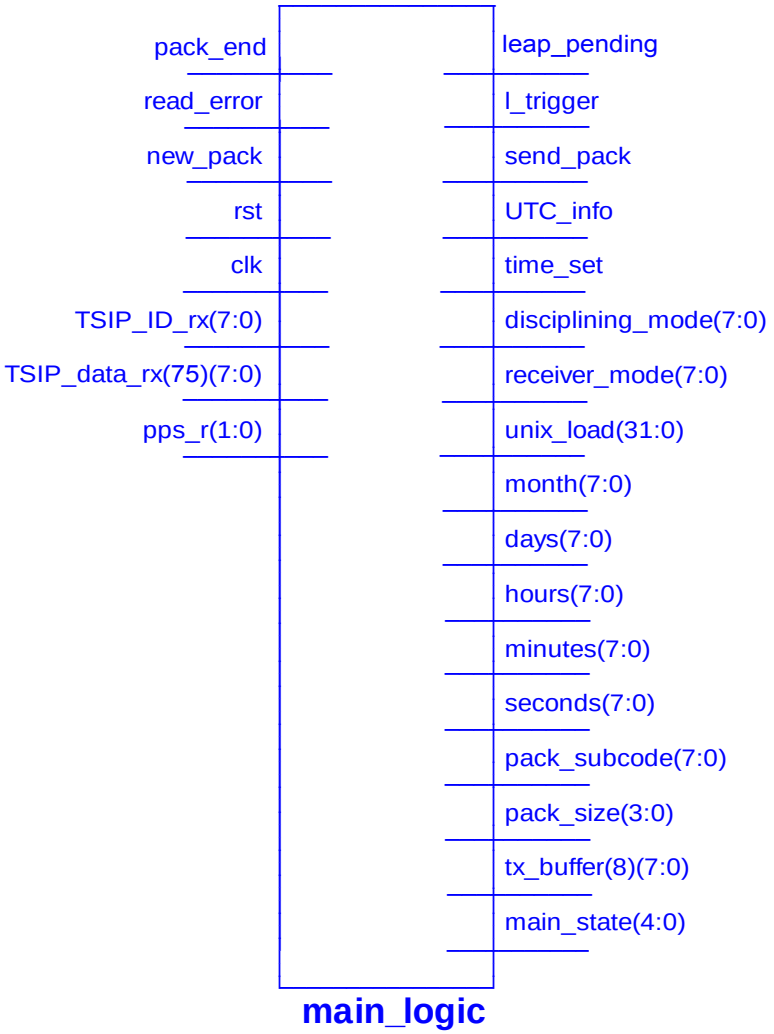


Figura 3.16 Componente *main_logic*.

La función principal del componente consiste en configurar el receptor GPS para que envíe los paquetes con información de tiempo (0x8F-AB y 0x8F-AC), y luego extraer la data de estos paquetes para actualizar el contador de segundos. La data puede demorar en ofrecer la información de tiempo UTC hasta 12.5 minutos ([24]) pues es la frecuencia en que la red de satélites GPS ofrece dicha información.

El proceso de la lógica principal consiste en una máquina de estado de 5 estados: *wait_for_pps*, *set_broadcast*, *set_UTC_TOD*, *wait_for_string* y *upload_time* (ver un diagrama simplificado en Figura 3.17) con los que se manejan señales de control para los componentes *thunderbolt_communication*, *registers_map*, *monitor_leap* y *count_unix*. A continucación se explicará el comportamiento de cada estado:

- *wait_for_pps*: Estado inicial que espera una cadena de 0b01 (transición bajo a alto) en

Tabla 3.12 Puertos de componente *main_logic*.

Puertos	Tipo	Modo	Descripción
CONTROL			
clk	*std_logic*	in	Reloj
rst	*std_logic*	in	Reinicio
REPORTE			
time_set	*std_logic*	out	Bandera: información tiempo GPS
UTC_info	*std_logic*	out	Bandera: información UTC
PPS			
pps_r	*std_logic_vector*	in	Registro señal pps
ESTADO			
main_state	*m_state*	out	Estado del sistema GTS
RECEPCIÓN			
TSIP_data_rx	*byte_vector_std*	in	Data de paquete recibido
TSIP_ID_rx	*std_logic_vector*	in	Identificación de paquete recibido
new_pack	*std_logic*	in	Nuevo paquete TSIP recibido
read_error	*std_logic*	in	Error en lectura
TRANSMISIÓN			
tx_buffer	*byte_vector_std*	out	Buffer de transmisión
send_pack	*std_logic*	out	Solicitud de envío
pack_end	*std_logic*	in	Fin de la transmisión del paquete
pack_size	*unsigned*	out	Tamaño de paquete a transmitir
FECHA			
pack_subcode	*std_logic_vector*	out	Sub-código de paquetes
seconds	*std_logic_vector*	out	Segundos
minutes	*std_logic_vector*	out	Minutos
hours	*std_logic_vector*	out	Horas
days	*std_logic_vector*	out	Días
month	*std_logic_vector*	out	Meses
TIEMPO POSIX			
unix_load	*unsigned*	out	Tiempo POSIX actualizado
l_trigger	*std_logic*	out	Bandera que indica actualización
leap_pending	*std_logic*	out	Bandera seg. intercalar pendiente
INFORMACIÓN DE RECEPTOR GPS			
receiver_mode	*std_logic_vector*	out	Modo de configuración
disciplining_mode	*std_logic_vector*	out	Condición de operación de reloj

la entrada *pps_r* para cambiar de estado dependiendo de una señal auxiliar *aux_state* que esta inicializada como 0, en ese caso cambia *aux_state* a 1 y pasa a al estado *set_broadcast* al regresar de ese estado y encontrar *aux_state* igual a 1, a la siguiente cadena 0b01 pasará al estado *set_UTC_TOD*

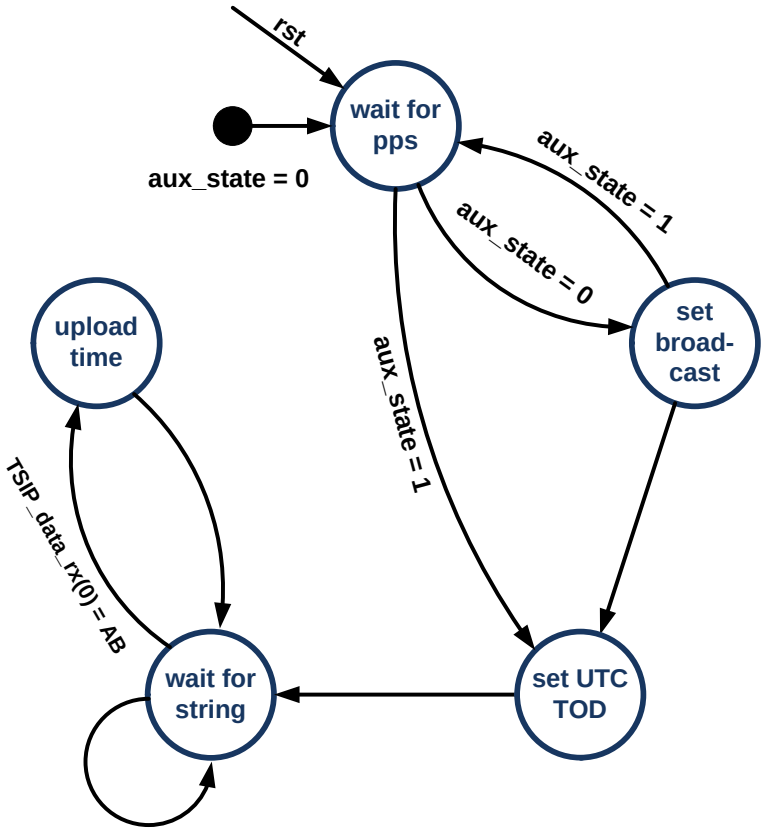


Figura 3.17 Máquina de estado de *main_logic*.

Este estado solo existe para sincronizar el inicio real del funcionamiento del sistema en dos segundos (PPS).

- *set_broadcast*: Se comunica con el componente *thunderbol_communication* para transmitir la máscara <DLE, 0x8E, 0xA5, 0x00, 0x05, 0x00, 0x00, DLE, ETX> al receptor GPS y este active el envío delos paquetes 0x8F-AB y 0x8F-AC automáticamente después de cada PPS. Al completar la transmisión regresa al estado *wait_for_pps*.

  Aunque el receptor GPS viene configurado por defecto para enviar los paquetes 0x8F-AB y 0x8F-AC, este estado sirve para asegurar ese comportamiento.

- *set_UTC_TOD*: Se comunica con el componente *thunderbol_communication* para transmitir la máscara <DLE, 0x8E, 0xA2, 0x01, DLE, ETX> al receptor GPS y este establezca los campos de "Time of Day" del paquete 0x8F-AB en escala UTC (para utilizar la información en la detección de segundos intercalares ya que por

defecto esos campos estan en escala GPS). Al completar la transmisión pasa al estado *wait_for_string*.

- *wait_for_string*:

  Espera un aviso (1) en la señal *new_pack* proveniente de *thunderbolt_communication* que indique que se ha leído un nuevo paquete TSIP. A parte del paquete 0x45 (que informa sobre la versión de software al encendido del receptor), los únicos paquetes que están configurados para transmisión desde el receptor GPS gracias a la máscara enviada en el estado *set_broadcast* son el 0x8F-AB y el 0x8F-AC.

  Ambos paquetes son evaluados, del paquete 0x8F-AB se extrae la información de tiempo GPS (si es válida *time_set* pasa a 0), la información UTC (cuando se adquiere la información *UTC_info* pasa a 0). Cuando ambas banderas (*time_set* y *UTC_info*) estan en estado bajo, se puede procesar la data del GPS y de UTC para generar una marca POSIX (*unix_load*) y pasar al estado *upload_time*.

  Del paquete 0x8F-AC se extrae la información referente a los segundos intercalares y modifica *leap_pending* dependiendo del bit 7 en el byte 11 de datos del paquete TSIP. También copia los bytes 1 y 2 del mismo paquete para actualizar las salidas *receiver_mode* y *disciplining_mode* que irán al componente *register_map*.

  En caso de existir un error avisado por el componente *thunderbolt_communication* a través de la señal *read_error*, se pasará al estado por defecto (*wait_for_pps*).

- *upload_time*:

  Se encarga de activar *l_trigger* para indicarle al componente *count_unix* que ya puede cargar los datos en *UNIX_load*.

La señal que almacena los estados es *main_state* que es una salida hacia el componente *thunderbol_communication*.

3.1.7 Componente memoria de registros

El componente *registers_map* (Figura 3.18) es una memoria de 16 direcciones que llena sus registros con configuraciones para los demás componentes del sistema o información de estado que puede ser solicitada por el usuario, para esto se comunica con varios de esos componentes a través de las señales de la Tabla 3.13.

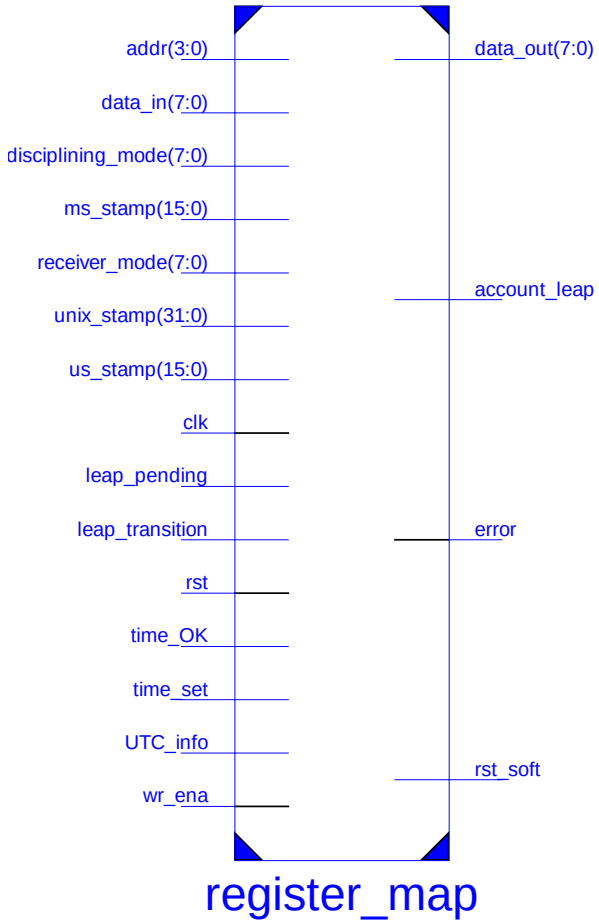


Figura 3.18 Componente *register_map*.

En la Figura 3.19 se puede ver que la memoria tiene 16 direcciones a las cuales se accede a través de un decodificador de 4 bits (*addr*). Para escribir o leer las direcciones se usa la señal *wr_ena* (activa para escritura en nivel alto). La matriz de memoria esta formada por registros de 8 bits como se puede ver en la Tabla 3.14. Las direcciones 0 y 1 son las únicas que se pueden escribir y leer, el resto son de solo lectura y si se tratan de escribir se levantará la bandera *error*. La señal *clk* se agrega para sincronizar el cambio de nivel en la comunican con los demás componentes del sistema.

La señal *data_in* se usa para cargar el byte a la dirección indicada en *addr* y debe permanecer estable durante el flanco ascendente de *clk*. Por su parte *data_out* permanecerá en el valor de la última dirección válida registrada en el flanco ascendente de *clk*. La señal *rst* regresa los registros de las direcciones 0 y 1 a su valor por defecto: 0 = 0b0000000X (X es el valor de *RST_LVL* negado) y 1 = 0b01101011, en caso que la señal *error* estuviera en 1 la retorna a 0.

Tabla 3.13 Puertos de componente *register_map*.

Parámetros	Tipo	Modo	Descripción
CONTROL			
clk	*std_logic*	in	Reloj
rst	*integer*	in	Reinicio
OPERACIÓN			
addr	*std_logic_vector*	in	Dirección de memoria
data_in	*std_logic_vector*	in	Data a escribir
data_out	*std_logic_vector*	out	Data leída
wr_ena	*std_logic*	in	Selección escritura o lectura
error	*std_logic*	out	Error de acceso
REGISTROS			
RST			
rst_soft	*std_logic*	out	Reinicio por software
LEAP CONFIGURATION			
leap_month	*std_logic_vector*	out	Mes de próximo seg. intercalar
leap_day	*std_logic_vector*	out	Día de próximo seg. intercalar
leap_sign	*std_logic*	out	Seg. intercalar positivo o negativo
accout_leap	*std_logic*	out	Configuración para seg. intercalares
PULSES COUNT			
c_stamp	*unsigned*	in	Número de pulsos de reloj (marca)
DISCIPLINING MODE			
disciplining_mode	*std_logic_vector*	in	Modo de configuración GPS
RECEIVER MODE			
receiver_mode	*std_logic_vector*	in	Condición de operación de reloj
LEAP STATUS			
leap_transition	*std_logic*	in	Segundo inrtercalar en curso
leap_pending	*std_logic*	in	Segundo intercalar pendiente
TIME STATUS			
read_error	*std_logic*	in	Fallo en comunicación con receptor GPS
time_OK	*std_logic*	in	Tiempo válido
UTC_info	*std_logic*	in	Información UTC
time_set	*std_logic*	in	Información de tiempo GPS
TIME STAMPS			
unix_stamp	*unsigned*	in	Marca de tiempo POSIX
ms_stamp	*unsigned*	in	Marca de tiempo milisegundos
us_stamp	*unsigned*	in	Marca de tiempo microsegundos

A esta memoria se accede a través del componente *bus_SPI* que interpreta los comandos de lectura y escritura enviados por dicha interfaz.

La descripción de cada registro se presenta a continuación:

- *RST*: reinicio del sistema, toma en cuenta solo el último bit. El registro es entrada del componente *reset_manager* (a través del componente *pulse_stretcher*).

Tabla 3.14 Memoria de escritura y lectura del sistema GTS.

(W \R)	MSB (7)	ADDR (6:4)	ADDR (3:0)	DATA (7:0)								Descripción
W\R	1\0	000	0 0	0	0	0	0	0	0	0	rst	RST
W\R	1\0	000	1	month_ 3	month_ 2	month_ 1	month_ 0	day_ 1	day_ 0	sign	account _leap	LEAP CONFIG
R	0	000	2	0	0	0	0	0	0	0	0	EMPTY
R	0	000	3	X	X	X	X	X	X	X	X	PULSES
R	0	000	4	disciplining _mode(7)	disciplining _mode(6)	disciplining _mode(5)	disciplining _mode(4)	disciplining _mode(3)	disciplining _mode(2)	disciplining _mode(1)	disciplining _mode(0)	DISC MODE
R	0	000	5	receiver_ mode(7)	receiver_ mode(6)	receiver_ mode(5)	receiver_ mode(4)	receiver_ mode(3)	receiver_ mode(2)	receiver_ mode(1)	receiver_ mode(0)	REC MODE
R	0	000	6	0	0	0	0	0	0	leap_ transition	leap_ pending	LEAP STATUS
R	0	000	7	0	0	0	0	reading_ error	time_OK	UTC_info	time_set	TIME STATUS
R	0	000	8	X	X	X	X	X	X	X	X	UNIX_3
R	0	000	9	X	X	X	X	X	X	X	X	UNIX_2
R	0	000	A	X	X	X	X	X	X	X	X	UNIX_1
R	0	000	B	X	X	X	X	X	X	X	X	UNIX_0
R	0	000	C	X	X	X	X	X	X	X	X	MS_1
R	0	000	D	X	X	X	X	X	X	X	X	MS_0
R	0	000	E	X	X	X	X	X	X	X	X	US_1
R	0	000	F	X	X	X	X	X	X	X	X	US_0

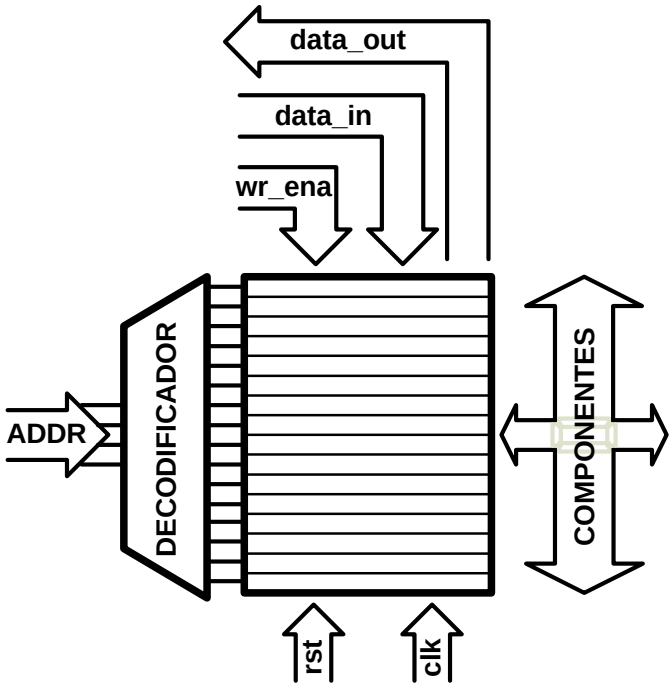


Figura 3.19 Diagrama de componente *register_map*.

- *LEAP CONFIG*: indica la configuración del sistema frente a un segundo intercalar y la fecha en que ocurriría. Las señales de este registro son entradas del componente *monitor_leap*.
- *EMPTY*: único registro vacío, siempre es 0.
- *PULSES*: almacena la marca de tiempo *c_stamp* proveniente del componente *stamp_sync*.
- *DISC MODE*: ofrece información sobre la condición de operación del reloj del receptor GPS, corresponde al byte de datos 2 del paquete 0x8F-AC. Esta información se extrae del componente *main_logic*.
- *REC MODE*: ofrece información sobre el modo de configuración del receptor GPS, corresponde al byte de datos 1 del paquete 0x8F-AC. Esta información se extrae del componente *main_logic*.
- *LEAP STATUS*: indica si hay un segundo intercalar pendiente (información extraída del componente *main_logic* por el paquete 0x8F-AC) y también cuando se da dicha transición (bit producido por el componente *monitor_leap*).

- *TIME STATUS*: indica si se obtuvo la información de tiempo GPS y la información UTC (a partir del paquete 0x8F-AB decodificado en componente *main_logic*), además si el tiempo ofrecido por el sistema ya es válido (bandera producida por el componente *count_unix*) y si hay un error de comunicación con el receptor GPS (detectado por el componente *main_logic*).
- *UNIX_3*, *UNIX_2*, *UNIX_1*, *UNIX_0*: almacenan la marca unix en sus 4 bytes, provienen del componente *stamp_sync*.
- *MS_1*, *MS_0*: almacenan la marca de milisegundos en sus 2 bytes, provienen del componente *stamp_sync*.
- *US_1*, *US_0*: almacenan la marca de microsegundos en sus 2 bytes, provienen del componente *stamp_sync*.

3.1.8 Componente SPI esclavo

El componente *SPI_slave* (Figura 3.20) fue implementado para establecer un bloque de comunicación SPI esclavo, toma los parámetros de configuración de la Tabla 3.15 y cuenta con las entradas y salidas descritas en la Tabla 3.16.

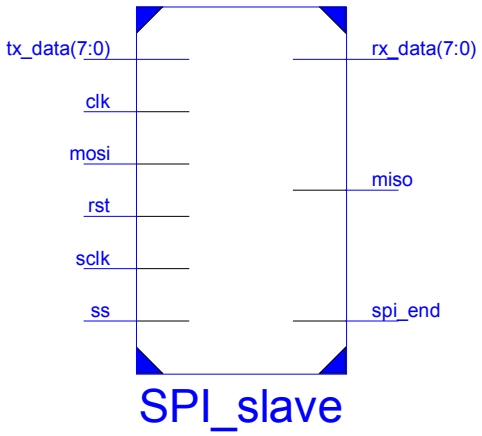


Figura 3.20 Componente *SPI_slave*.

Tabla 3.15 Parámetros de componente *SPI_slave*.

Parámetros	Tipo	Descripción
RST_LVL	*std_logic*	Nivel lógico de señal de reinicio
CPOL	*std_logic*	Polaridad de la señal de reloj
CPHA	*std_logic*	Fase de la señal de reloj

Tabla 3.16 Puertos de componente *SPI_slave*.

Puertos	Tipo	Modo	Descripción
INTERFAZ EXTERNA			
sclk	*std_logic*	in	Reloj serial de maestro SPI
mosi	*std_logic*	in	Línea de recepción serial
miso	*std_logic*	out	Línea de transmisión serial
ss	*std_logic*	in	Selección de esclavo SPI
CONTROL			
clk	*std_logic*	in	Reloj
rst	*std_logic*	in	Reinicio
DATA			
rx_data	*std_logic_vector*	out	Data recibida en transacción SPI
tx_data	*std_logic_vector*	in	Data a transmitir en transacción SPI
INTERFAZ LÓGICA			
spi_end	*std_logic*	out	Bandera de finalización de recepción

El componente se analiza mejor describiendo cada proceso que realiza: *ss_shift_register*, *mosi_shift_register*, *sclk_shift_register*, *spi_cycle*, *spi_transaction* y *spi_finish* (Figura 3.21); todos trabajan basados en el flanco de subida de la señal *clk* y los tres últimos pueden reiniciarse mediante la señal *rst*.

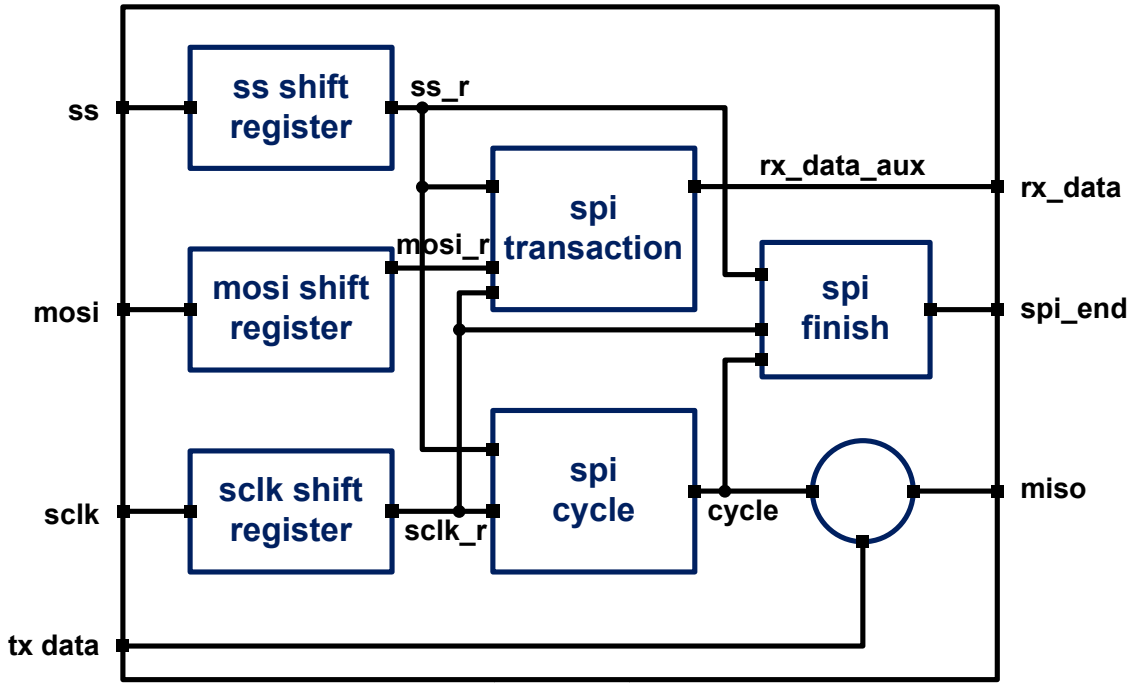


Figura 3.21 Diagrama de componente *SPI_slave*.

ss_shift_register y *mosi_shift_register* pasan sus entradas a través de un FFD para que cuando los otros procesos lean las salidas *ss_r* y *mosi_r* se forme un circuito estable.

sclk_shift_register pasa su entrada por dos FFD para ofrecer una salida donde se pueda detectar flancos de subida o bajada, donde el bit de la posición cero es el valor más reciente de la señal externa detectado en un flanco de subida de *clk*. La función de *ss_r* es similar a la de la señal de reinicio, mientras este en 1 ninguno de los procesos a los que se conecta funcionarán.

Gracias a las declaraciones genéricas, el bloque puede configurarse para operar en las 4 condiciones de reloj (*CPOL*, *CPHA*). En la Figura 3.22 se puede ver el diagrama de tiempos del bloque para los 4 modos de reloj.

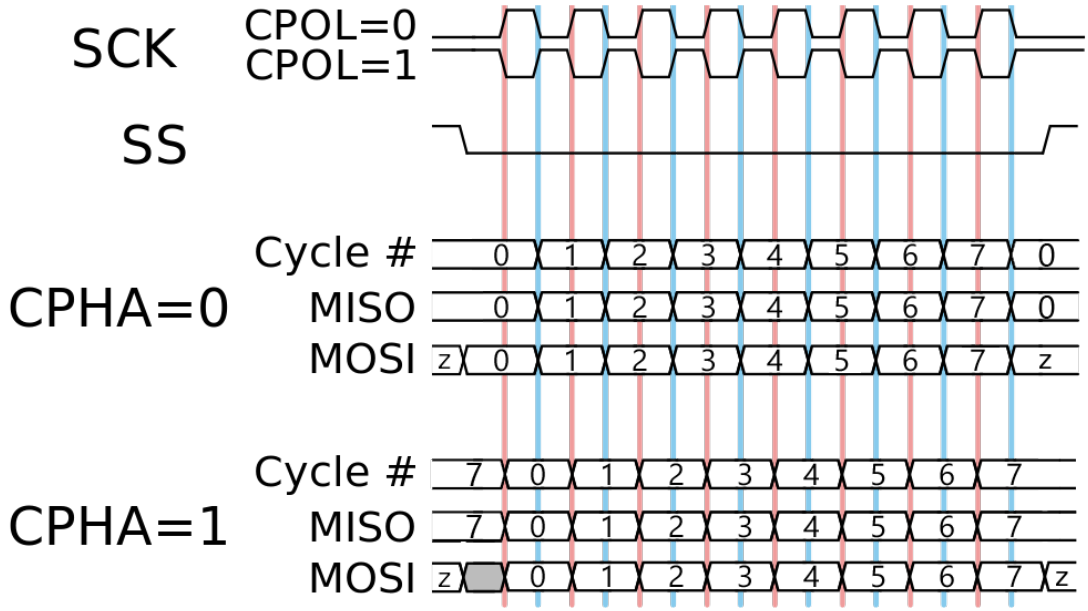


Figura 3.22 Diagrama de tiempos de transacción SPI. Modificación de ©Cburnett / Wikimedia Commons / CC-BY-SA-3.0 / GFDL.

Para la operación de este componente se toman los flancos detectados en la señal *sclk_r* como flancos de lectura o escritura, cuya secuencia binaria depende de los valores *CPOL* y *CPHA* (ver Tablas 3.17 y 3.18) que dan como resultado las funciones lógicas:

wr_edge (1 downto 0) := (CPOL xnor CPHA) & (CPOL xor CPHA) y

rd_edge (1 downto 0) := (CPOL xor CPHA) & (CPOL xnor CPHA),

asignadas a las constantes *wr_edge* y *rd_edge*.

El proceso *spi_cycle* toma en cuenta la señal interna *cycle*, que es un contador binario de 3 bits con un valor por defecto correspondiente a un vector con todos sus bits configurados al valor de *CPHA*. Cuando se detecta una secuencia *rd_edge* en la señal *sclk* (flanco de lectura), *cycle* aumenta en uno estableciendo así el ciclo de reloj en el que

Tabla 3.17 Flanco de escritura.

sclk_r(1:0)		CPOL	CPHA
1	0	0	0
0	1	0	1
0	1	1	0
1	0	1	1

Tabla 3.18 Flanco de lectura.

sclk_r(1:0)		CPOL	CPHA
0	1	0	0
1	0	0	1
1	0	1	0
0	1	1	1

se encuentra la transacción SPI. Inicialmente se actualizaba el contador al detectar una secuencia *wr_edge* pero para poder alcanzar velocidades mayores de transferencia se optó por adelantar la actualización medio ciclo. Fuera del proceso, la salida *miso* se actualiza dependiendo del valor del bit en la posición "*cycle*" del vector de entrada *tx_data*.

El proceso *spi_transaction* espera un flanco de lectura en la señal *sclk* para desplazar el vector de 7 bits *rx_data_aux* un bit hacia la izquierda. Este vector actualiza la salida *rx_data* fuera del proceso.

El proceso *spi_finish* tiene la función de indicar la finalización de la transferencia SPI y lo hace tomando en cuenta las señales *sclk_r* y *cycle*, detectando una secuencia *rd_edge* y el último ciclo de la transacción respectivamente.

Si solo se toma en cuenta la máxima frecuencia a la que el dispositivo maestro puede operar el bus SPI, y la máxima frecuencia permitida para las líneas del FPGA (cumpliendo con los tiempos t_{setup} y t_{hold}), en teoría la frecuencia máxima de transferencia SPI que se puede alcanzar responde a la fórmula:

$$2\ periodo_{clk} < periodo(sclk),$$

que requiere de un período *clk* pues no se sabe en que momento del ciclo de reloj llegará el flanco de subida de *sclk*, y otro período adicional para el procesamiento de la señal *sclk* en el dominio de reloj *clk*. En la Figura 3.23 se puede observar mejor el comportamiento de las señales para una frecuencia de 24MHz, donde se transmite el byte 0xB4 y se recibe el byte 0xA9, al ser solo una simulación de comportamiento no se toma en cuenta el jitter de las señales, ni los tiempos t_{rise} y t_{fall}.

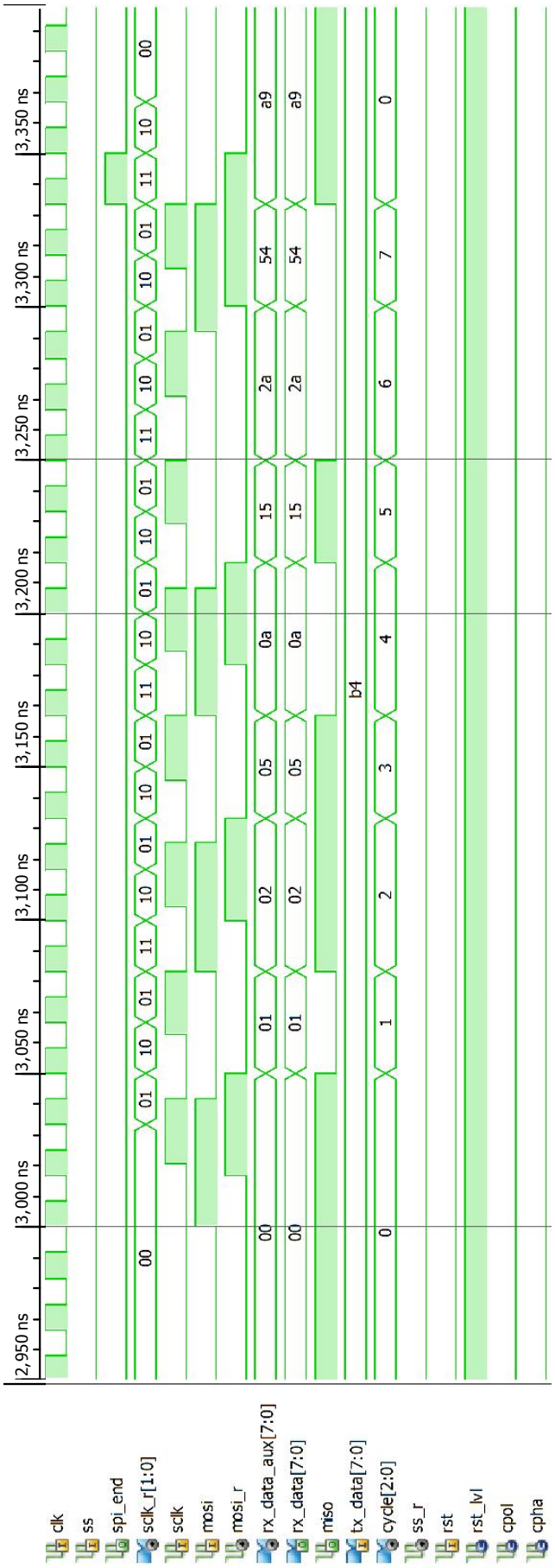


Figura 3.23 Testbench de componente *SPI_slave*.

En práctica debido a que las señales de reloj no están sincronizadas, el jitter que poseen y las restricciones de los tiempos t_{rise} y t_{fall}, se logrará una velocidad SPI significativamente menor a la mitad de la frecuencia del reloj *clk*.

La frecuencia máxima alcanzada por este componente fue de 15MHz, usando la tarjeta TIVA C tm4c1294, la siguiente frecuencia que se podía probar era de 30MHz que ya no cumplía los requisitos de la ecuación. En la Figura 3.24 se pueden ver las líneas MOSI, SCLK y SS, con las cuales el dispositivo maestro esta enviando el byte 0x6B. El flanco de subida de SCLK cae aproximadamente en la mitad del bit de datos en MOSI.

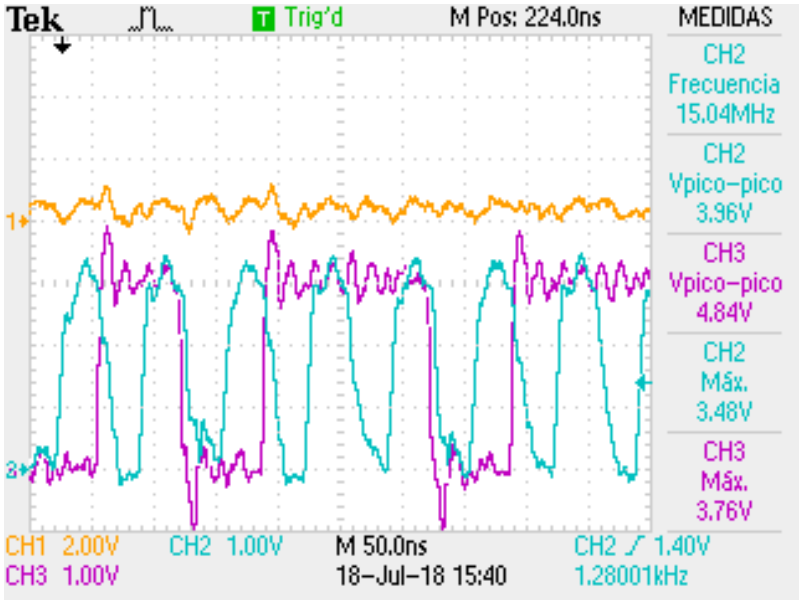


Figura 3.24 Transferencia SPI en TIVA a 15MHz. *SS* en amarillo, *SCLK* en cyan, *MOSI* en magenta.

La Figura 3.25 muestra el jitter de las señales del dispositivo maestro, las cuales alcanzan un valor de hasta 20ns como se ve en la señal *MOSI* (en magenta).

3.1.9 Componente de filtro antirebote

El componente *debouncer* (Figura 3.26, Tabla 3.19) fue el último componente creado, es simple y ya se había implementado en el proceso *rx_debouncer* del componente *UART* (Sección 3.1.2) y se diseñó para evitar cambios de estado indeseados en la línea *SS* antes de ingresar al componente *SPI_slave*.

La Figura 3.27 es una gráfica de persistencia donde se aprecia el cross-talk inducido por la señal *SCLK* (en magenta) a la línea *SS* (en cyan). La señal de color amarillo es una señal pasada por un FFD dentro del FPGA, la cual permite ver que la línea *SS* esta pasando

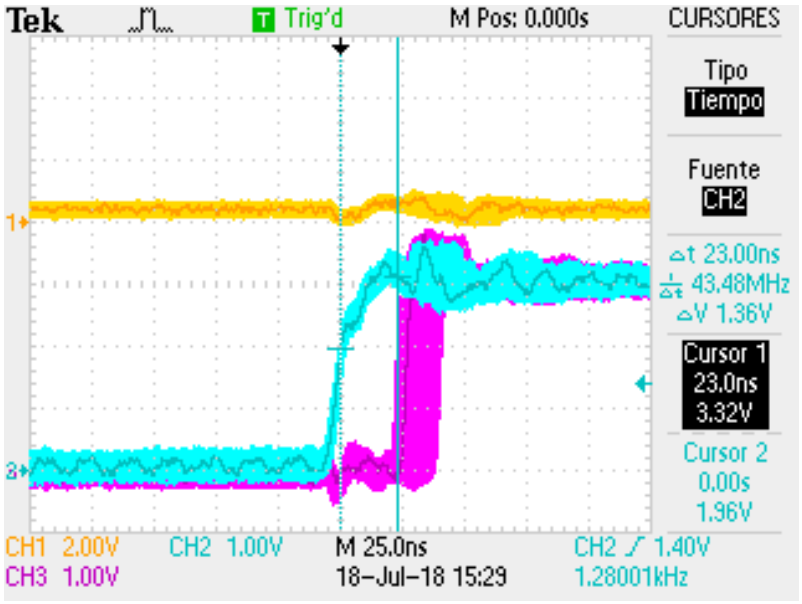


Figura 3.25 Jitter detectado en las señales SPI de la tarjeta TIVA.

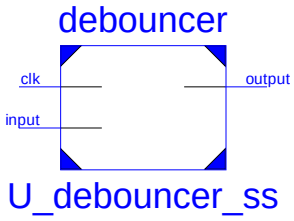


Figura 3.26 Componente *debouncer*.

Tabla 3.19 Puertos de componente *debouncer*.

Puertos	Tipo	Modo	Descripción
clk	*std_logic*	in	Reloj
input	*std_logic*	in	Señal externa
output	*std_logic*	out	Señal filtrada

a un estado alto cuando no debería. Esto fue de vital importancia a la hora de descartar errores en el sistema.

Por esto se aplicó la solución de pasar la señal *input* consecutivamente a través de dos FFD y comparar las salidas de ambos, si eran iguales se actualizaba la salida *output*, de lo contrario se mantenía el valor de salida anterior, evitando así reinicios indeseados en la transacción SPI.

En la Figura 3.28 ya se puede ver la señal *ss_deb* (en amarillo) pasada a través del componente *debouncer* reduciendo el cross-talk.

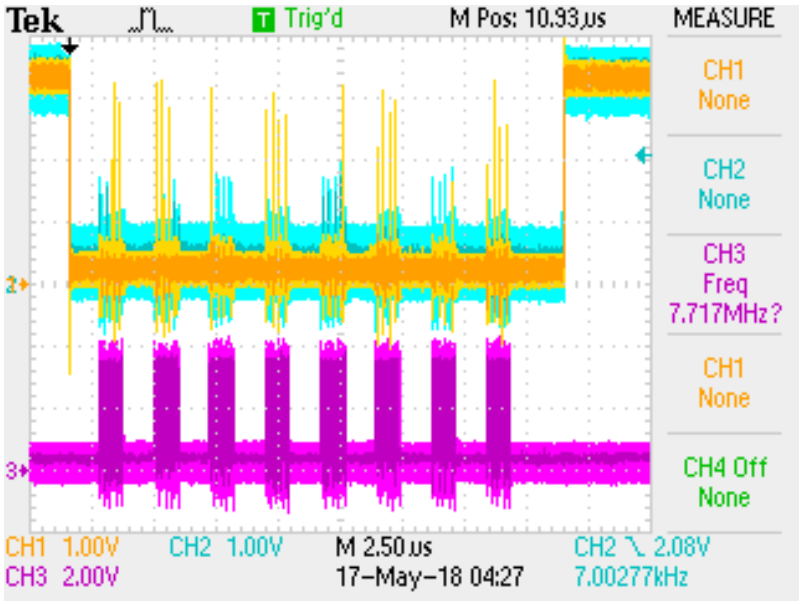


Figura 3.27 Consecuencias de cross-talk entre señales SPI.

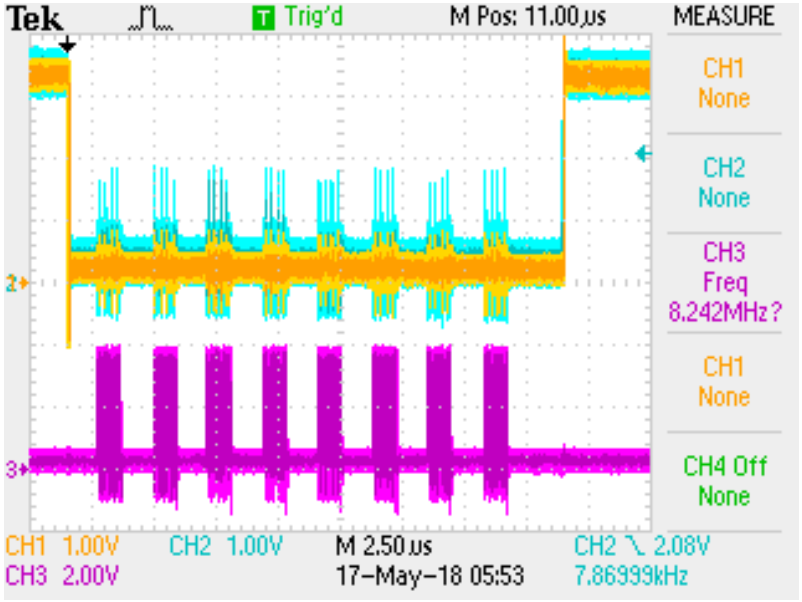


Figura 3.28 Señales SPI, con línea *SS* pasada por el filtro antirebote.

3.1.10 Componente de bus SPI

El componente *SPI_bus* (Figura 3.29, Tabla 3.20) fue diseñado para establecer la comunicación entre los bloques *register_map* y *SPI_slave* de manera que permite leer o escribir los registros del sistema a través de un dispositivo SPI maestro.

Provee al sistema con dos modos de operación: *auto* y *request*.

- *auto*: es el modo por defecto, ofrece una lectura inmediata de las marcas de tiempo

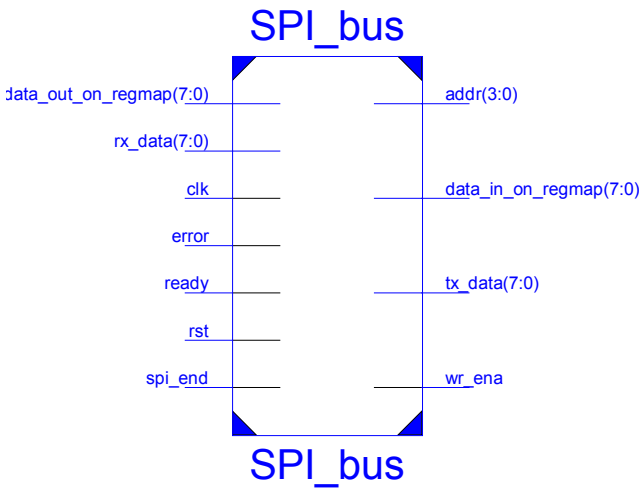


Figura 3.29 Componente *SPI_bus*.

Tabla 3.20 Puertos de componente *SPI_bus*.

Puertos	Tipo	Modo	Descripción
CONTROL			
clk	*std_logic*	in	Reloj
rst	*std_logic*	in	Reinicio
ACTUALIZACIÓN DE MARCAS			
ready	*std_logic*	in	Bandera: marcas actualizadas
INTERFAZ PARA ESCLAVO SPI			
rx_data	*std_logic_vector*	in	Data recibida
tx_data	*std_logic_vector*	out	Data a transmitir
spi_end	*std_logic*	in	B: finalización de recepción
INTERFAZ PARA MEMORIA			
addr	*std_logic_vector*	out	Dirección de memoria
data_in_on_regmap	*std_logic_vector*	out	Data a escribir
data_out_on_regmap	*std_logic_vector*	in	Data leída
wr_ena	*std_logic*	out	Selección escritura o lectura
error	*std_logic*	in	Error de acceso

(divididas en 8 bytes), se puede extraer dichos registros enviando el código 0xAA en 8 transferencias consecutivas. Para operar en este modo es muy útil usar la señal *ready_to_read* como señal de interrupción para el dispositivo SPI maestro, de manera que cada vez que haya un flanco de subida en esa señal se inicie una transacción de 8 bytes. Al momento que se envía un código distinto a 0xAA se pasa inmediatamente al modo *request*.

- *request*: en este modo se accede a cada registro independientemente, el dispositivo maestro debe enviar un código con formato 0bX000XXXX, si los bits 6, 4 y 5 no

son cero el sistema responderá con un código de error 0xF0. El bit 7 representa escritura/lectura del registro y los 4 LSB bits representan la dirección de memoria. En el caso de escritura, se espera por el siguiente byte leído para escribirlo en la dirección previamente seleccionada.

Estos modos se basan en dos procesos principales: *auto_stamps* y *transfer_mode*. El primer proceso selecciona automáticamente el byte a transmitir correspondiente a la marca de tiempo, su funcionamiento se base en las señales *ready* y *spi_end*, la primera indica en que momento la dirección de memoria (*addr*) debe volver a seleccionar el primer byte de las marcas, ya que estas han sido actualizadas; mientras que la segunda hace que la data a transmitir cambie desde la dirección 9 hasta la 15 de la memoria de registros, pasando así por los 8 bytes de la marca de tiempo. La señal de dirección controlada aquí es una señal auxiliar que se usa luego en el proceso que selecciona el comportamiento de la transferencia.

El segundo proceso maneja el modo de operación de la comunicación usando una máquina de estados y varias condiciones (ver Figura 3.30). Antes de pasar a la máquina se evalúa si hay un error al tratar de acceder a la memoria de registros, de haberlo se pasa inmediatamente al estado *error_state*, si no se espera por un cambio en la señal *spi_end* que indique la llegada de un nuevo byte (*rx_data*) y así actualizar el estado actual. Si *rx_data* es igual al código automático 0xAA se continuará en el modo *auto* y la señal *addr* irá a la siguiente dirección de la marca de tiempo, si no es igual a 0xAA se pasará al estado *detect_request* el cual esperará por un nuevo byte, si es el código automático retornará a la dirección indicada por el proceso *auto_stamps* y al estado *auto*, de ser diferente evaluará que los bits 4, 5 y 6 sean 0, si no lo son pasará al estado *error_state*, mientras que si cumple con la condición tomará los bits 4 bits LSB para asignarlos a la dirección *addr* y luego evaluará el bit 7 para determinar si es una operación de lectura (0) o escritura (1). De ser escritura se pasa al estado *ram_writing* el cual activa la señal *wr_ena* y asigna la data recibida (*rx_data*) a la señal de entrada para la memoria de registros (*data_in_on_regmap*), si es lectura se continuará hacia el estado *detect_request* pues ya se asigno la dirección a leer. Cuando la máquina se encuentra en el estado *error_state* y se recibe el byte 0xAA se pasa al estado *auto*, caso contrario se pasará al estado *detect_request* el cual evaluará la data de la manera que se mencionó anteriormente.

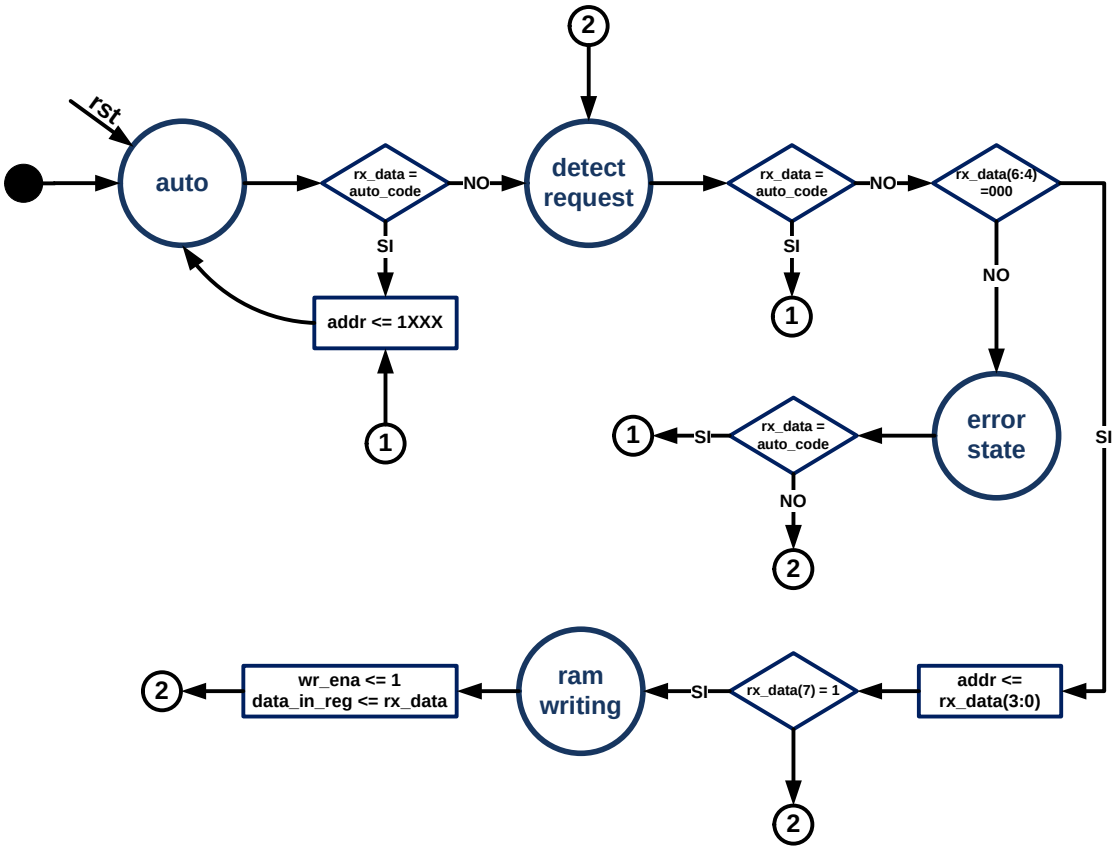


Figura 3.30 Máquina de estado de *SPI_bus*.

3.1.11 Componente monitor de segundos intercalares

El componente *monitor_leap* (Figura 3.31, Tabla 3.21) es el encargado de avisar el momento en que esta ocurriendo un segundo intercalar, para esto es necesario que el usuario configure el registro posicionado en la dirección 1 (*LEAP CONFIG*) (ver Tabla 3.14), donde:

- *month_2*, *month_1*, *month_0* representan el mes en que ocurrirá el segundo intercalar con un número binario del 1 al 12.
- *day_1*, *day_0* representan el día en que ocurrirá el segundo intercalar, con valores binarios del 28 al 31 donde solo se toman los dos últimos bits.
- *sign* representa el signo del segundo intercalar, 0 es negativo, 1 es positivo
- *account_leap* indica si la marca unix no se actualizará con segundos intercalares (0), o de ser 1 si sí tomará en cuenta estos segundos.

El componente evalúa si se toma en cuenta el conteo de un segundo intercalar, de ser el caso pasa a comparar las señales de fecha del registro *LEAP_CONFIG* con el tiempo

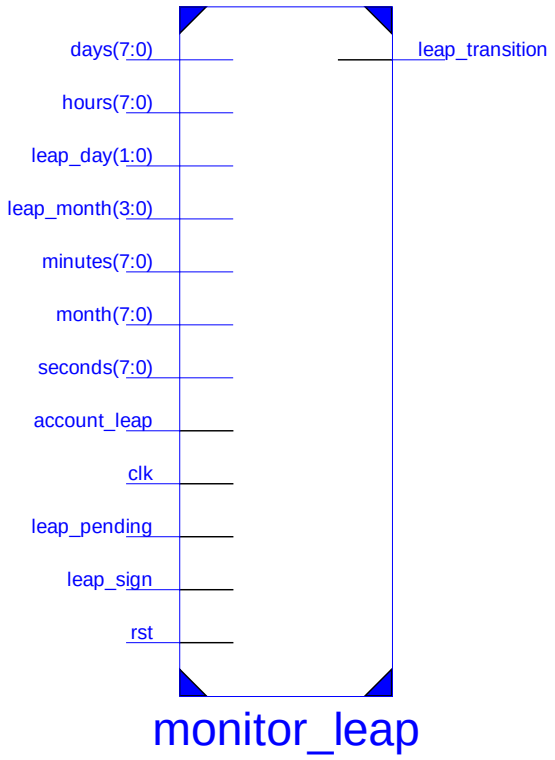


Figura 3.31 Componente *monitor_leap*.

Tabla 3.21 Puertos de componente *monitor_leap*.

Puertos	Tipo	Modo	Descripción
CONTROL			
clk	*std_logic*	in	Reloj
rst	*std_logic*	in	Reinicio
DE PAQUETE 0x8F-AB			
seconds	*std_logic_vector*	in	Segundos UTC
minutes	*std_logic_vector*	in	Minutos UTC
hours	*std_logic_vector*	in	Horas UTC
days	*std_logic_vector*	in	Días UTC
month	*std_logic_vector*	in	Mes UTC
REGISTRO "LEAP CONFIG"			
leap_month	*std_logic_vector*	in	Mes de próximo seg. intercalar
leap_day	*std_logic_vector*	in	Día de próximo seg. intercalar
leap_sign	*std_logic*	in	Seg. intercalar positivo o negativo
account_leap	*std_logic*	in	Configuración para seg. intercalares
REGISTRO "LEAP STATUS"			
leap_pending	*std_logic*	in	Seg. intercalar pendiente
leap_transition	*std_logic*	out	Seg. intercalar ocurriendo

actual extraído del paquete 0x8F-AB, si es la misma fecha espera el último segundo del día para indicar una transición del segundo intercalar.

3.1.12 Componente de cronometrización de señal externa

El componente *stamp_sync* (Figura 3.32, Tabla 3.22) tiene dos funciones principales: cronometrar la señal *sync* y proveer la señal *ready_to_read*.

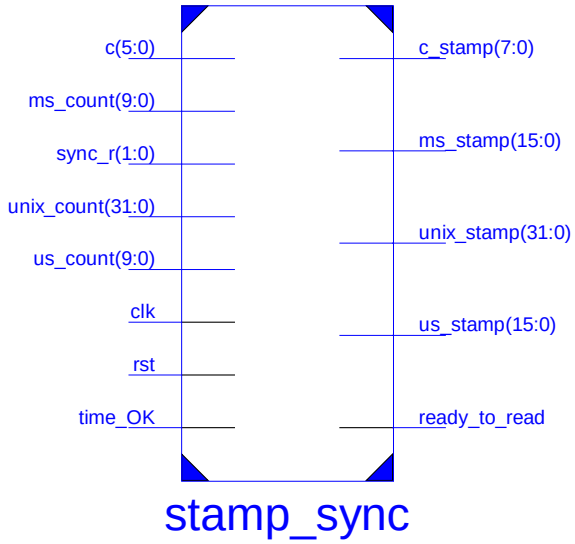


Figura 3.32 Componente *stamp_sync*.

Tabla 3.22 Puertos de componente *stamp_sync*.

Puertos	Tipo	Modo	Descripción
CONTROL			
clk	*std_logic*	in	Reloj
rst	*std_logic*	in	Reinicio
SEÑAL A MARCAR			
sync_r	*std_logic_vector*	in	Señal a cronometrar
CONTADORES			
unix_count	*unsigned*	in	Contador de segundos
ms_count	*unsigned*	in	Contador de milisegundos
us_count	*unsigned*	in	Contador de microsegundos
c	*unsigned*	in	Contador de pulsos
MARCAS DE TIEMPO			
unix_stamp	*unsigned*	out	Marca de segundos (4 bytes)
ms_stamp	unsigned	out	Marca de milisegundos (2)
us_stamp	*unsigned*	out	Marca de microsegundos (2)
c_stamp	*unsigned*	out	Marca de pulsos
INTERFAZ DE COMUNICACIÓN			
time_OK	*std_logic*	in	Tiempo es válido
ready_to_read	*std_logic*	out	Indica actulización de marcas

Cuando se detecta un cambio de estado de 1 a 0 en la señal *sync_r*, los contadores representados por las señales de entrada *c*, *us_count*, *ms_count* y *unix_count* son registrados

en las señales *c_stamp*, *us_stamp*, *ms_stamp* y *unix_stamp*. Estas señales almacenan el instante de tiempo en el que llegó la señal *sync* en un formato de n-bytes y son entradas a la memoria de registro en las direcciones 3, y de la 9 a la 15.

Al detectar el mismo cambio de estado en *sync_r* se genera un pulso de $1\mu s$ en la señal *ready_to_read* que al ser una salida del sistema puede ser utilizada como señal de interrupción para el dispositivo que maneja el GCTS.

Un ejemplo del comportamiento del componente se puede ver en la Figura 3.33 donde se observan a los 4 contadores operando en base a la señal de reloj, luego el pulso *sync* llega en el instante 118 352 872 ns, este pulso es captado por el inmediato flanco de subida de *clk* y al siguiente se activa la señal *ready_to_read* y se actualizan los registros de las marcas de tiempo con los valores del primer flanco de subida de *clk* después de la llegada del pulso *sync*. La frecuencia de la señal *clk* es 60MHz.

3.1.13 Componente de reinicio al encendido

El componente *power_on_reset* (Figura 3.34, Tabla 3.23) emula un circuito *power-on reset* que cuando inicia el sistema genera una señal impulso de reinicio que va a lo largo de todo el circuito fijando las señales en un estado inicial conocido.

Tabla 3.23 Puertos de componente *power_on_reset*.

Puertos	Tipo	Modo	Descripción
clk	*std_logic*	in	Reloj
rst_int	*std_logic*	out	Señal de reinicio interno

Esto se logra manteniendo la señal *rst_int* en el nivel *RST_LVL* (indicado con declaración generic) mientras un contador interno, que se actualiza a cada flanco de subida de *clk*, llega un valor máximo determinado. Luego se desactiva la señal *rst_int* llevándola al valor opuesto a *RST_LVL*.

3.1.14 Componente de manejo de señales de reinicio

El componente *reset_manager* (Figura 3.35, Tabla 3.24) consiste en una compuerta de 3 entradas la cual se configura con la declaración *GENERIC* y el parámetro *RST_LVL* de tipo *std_logic* que toma valor de 1 o 0 y determina si la compuerta es AND o OR.

El reinicio manual se da por la señal *rst_man* al presionar el botón indicado, el reinicio inicial proviene del componente *power_on_reset* con la señal *rst_int* y el reinicio

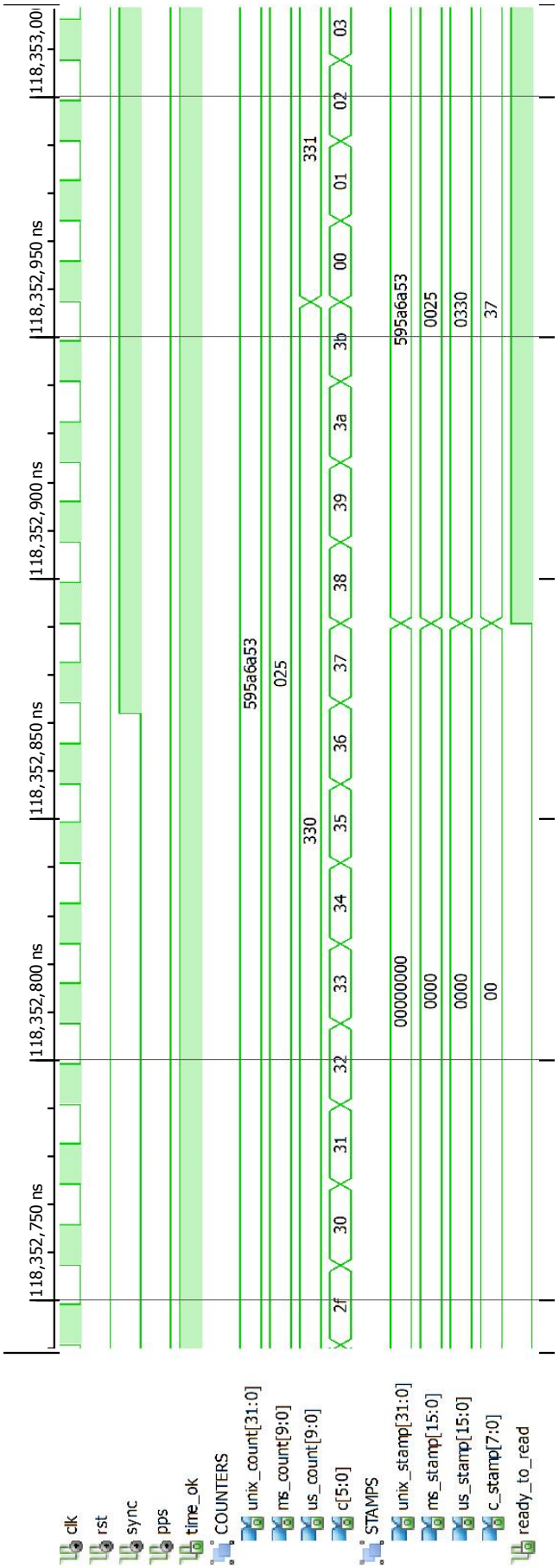


Figura 3.33 Testbench de componente *stamp_sync*.

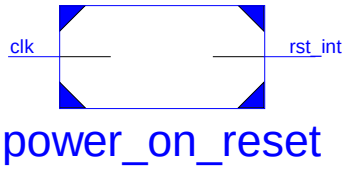


Figura 3.34 Componente *power_on_reset*.

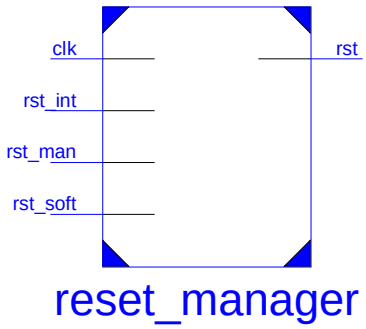


Figura 3.35 Componente *reset_manager*.

por software con la señal *rst_soft* se da mediante el cambio del registro de la dirección 0 del componente *register_map* a través de un comando enviado por el usuario por el bus SPI.

Tabla 3.24 Puertos de componente *reset_manager*.

Puertos	Tipo	Modo	Descripción
clk	*std_logic*	in	Reloj
rst_man	*std_logic*	in	Reinicio manual
rst_int	*std_logic*	in	Reinicio inicial
rst_soft	*std_logic*	in	Reinicio por software
rst	*std_logic*	out	Señal de reinicio global

Cuando *RST_LVL* es 1 se forma una compuerta OR de 3 entradas cuya salida pasa por un FF-D (ver Figura 3.36).

3.1.15 Componente ensanchador de pulso

El componente *pulse_stretcher* (Figura 3.37, Tabla 3.25) es un componente simple que tiene como único propósito ensanchar el pulso de reinicio por software mediante las declaraciones genéricas *PULSES* (número de pulsos de reloj a extender la entrada) y *SET_VAL* (valor activo de la entrada). Por defecto ofrece un reinicio de $1\mu s$ para el sistema, pero puede modificarse de acuerdo a requerimientos futuros.

Se usa entre la salida del registro *RST* del componente *register_map* y la entrada al componente *reset_manager*.

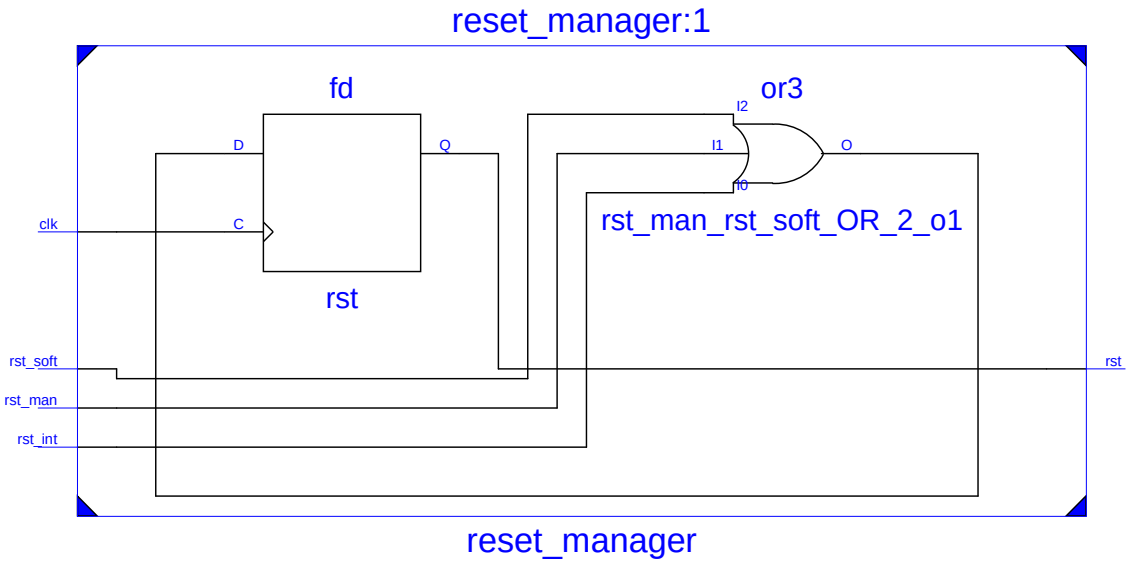


Figura 3.36 Diagrama de componente *reset_manager*.

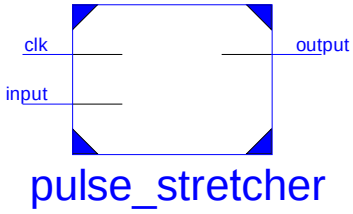


Figura 3.37 Componente *pulse_stretcher*.

Tabla 3.25 Puertos de componente *pulse_stretcher*.

Puertos	Tipo	Modo	Descripción
clk	*std_logic*	in	Reloj
input	*std_logic*	in	Pulso a ensanchar
output	*std_logic*	out	Pulso ensanchado

4 Resultados y Discusión

Se realizaron tres tipos de pruebas: *i*, *ii*, *iii*.

Las pruebas tipo *i* se realizaron para verificar el correcto funcionamiento del sistema observando su respuesta a señales *SYNC* con diferentes periodos a partir de pulsos extraídos del CR.

Las pruebas tipo *ii* y *iii* se realizaron para validar la portabilidad del sistema en distintos FPGA, se configuró dos sistemas GCTS independientes y se observó la respuesta de ambos a una misma señal de excitación *SYNC*.

4.1 Pruebas tipo *i*

En las pruebas tipo *i* se analizó la diferencia de tiempo entre los pulsos *SYNC*, verificando la continuidad del periodo de la señal. Se utilizó una tarjeta FPGA, el CR, el receptor Thunderbolt E, el generador de señales y una tarjeta basada en un microcontrolador, se puede ver un diagrama de las conexiones en la Figura 4.1 y los equipos en la Figura 4.2. Estos equipos trabajaron de la siguiente manera:

- El receptor GPS proporcionó el reloj de referencia de 10MHz, que el generador de señales usó como señal de referencia externa y lo multiplicó por 6 para que el CR reciba una señal de 60MHz, además en la mayoría de experimentos también se usó el pulso PPS (proporcionado por el receptor GPS), así el Controlador de Radar operó en sincronismo total con el tiempo GPS.
- El GCTS funcionó a partir de: la señal *CLK* proporcionada por el CR, los datos obtenidos por comunicación serial del receptor ThunderBolt E y la señal *PPS*. Podía

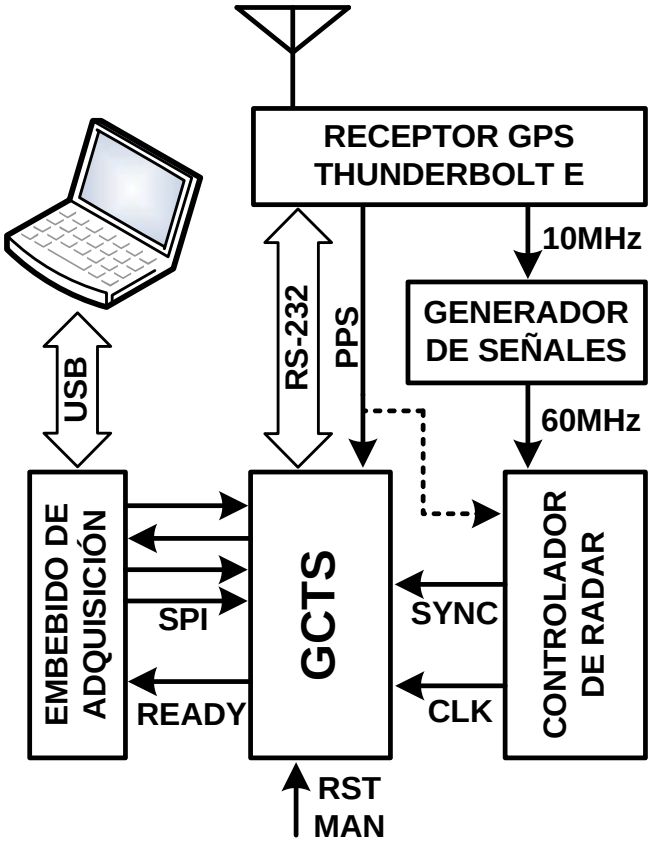


Figura 4.1 Diagrama de conexiones para pruebas tipo I.

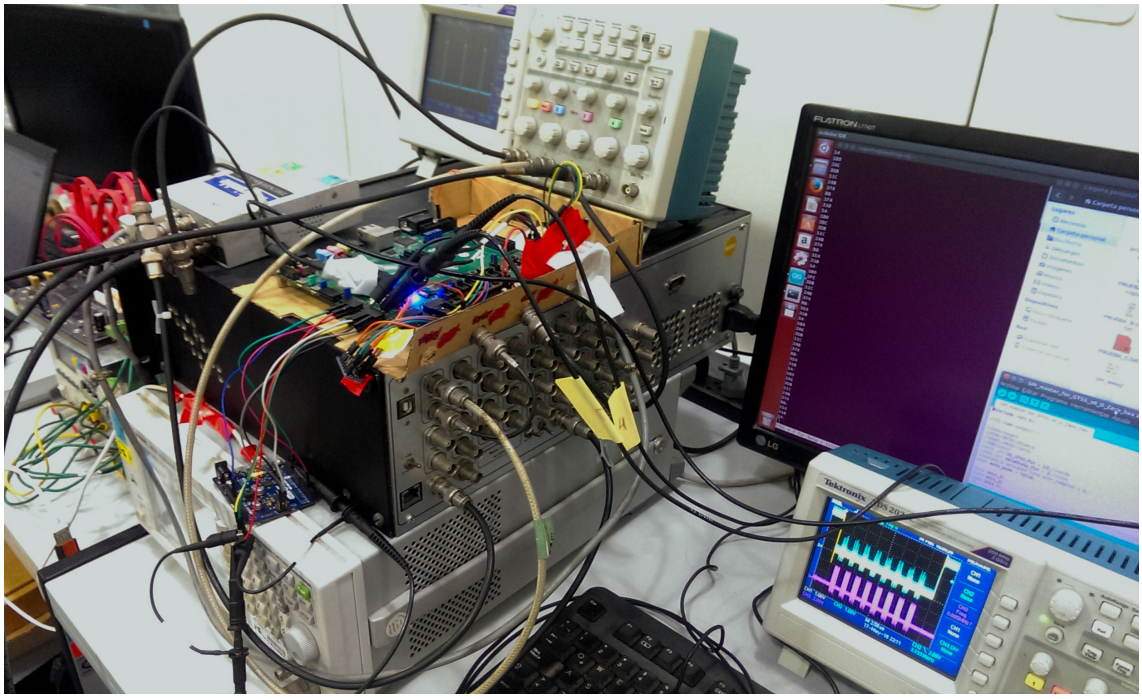


Figura 4.2 Equipos usados en las pruebas tipo I.

ser reiniciado manualmente mediante *RST_MAN* y fue excitado en su entrada *SYNC* por la señal TR o TXA del CR, luego de recibir esta señal, si el sistema ya había adquirido la información de tiempo necesaria, activó la señal *READY* por $1\mu s$ y repitió esta acción por cada pulso en la entrada *SYNC*.

- El embebido de adquisición configuró los parámetros de operación del GCTS mediante un bus SPI, luego este dispositivo tomó la señal *READY* como un pin de interrupción el cual daba inicio a una transacción para extraer las marcas de tiempo y enviarlas en formato hexadecimal como texto a través de USB a la computadora para que almacene los datos.
- Finalmente, con un script de Python se calculó las diferencias entre las marcas de tiempo contiguas y se compararon con el periodo correspondiente a la frecuencia del pulso *SYNC*.

En la Tabla 4.1 se detallan las configuraciones y resultados de cada una de las 17 pruebas tipo *i* realizadas, identificadas desde la letra A hasta la Q. Las columnas representan lo siguiente:

- "Frec. señal" indica la frecuencia de la señal *SYNC*.
- "Sinc." indica si durante la prueba el CR se sincronizó con el PPS.
- "Señal (CR)" indica que señal del CR se cronometró.
- "Tarjeta FPGA" indica en que tarjeta se implementó el sistema.
- "Embebido de adquisición" indica la tarjeta con la cual se extrajo la data del sistema.
- "SPI (MHz)" indica la velocidad a la que funcionó el bus SPI.
- "Marcas almacenadas" indica que registros correspondientes a las marcas de tiempo fueron almacenados en la PC.
- "Fallos" indica el número de errores encontrados en los datos almacenadas.
- "Tiempo (hh:mm)" indica la duración de la prueba en horas y minutos.
- "Errores (ppm)" indica la relación (en partes por millon) del número de errores con respecto a la cantidad total de marcas de tiempo obtenidas.

Tabla 4.1 Pruebas tipo *i*.

Prueba	Frec. señal	Sinc.	Señal (CR)	Tarjeta FPGA	Embebido de adquisición	SPI (MHz)	Marcas almacenadas	Fallos	Tiempo (hh:mm)	Errores (ppm)
A	8KHz	PPS	TR	ZedBoard	Teensy 3.2	8	*US_1*, *US_0*	1	00:50	0.04
B	8KHz	—	TR	ZedBoard	Teensy 3.2	8	*US_1*, *US_0*	0	01:37	0.00
C	8KHz	PPS	TR	ZedBoard	Teensy 3.2	8	*US_1*, *US_0*	1	01:37	0.02
D	8KHz	PPS	TR	ZedBoard	Teensy 3.2	8	*US_1*, *US_0*	1	01:39	0.02
E	4KHz	PPS	TR	ZedBoard	Teensy 3.2	8	*US_1*, *US_0*	0	00:46	0.00
F	4KHz	PPS	TR	ZedBoard	Teensy 3.2	8	*US_1*, *US_0*	0	01:56	0.00
G	3,33KHz	—	TR	ZedBoard	Arduino Zero	6	*US_1*, *US_0*	1	01:02	0.08
H	3,33KHz	—	TR	ZedBoard	Arduino Zero	6	*US_1*, *US_0*	0	00:57	0.00
I	1KHz	PPS	TR	ZedBoard	Arduino Zero	6	todas	1	13:53	0.02
J	1KHz	PPS	TXA	ZedBoard	Arduino Zero	6	todas	1	04:54	0.06
K	1KHz	PPS	TXA	X-SP6-X9	Teensy 3.2	6	todas	1	01:45	0.16
L	1KHz	PPS	TXA	X-SP6-X9	Arduino Zero	6	todas	0	01:24	0.00
M	500Hz	PPS	TXA	ZedBoard	Arduino Zero	6	todas	0	01:26	0.00
N	500Hz	—	TXA	ZedBoard	Arduino Zero	6	todas	0	14:49	0.00
O	500Hz	PPS	TXA	ZedBoard	Arduino Zero	6	todas	0	02:07	0.00
P	500Hz	PPS	TR	ZedBoard	Arduino Zero	6	todas	1	14:13	0.04
Q	20Hz	PPS	TXA	ZedBoard	Arduino Zero	6	todas	0	14:19	0.00

A pesar de obtener los 8 bytes de las marcas de tiempo, debido al tiempo necesitado por las operaciones en el Teensy 3.2 y el Arduino Zero, en las pruebas con frecuencias mayores a 1KHz solo se transmitió a la computadora las marcas correspondientes a los microsegundos (*US_1*, *US_0*). Extractos de algunas pruebas se muestran decodificados en la Tabla 4.2, donde se puede ver la diferencia entre marcas consecutivas de $125\mu s$ para las pruebas A y B, $250\mu s$ para la prueba E y $300\mu s$ para la prueba H.

Tabla 4.2 Pruebas A, B, E y H.

A	B	E	H
0	118	0	23
125	243	250	323
250	368	500	623
375	493	750	923
500	618	0	223
625	743	250	523
750	868	500	823
875	993	750	123
0	118	0	423
125	243	250	723
250	368	500	23
375	493	750	323
500	618	0	623
625	743	250	923
750	868	500	223

Con frecuencias de 1KHz y menores, como el tiempo de procesamiento de los embebidos de adquisición era el mismo, se obtuvo y transmitió las marcas de tiempo completas (*UNIX_3*, *UNIX_2*, *UNIX_1, UNIX_0*, *MS_1*, *MS_0*, *US_1*, *US_0*). En estas pruebas se dejó el experimento adquiriendo por más tiempo que en las pruebas anteriores.

Un extracto de las marcas obtenidas el día 29 de Mayo a las 22:55 UTC en la prueba I se muestra decodificado en la Tabla 4.3. Esta es la última prueba tipo *i* en la que se usa la señal TR del CR (excepto en la prueba P) pues en las siguientes se usa la señal TXA. Obsérvese que la marca de microsegundos es $0\mu s$.

Un extracto de las marcas obtenidas el día 30 de Mayo a las 16:39 UTC en la prueba J se muestra decodificado en la Tabla 4.4. En esta prueba y en las siguientes (excepto en las prueba N y P) se pueden ver los $12\mu s$ de desfase entre las señales TR y TXA.

Un extracto de las marcas obtenidas el día 22 de Mayo a las 13:38 UTC en la

Tabla 4.3 Prueba I.

UNIX	ms	us
1527634542	993	0
1527634542	994	0
1527634542	995	0
1527634542	996	0
1527634542	997	0
1527634542	998	0
1527634542	999	0
1527634543	0	0
1527634543	1	0
1527634543	2	0
1527634543	3	0
1527634543	4	0
1527634543	5	0
1527634543	6	0

Tabla 4.4 Prueba J.

UNIX	ms	us
1527698396	473	12
1527698396	474	12
1527698396	475	12
1527698396	476	12
1527698396	477	12
1527698396	478	12
1527698396	479	12
1527698396	480	12
1527698396	481	12
1527698396	482	12
1527698396	483	12
1527698396	484	12
1527698396	485	12
1527698396	486	12

prueba N se muestra decodificado en la Tabla 4.5. En esta prueba se ve $26\mu s$ en la marca de microsegundos pues el CR no esta activado para sincronismo con la señal PPS. Las marcas de milisegundos aumentan en 2ms en concordancia con la frecuencia de 500Hz.

Un extracto de las marcas obtenidas el día 28 de Mayo a las 22:50 UTC en la prueba P se muestra decodificado en la Tabla 4.6. Las marcas en microsegundos son $0\mu s$ pues se cronometró la señal TR en sincronismo PPS, las marcas en milisegundos aumentan en 2ms de acuerdo a la frecuencia de 500Hz.

Tabla 4.5 Prueba N.

UNIX	ms	us
1526996329	986	26
1526996329	988	26
1526996329	990	26
1526996329	992	26
1526996329	994	26
1526996329	996	26
1526996329	998	26
1526996330	0	26
1526996330	2	26
1526996330	4	26
1526996330	6	26
1526996330	8	26
1526996330	10	26
1526996330	12	26

Tabla 4.6 Prueba P.

UNIX	ms	us
1527547815	456	0
1527547815	458	0
1527547815	460	0
1527547815	462	0
1527547815	464	0
1527547815	466	0
1527547815	468	0
1527547815	470	0
1527547815	472	0
1527547815	474	0
1527547815	476	0
1527547815	478	0
1527547815	480	0
1527547815	482	0

Un extracto de las marcas obtenidas el día 18 de Mayo a las 22:42 UTC en la prueba Q se muestra decodificado en la Tabla 4.7. Se puede notar que las marcas de milisegundos aumentan exactamente en 50ms que corresponde a la frecuencia de 20Hz.

Los errores encontrados en las marcas de tiempo adquiridas (formato hexadecimal) se muestran a continuación en tablas:

En la prueba C (ver Tabla 4.8) hay un error donde se transmiten 2 marcas de microsegundos juntas: 0x7D y 0x1F4.

Tabla 4.7 Prueba Q.

UNIX	ms	us
1526683350	600	12
1526683350	650	12
1526683350	700	12
1526683350	750	12
1526683350	800	12
1526683350	850	12
1526683350	900	12
1526683350	950	12
1526683351	0	12
1526683351	50	12
1526683351	100	12
1526683351	150	12
1526683351	200	12
1526683351	250	12

Tabla 4.8 Error en prueba C.

Error en prueba C	
HEX	**DEC**
1F4	500
271	625
2EE	750
36B	875
0	0
7D1F4	**-**
271	675
2EE	750
36B	875
0	0
7D	125
FA	250
177	375
1F4	500

En la prueba G (ver Tabla 4.9) hay un error de un salto de 900μs.

En la prueba J (ver Tabla 4.10) hay un error por una separación de 517ms.

En la prueba K (ver Tabla 4.11) hay un error porque hay una separación de más de 300ms entre dos marcas y estas además se mezclan y son enviadas en una sola transmisión USB.

Tabla 4.9 Error en prueba G.

Error en prueba G	
HEX	**DEC**
13	19
13F	319
26B	619
397	919
DB	219
207	519
333	819
77	119
1A3	419
2CF	719
13	**19**
397	**919**
DB	219
207	519
333	819
77	119
1A3	419
2CF	719

Tabla 4.10 Error en prueba J.

Error en prueba J	
HEX	**DEC**
5B0ED62E, 2D2, C	1527698990, 722, 12
5B0ED62E, 2D3, C	1527698990, 723, 12
5B0ED62E, 2D4, C	1527698990, 724, 12
5B0ED62E, 2D5, C	**1527698990, 725, 12**
5B0ED62F, F2, C	**1527698991, 242, 12**
5B0ED62F, F3, C	1527698991, 243, 12
5B0ED62F, F4, C	1527698991, 244, 12

Tabla 4.11 Error en prueba K.

Error en prueba K	
HEX	**DEC**
5B16C4A2, E5, C	1528218786, 229, 12
5B16C4A2, E6, C	1528218786, 230, 12
5B16C4A2, E7, C	1528218786, 231, 12
5B16C4A2, E8, C	1528218786, 232, 12
5B16C4A25B16C4A2, 23D, C	**-**
5B16C4A2, 23E, C	1528218786, 574, 12
5B16C4A2, 23F, C	1528218786, 575, 12

En la prueba P (ver Tabla 4.12) hay un error donde la computadora recibe 5 datos en lugar de 3 y además hay una separación de 226ms en lugar de los 2ms normales.

Tabla 4.12 Error en prueba P.

Error en prueba P	
HEX	**DEC**
5B0C8C1B, 320, 0	1527548955, 800, 0
5B0C8C1B, 322, 0	1527548955, 802, 0
5B0C8C1B, 324, 0	1527548955, 804, 0
5B0C8C1B, 326, 5B0C8C1C, 20, 0	**-**
5B0C8C1C, 22, 0	1527548956, 34, 0
5B0C8C1C, 24, 0	1527548956, 36, 0
5B0C8C1C, 26, 0	1527548956, 38, 0

Se determinó que estos errores se debieron a fallas en el sistema operativo de la computadora que probablemente se colgaba al estar ejecutando otros procesos y el buffer de recepción USB se sobreescribía con la data que seguía recibiendo, es por eso que se ven saltos de tiempo que no deberían estar e irregularidades en el formato de la data.

4.2 Pruebas tipo *ii* y *iii*

En las pruebas tipo *ii* y *iii* se comparó los datos adquiridos por dos GCTS completamente independientes: A y B, una prueba similar se realizó en [3] (descrita en la sección 7), se usó dos tarjetas FPGA, dos receptores Thunderbolt E, dos elevadores de frecuencia, y dos embebidos de excitación, se pueden ver un diagrama de las conexiones en la Figura 4.3 y los equipos en la Figura 4.4. Estos equipos trabajaron de la siguiente manera:

- Los receptores GPS proporcionaron 2 relojes de referencia de 10MHz, que fueron elevadas a una señal de reloj de 60 MHz (*CLK*) apta para los FPGA de dos maneras:
  - un generador de señales tomó los 10MHz, los multiplicó por 6 y pasó esta señal al CR el cual generó la señal de reloj. (Sistema A),
  - un generador de funciones tomó los 10MHz y generó la señal de reloj. (Sistema B)
- Los GCTS funcionaron a partir de: las señales *CLK*, los datos obtenidos de los receptores ThunderBolt E y sus señales *PPS*. Ambos fueron excitados en su entrada

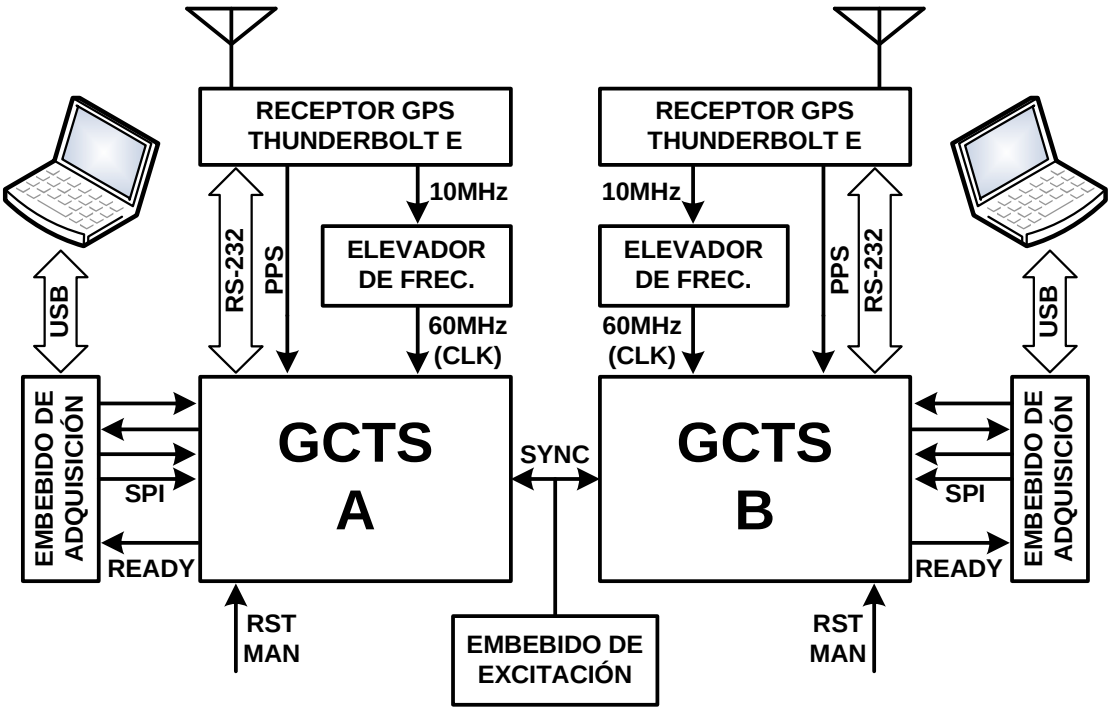


Figura 4.3 Diagrama de conexiones para pruebas tipo II.

Figura 4.4 Equipos usados en las pruebas tipo II y III.

SYNC por una señal periódica de un Arduino Pro Mini, salvo la última prueba en que se utilizó la señal TXA del CR sin sincronismo PPS, luego de recibir la señal de excitación, si los sistemas ya había adquirido la data información de tiempo necesaria, activaron su señal *READY* por $1\mu s$ repitiendo esta acción por cada pulso en la entrada *SYNC*.

- Los embebidos de adquisición configuraron los parámetros de operación del GCTS

mediante un bus SPI, después tomaron la señal *READY* como un pin de interrupción que daba inicio a una transacción para extraer las marcas de tiempo en las pruebas tipo *ii*, y las marcas de milisegundos, microsegundos y el registro *PULSES* en las pruebas tipo *iii*, para luego enviarlas en formato hexadecimal como texto a través de USB a dos computadoras para que almacenen los datos.

- Finalmente, con un script de Python se evaluó la diferencia entre la información obtenida por los GCTS A y B.

El GCTS A fue implementado en la tarjeta X-SP6-X9, su embebido de adquisición fue el Arduino Zero y se usó el generador de funciones para su señal de reloj, por su parte el GCTS B fue implementado en la tarjeta ZedBoard, su embebido de adquisición fue el Teensy 3.2 y se usó el generador de señales y el CR para su señal de reloj.

En las Tablas 4.13 y 4.14 se detallan las configuraciones y resultados de las 5 pruebas realizadas (desde la letra V hasta la Z), ambos embebidos usaron su bus SPI a una velocidad de 3MHz y para las conexiones desde la salida del Arduino Pro Mini (o CR) hasta las tarjetas FPGA se usó una "T" y cables de la misma longitud. Las columnas de las tablas representan lo siguiente:

- "Frec. señal" indica la frecuencia de la señal *SYNC*.
- "Señal" indica si la señal viene del Arduino Pro Mini o del CR.
- "Marcas extraídas" indica que marcas y registros se obtuvieron.
- "Tiempo (hh:mm)" indica la duración de la prueba en horas y minutos.
- "Diferencias de $\pm 1 \mu s$" indica el porcentaje de diferencias de $\pm 1 \mu s$ entre marcas (correspondientes al mismo instante) extraídas de los sistemas.
- "Errores" indica el número de errores encontrados, en la Tabla 4.14 es expresado en ppm.

Solo en la Tabla 4.14:

- "Diferencias de pulsos" indica el porcentaje de marcas de tiempo obtenidas correctamente que difieren en el número de pulsos contados por el registro *PULSES*.
- "Dif. pulsos (min/máx)" indica la menor y mayor diferencia entre la cuenta de pulsos de ambos sistemas.

Tabla 4.13 Pruebas tipo *ii*.

Prueba	Frec. señal	Señal	Marcas extraídas	Tiempo (hh:mm)	Diferencias de ±1us	Errores
V	50Hz	Pro Mini	UNIX/ms/us	00:06	4.6 %	0
W	50Hz	Pro Mini	UNIX/ms/us	02:31	1.4 %	0

Tabla 4.14 Pruebas tipo *iii*.

Prueba	Frec. señal	Señal	Marcas extraídas	Tiempo (hh:mm)	Diferencias de ±1us	Errores (ppm)	Diferencias de pulsos	Dif. pulsos (mín/máx)
X	50Hz	Pro Mini	ms/us/c	13:58	3.1 %	110.5	95.9 %	1 / 4
Y	50Hz	Pro Mini	ms/us/c	00:53	2.6 %	0.0	97.4 %	1 / 3
Z	500Hz	CR	ms/us/c	01:36	0.0 %	2.8	99.6 %	1 / 3

En las pruebas tipo *ii* no se encontraron errores, probablemente por el poco tiempo de duración de las pruebas.

Un extracto de las marcas obtenidas el día 7 de Junio a las 22:47 UTC en la prueba V se muestra decodificado en la Tabla 4.15. Se puede notar que las marcas de tiempo aumentan en 20ms aproximadamente correspondiendo con la frecuencia de 50Hz de la señal del Arduino Pro Mini, también se puede ver una diferencia de $1\mu s$ (803 y 802) que se debe a los defases en las cuentas del registro *PULSES*. Esto se demuestra en las pruebas tipo *iii*.

Tabla 4.15 Prueba V.

GCTS A			GCTS B		
UNIX	**ms**	**us**	**UNIX**	**ms**	**us**
1528411628	310	275	1528411628	310	275
1528411628	330	333	1528411628	330	333
1528411628	350	392	1528411628	350	392
1528411628	370	451	1528411628	370	451
1528411628	390	509	1528411628	390	509
1528411628	410	568	1528411628	410	568
1528411628	430	627	1528411628	430	627
1528411628	450	685	1528411628	450	685
1528411628	470	744	1528411628	470	744
1528411628	**490**	**803**	**1528411628**	**490**	**802**
1528411628	510	861	1528411628	510	861
1528411628	530	920	1528411628	530	920
1528411628	550	979	1528411628	550	979
1528411628	571	37	1528411628	571	37
1528411628	591	96	1528411628	591	96

En las pruebas tipo *iii* hubieron errores en 2 de las 3 pruebas.

Un extracto de las marcas obtenidas en la prueba X se muestra decodificado en la Tabla 4.16. Se puede notar que las marcas de tiempo aumentan en 20ms aproximadamente y que las diferencias de $1\mu s$ son causadas por que los contadores de los registros *PULSES* se han desfasado 2 pulsos de reloj. En el resto de datos, las únicas diferencias son el desfase de los pulsos que varía entre 2 y 3.

Un extracto de las marcas obtenidas en la prueba Z se muestra decodificado en la Tabla 4.17. Las marcas de tiempo aumentan cada 2ms, las única diferencias se da en el registro *PULSES* el cual se ha desfasado 2 pulsos de reloj.

La mayoría de los errores encontrados en este tipo de pruebas se dieron como se

Tabla 4.16 Prueba X.

GCTS A			GCTS B		
ms	**us**	**pulsos**	**ms**	**us**	**pulsos**
254	890	21	254	890	19
274	948	49	274	948	47
295	6	55	295	6	53
315	**65**	**0**	**315**	**64**	**58**
335	123	21	335	123	19
355	181	27	355	181	25
375	239	40	375	239	38
395	298	16	395	298	14
415	356	7	415	356	4
435	414	27	435	414	25
455	472	56	455	472	53
475	**531**	**1**	**475**	**530**	**59**
495	589	22	495	589	20
515	647	13	515	647	10
535	705	41	535	705	39
555	763	47	555	763	44
575	822	7	575	822	5

Tabla 4.17 Prueba Z.

GCTS A			GCTS B		
ms	**us**	**pulsos**	**ms**	**us**	**pulsos**
141	843	20	141	843	18
143	843	20	143	843	18
145	843	20	145	843	18
147	843	20	147	843	18
149	843	20	149	843	18
151	843	20	151	843	18
153	843	20	153	843	18
155	843	20	155	843	18
157	843	20	157	843	18
159	843	20	159	843	18
161	843	20	161	843	18
163	843	20	163	843	18
165	843	20	165	843	18
167	843	20	167	843	18
169	843	20	169	843	18

aprecia en la Tabla 4.18, en este caso se puede ver que en el adquisición de la sexta marca en el sistema GCTS B se debía extraer 0x18B en la marca UNIX pero en su lugar se obtuvo 0x116 debido a un desplazamiento del bit MSB hacia la izquierda.

Tabla 4.18 Error en prueba Z.

DECIMAL						HEXADECIMAL					
GCTS A			GCTS B			GCTS A			GCTS B		
385	843	19	385	843	18	181	34B	13	181	34B	12
387	843	19	387	843	18	183	34B	13	183	34B	12
389	843	19	389	843	18	185	34B	13	185	34B	12
391	843	19	391	843	18	187	34B	13	187	34B	12
393	843	19	393	843	18	189	34B	13	189	34B	12
395	843	19	**278**	6169	1	**18B**	34B	13	**116**	1819	1
397	843	19	397	843	18	18D	34B	13	18D	34B	12
397	843	19	399	843	18	18F	34B	13	18F	34B	12
397	843	19	401	843	18	191	34B	13	191	34B	12
397	843	19	403	843	18	193	34B	13	193	34B	12
385	843	19	385	843	18	195	34B	13	195	34B	12

Lo importante de esto es que el error solo se da en un sistema a la vez, por lo que se dedujo que estos problemas eran externos al módulo hardware desarrollado. Cabe resaltar también que la mayoría de errores ocurrieron en el sistema GCTS B por la lectura SPI con la tarjeta Teensy 3.2.

5 Conclusiones y Recomendaciones

5.1 Conclusiones

- Se diseñó e implementó en un FPGA el sistema GCTS capaz de proporcionar marcas de tiempo UTC con una precisión de hasta 1 periodo de la señal reloj con la que opera, más los $\pm$15ns propios de la precisión del receptor GPS.
- El sistema funcionó correctamente en las pruebas en conjunto con los equipos del ROJ, donde se verificó la confiabilidad en operación continua del sistema y su portabilidad a distintos FPGA.
- En modo de funcionamiento automático, el período mínimo de la señal de excitación a cronometrar no debe ser menor al tiempo que toma la transmisión SPI de las 8 marcas de tiempo.
- Con dos sistemas configurados independientemente a una frecuencia de 60MHz, se observó una diferencia menor a 50ns en respuesta a una misma señal de excitación.
- El módulo hardware GCTS puede adaptarse para uso sin interfaz de comunicación externa como un componente dentro de un sistema de adquisición, donde podría marcar una señal de excitación de hasta un periodo mínimo de dos ciclos de reloj.
- El sistema además de servir para radares también puede ser usado en otras aplicaciones donde se necesite cronometrar una señal de control con referencia a la escala UTC.

5.2 Recomendaciones y trabajos futuros

- Desde el 2012 se ha pospuesto la decisión de continuar usando segundos intercalares pues generan un gran problema en diversas aplicaciones ya que vuelven a la escala UTC una escala no continua, la discusión se ha postergado, por última vez en el 2015, hasta la Conferencia Mundial de Radiocomunicaciones en el 2023. Debido a esto el sistema GCTS podría ser configurado para proporcionar marcas de tiempo en la escala GPS, ya que además de que se usaría menos lógica, el tiempo GPS es una escala continua.
- Tener cuidado al elegir y conectar el embebido que funcionará como maestro SPI y extraerá los datos pues durante las pruebas se detectaron fallos debido a ruido y cross-talk. En las pruebas, el embebido que mejor respondió fue el Arduino Zero, aunque si se hubiera logrado una comunicación USB con la TIVA, probablemente hubiera tenido mejor rendimiento.
- Para usar el GCTS en otras aplicaciones debería establecerse un protocolo de comunicación más robusto a través de la interfaz SPI, ya que la mayoría de problemas se daban por corrimiento u omisión de bits.
- Pese a que las conexiones entre las tarjetas de desarrollo y el resto de hardware fueron hechas mediante cables, el funcionamiento del sistema fue correcto y en la ínfima cantidad de fallos se detectó su origen. Sin embargo, una tarjeta PCB diseñada e implementada específicamente para el GCTS sería ideal.
- Para que el GCTS sea parte del sistema de adquisición de datos de radar del ROJ (JARS) será necesario adaptar y eliminar algunos componentes del sistema, principalmente los que manejan la comunicación externa.

Bibliografía

[1] *Support boundary to configure the Windows Time service for high-accuracy environments*, Article ID 939322, rev. 16, https://docs.microsoft.com/en-us/windows-server/networking/windows-time-service/support-boundary, Microsoft Support, jul. de 2017.

[2] P. Włodarczyk, S. Pustelny, D. Budker y M. Lipinski, «Multi-Channel Data Acquisition System with Absolute Time Synchronization», *Nuclear Instruments and Methods in Physics Research Section A Accelerators Spectrometers Detectors and Associated Equipment*, vol. 763, nov. de 2013.

[3] A. U. Abeysekara, T. N. Ukwatta, D. Edmunds, J. Linnemann, A. Imran, G. Kunde e I. Wisher, «GPS Timing and Control System of the HAWC Detector», *[astro-ph.IM]*, oct. de 2014, arXiv:1410.6681v2, https://arxiv.org/pdf/1410.6681v2.pdf.

[4] *Navstar GPS: User Equipment Introduction*, Public Release Version, U.S. Coast Guard, sep. de 1996.

[5] M. A. Lombardi, L. M. Nelson, A. N. Novick y V. S. Zhang, «Time and Frequency Measurements Using the Global Positioning System», *Cal Lab: The international journal of metrology*, págs. 26-33, jul. de 2001.

[6] J. G. Webster, ed., *The Measurement, Instrumentation and Sensors Handbook*. CRC Press, 1999.

[7] *The International System of Units*, 8.ª ed., Bureau International des Poids et Mesures, 2006.

[8] *Recommendation ITU-R TF.460: Standard-frequency and time-signal emissions*, ver. 6, International Telecommunications Union, 2002.

[9] *Global Positioning System Standard Positioning Service Performance Standard*, 4.ª ed., United States Department of Defense, sep. de 2008.

[10] D. W. Allan, N. Ashby y C. C. Hodge, «The Science of Timekeeping», Hewlett Packard, Application Note 1289, jun. de 1997.

[11] *The Open Group Base Specifications Issue 7*, Issue: 7, IEEE, 2016.

[12] *Application Note 83: Fundamentals of RS–232 Serial Communications*, Dallas Semiconductor, 1998.

[13] P. Bohàčik, «MPC5121e Serial Peripheral Interface (SPI)», Freescale Semiconductor, Application Note AN3904, 2009, rev. 0.

[14] S. Kingsley y S. Quegan, *Understanding Radar Systems*, 1992 reprint. SciTech Publishing, 1999.

[15] M. A. Richards, J. A. Scheer y W. A. Holm, *Principles of Modern Radar. Vol. I: Basic Principles*, 1.ª ed. SciTech Publishing, 2010.

[16] *RADAR*, http://www.allstar.fiu.edu/aero/radar.htm, Allstar Network, mayo de 2017.

[17] B. J. LaMeres, *Introduction to Logic Circuits & Logic Design with VHDL*, 1.ª ed. Springer, 2017.

[18] V. A. Pedroni, *Circuit Desing with VHDL*, 1.ª ed. MIT Press, 2004.

[19] T. L. Floyd, *Fundamentos de sistemas digitales*, 9.ª ed. Pearson Education Inc, 2006.

[20] *Metastability in Altera Devices*, Application Note 42, ver. 4, Altera, 1999.

[21] *ETS 300 462-1: Transmission and Multiplexing (TM); Generic requirements for synchronization networks; Part 1: Definitions and terminology for synchronization networks*, European Telecommunications Standards Institute, 1997.

[22] *ZedBoard (Zynq^{TM} Evaluation and Development) Hardware User's Guide*, ver. 2.2, XILINX, 2014.

[23] *Spartan-6 Family Overview*, ver. 2.0, XILINX, 2015.

[24] *Trimble ThunderBolt E GPS Disciplined Clock User Guide*, ver. 1.0, rev. D, Trimble, 2012.

Presupuesto

El presupuesto para el presente trabajo se puede ver en el Cuadro 1

Tabla 1 Presupuesto.

ACTIVIDAD	Monto(S/.)
BIENES	
Bibliografía	400
Impresiones	100
Encuadernación	200
Útiles oficina	100
Laptop	3000
Equipos electrónicos	7000
SERVICIOS	
Asesoramiento	300
Consultores	300
Técnicos	200
Viajes	1500
Internet	900
TOTAL	14000