Universidad Zaragoza
1542

Trabajo Fin de Grado

Implementación en FPGA del juego de la vida

Game of Life implementation on an FPGA

Autora

Carla Cabrejas Escosa

Directores

Isidro Urriza Parroqué
Luis Ángel Barragán Pérez

Ingeniería Electrónica y Automática

Escuela de Ingeniería y Arquitectura

Año 2022 - 2023

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe remitirse a seceina@unizar.es dentro del plazo de depósito)

D./Dª. Carla Cabrejas Escosa ,

en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de Estudios de la titulación de Grado en Ingeniería Electrónica y Automática (Título del Trabajo)

Implementación en FPGA del Juego de la Vida.

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 05/06/2023

Fdo: CARLA

Tabla de contenidos

Lista de figuras

Lista de tablas

Resumen

En el presente trabajo se aborda la implementación en FPGA (Field-Programmable Gate Array) del Juego de la Vida de Conway, un autómata celular que simula la evolución de células en una cuadrícula bidimensional. El objetivo principal es desarrollar un algoritmo en VHDL y verificar su funcionamiento, comparándolo con un modelo de referencia realizado en Matlab. Una vez que el diseño sea correcto, se representará de una manera gráfica en un panel de LEDs.

Se realizará una revisión de los fundamentos teóricos del Juego de la Vida, incluyendo las reglas de evolución, la representación de la cuadrícula y los patrones característicos.

Se llevará a cabo una explicación de los recursos, software y hardware que han sido utilizados.

Se desarrollará la implementación del algoritmo en Matlab, lo que servirá como modelo de referencia.

Posteriormente, se propondrá una arquitectura para la implementación en FPGA. Se describirán los componentes del sistema, como el módulo que realiza las iteraciones (que contiene el generador de vecinos y el evaluador de reglas), y el controlador de visualización.

Posteriormente, se realizará la síntesis e implementación del diseño propuesto.

Finalmente, se realizarán pruebas del sistema implementado, evaluando su funcionalidad. Por un lado, se comparará mediante ficheros con la implementación de Matlab. Por otro lado, se comprobará visualmente en un panel de LEDs con un modo de funcionamiento denominado “debug”, que permite ejecutar una única iteración y visualizarla al presionar un pulsador.

1. Introducción

El juego de la vida es un autómata celular desarrollado por el matemático John Horton Conway en 1970 [5]. Se trata de un ejemplo clásico de autómata celular cuyas sencillas reglas no lo eximen de un comportamiento sorprendentemente complejo y fascinante. Ha llegado a convertirse en un tema muy estudiado por investigadores en el campo de la computación, para explorar conceptos de sistemas complejos, teoría de autómatas y algoritmos.

En el presente trabajo fin de grado, se va a llevar a cabo el desarrollo, implementación y verificación del juego de la vida de Conway en una FPGA. Se trata de un dispositivo programable que permite una gran flexibilidad y adaptación a la hora de implementar sistemas digitales, cuyo uso ha sido impulsado por los avances en la electrónica digital, lo que ha llevado al desarrollo de sistemas de alto rendimiento, cada vez más complejos, para los cuales las FPGAs constituyen una herramienta fundamental. Además, su capacidad de procesamiento paralelo y en tiempo real hace que sea la opción perfecta para estudiar el comportamiento de juegos y autómatas celulares.

A lo largo de este capítulo se van a tratar temas relacionados con los objetivos y alcance del trabajo, con la utilidad real y actual del juego de la vida de Conway, así como con la explicación de términos y conceptos importantes para el desarrollo del trabajo fin de grado, y con el porqué de la elección de la FPGA para su implementación.

1.1. Objetivos y alcance

El objetivo principal de este trabajo es diseñar, implementar, y verificar el juego de la vida de Conway en una FPGA, para poder estudiar su evolución y funcionamiento.

El proyecto se llevará a cabo utilizando el lenguaje de descripción hardware VHDL, que será desarrollado en el entorno de trabajo Vivado Design Suite de Xilinx, capaz de describir su funcionamiento mediante circuitos digitales.

Se pretende conocer más acerca del algoritmo creado para simular el juego de la vida en una FPGA, al tratarse de un dispositivo que presenta un gran paralelismo a la estructura del juego.

Una vez que el desarrollo en VHDL se finalice, se comparará su funcionamiento con los resultados obtenidos gracias al modelo de referencia hecho en Matlab.

También se realizará la interfaz para la comunicación con un panel de 8 por 8 LEDs sobre el que se mostrará la evolución de las celdas en cada iteración.

Asimismo, se comentarán los tiempos de propagación de las señales y utilización de recursos.

Este proyecto puede servir como base para futuras investigaciones de autómatas celulares y otros algoritmos complejos, así como para explorar otras técnicas de implementación.

En la Figura 1 se muestra el diagrama de Gantt de las distintas tareas llevadas a cabo durante el desarrollo de este proyecto. Se representan las semanas de trabajo desde el 31 de enero hasta el 4 de junio.

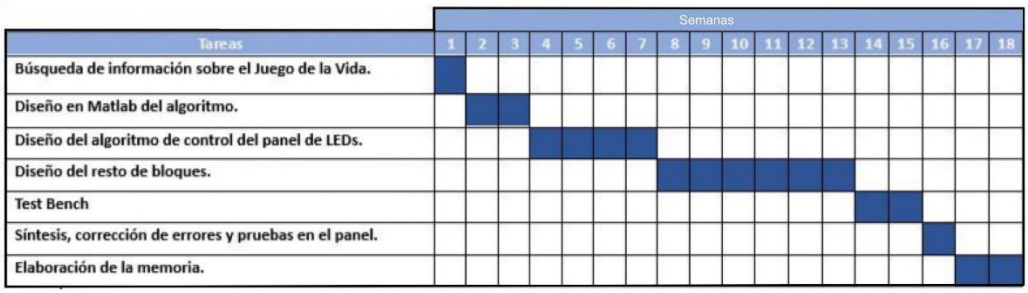


Figura 1. Diagrama de Gantt

1.2. Descripción del algoritmo

1.2.1. Autómata celular [16]

Un autómata celular es un modelo matemático y computacional, que evoluciona de forma discreta. Consiste en una cuadrícula bidimensional o tridimensional, compuesta por un conjunto de celdas interconectadas que adquieren distintos valores o estados.

Su nombre se debe a la similitud con el crecimiento de las células, de esta forma, las celdas del autómata evolucionan según una expresión matemática, dependiente del estado de las celdas en el estado anterior y del estado de sus vecinas.
El estado o valor de una celda puede ser discreto o continuo, tomando valores como ‘1’ o ‘0’ si es discreto, o números reales si es continuo.

En cada iteración, se aplica la expresión matemática que constituye una regla de actualización y el sistema evoluciona generando patrones y estructuras complejas.

Estos autómatas se utilizan en diversos campos como la computación, la biología o la física, ya que se trata de un concepto simple que es capaz de modelar fenómenos muy complejos, relacionando, en algunos casos, sistemas naturales con sistemas artificiales (como es el caso del crecimiento de las células humanas).

Algunos ejemplos de utilización de autómatas celulares son: el estudio de poblaciones biológicas, para simular la propagación y muerte de las células. La simulación de máquinas Turing completas. La generación de gráficos por computadora. Incluso la simulación de la dinámica social y los patrones de comportamiento en comunidades humanas

En cuanto al ámbito que nos ocupa, el de la computación, los autómatas celulares son grandes herramientas para la simulación de sistemas con diseño modular, ya que ambos conceptos se basan en la idea de dividir un sistema complejo en componentes más simples y definir reglas locales para su interacción.

1.2.2. El juego de la vida

El juego de la vida es un ejemplo de autómata celular. Se trata de un juego con reglas muy simples, pero que ha llamado la atención de muchos científicos debido a su complejo comportamiento [6].
Consiste en una cuadrícula bidimensional, con simetría cilíndrica, formada por celdas que pueden estar vivas o muertas. El tamaño de la matriz en la que se desarrolla el juego es variable y depende de los recursos de los que se disponga, pudiendo llegar a desarrollarse sobre una matriz infinita.
En el caso de este trabajo, el panel de LEDs sobre el que se va a representar la matriz de celdas es de 8 por 8, por ello se ha escogido este tamaño para su implementación.
La evolución de este juego depende de un estado inicial y no requiere intervención adicional, cada celda evoluciona según la interacción con sus ocho vecinas adyacentes.
A partir del estado inicial, el tablero evoluciona en pasos discretos de tiempo, cambiando el estado de las celdas (muerta o viva).

Las reglas de evolución son las siguientes:

1- Si una celda muerta tiene 3 celdas vecinas vivas, revivirá en la siguiente iteración (Reproducción).

2- Una celda muere si tiene más de 3 o menos de 2 celdas vecinas vivas (Superpoblación/Subpoblación).

3- Una celda viva seguirá viva en la siguiente iteración si tiene 2 o 3 vecinas vivas (Familia).

Estas reglas determinan cómo evoluciona la cuadrícula en el tiempo. A continuación, se muestran algunos de los patrones más característicos [4]:

- **Extinción**: tras un número finito de iteraciones todas las celdas mueren.

- **Estabilización**: tras un número finito de iteraciones las celdas mantienen el mismo valor u oscilan entre dos estados diferentes. La figura 2 muestra un ejemplo de patrón que mantiene el mismo valor en todas las iteraciones. Mientras que la figura 3 muestra un ejemplo de patrón oscilante.

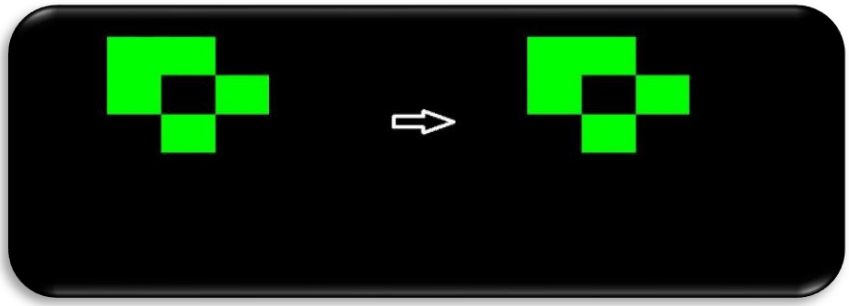


Figura 2. Patrón barco, iteración 1 y 50

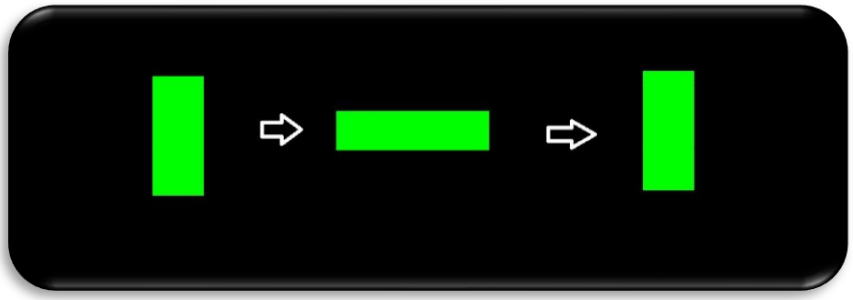


Figura 3. Patrón oscilador, iteraciones 1,6 y 49

- **Crecimiento o variación constante**: el valor de las celdas cambia permanentemente.

En la figura 4 se muestra un ejemplo de crecimiento constante.

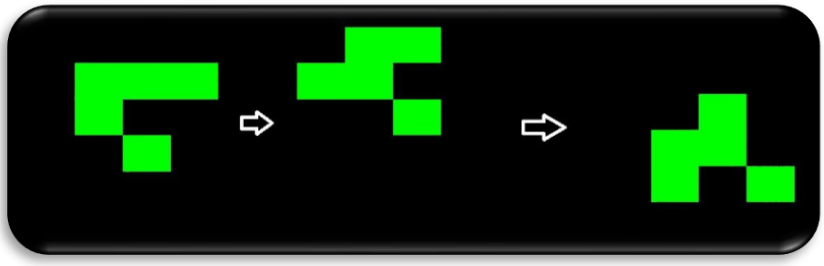


Figura 4. Patrón planeador, iteraciones 1,3 y 25

El juego de la vida de Conway es muy utilizado en la investigación de sistemas complejos y autómatas. Se considera una herramienta muy valiosa en la simulación y modelado de fenómenos naturales, como el estudio de patrones físicos.

En el ámbito que nos ocupa, representa un desafío para programadores. Se pueden estudiar diversas técnicas de diseño e implementación en diferentes dispositivos hardware o lenguajes de programación. Determinar la estructura de datos más adecuada para almacenar y acceder a las celdas, así como realizar operaciones como contar vecinos y actualizar estados, puede representar un desafío técnico.

1.3. Estado del arte

En el ámbito de la electrónica, las implementaciones del juego de la vida han evolucionado con el avance de la tecnología y se han convertido en un gran tema de investigación en el ámbito de la electrónica digital, dado que existen numerosas formas eficientes y optimizadas de implementar el juego de la vida en diversas plataformas.

Un área que se estudia con relación al juego de la vida en la electrónica es su implementación en circuitos integrados, buscando una ejecución rápida y eficiente. Asimismo, la investigación ha llegado al estudio de algoritmos eficientes para la actualización de celdas, así como la compresión de datos para reducir la memoria necesaria para almacenar el estado del juego.

Otro campo en el que el juego de la vida ha resultado ser útil es en seguridad informática, ya que se ha demostrado que este algoritmo es capaz de generar patrones aleatorios que pueden utilizarse en criptografía, por lo que se ha contemplado la posibilidad de integrar el juego de la vida como un componente en sistemas de seguridad.

Se ha diseñado de diferentes formas, tanto en software y hardware especializado, como en un microprocesador con cualquier lenguaje de programación, en plataformas de programación como Matlab, en ASICs (Application-Specific Integrated Circuits), o en FPGAs y GPUs (Graphics Processing Units). Cada tecnología ofrece diferentes niveles de rendimiento y flexibilidad.

En este trabajo va a realizarse la implementación del juego de la vida en Matlab (lo que será el modelo de referencia) y en FPGA (lo que constituye el proyecto principal).
Una FPGA es un complejo circuito integrado digital programable, compuesto por matrices de bloques lógicos configurables y puertos de entrada/salida, como se muestra en la figura 5.

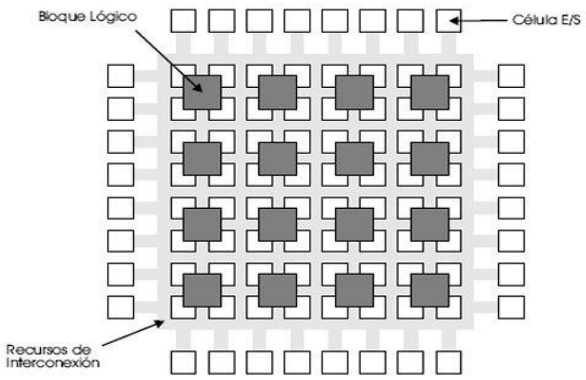


Figura 5. Estructura de una FPGA [17]

La interconexión y funcionalidad de dichos bloques puede ser programada mediante un lenguaje de descripción de hardware especializado.

La elección de este componente para la realización del trabajo se ha basado en cinco ideas importantes:

- Las FPGAs son adecuadas para tareas complejas que requieren altas velocidades de procesamiento y que, además, pueden procesar en paralelo, en contraste con otros métodos de implementación en software que serían secuenciales. El juego de la vida requiere operaciones simultáneas en varias celdas de la cuadrícula, por lo que el paralelismo de la FPGA puede ser beneficioso para realizar estos cálculos.

- Las FPGAs permiten al diseñador programar cualquier funcionalidad conectando diferentes bloques, lo que brinda una gran flexibilidad y, además, son reconfigurables por lo que se pueden adaptar para la implementación del juego de la vida, explorando diferentes reglas o variaciones.

- Además, permiten trabajar con datos de un bit, lo cual es beneficioso en la implementación del juego de la vida, ya que permite almacenar el estado de las celdas (viva o muerta) como un ‘1’ o un ‘0’.

- A nivel educativo, trabajar con una FPGA resulta un desafío interesante, ya que permite conocer más a fondo aspectos de la electrónica digital y la implementación de algoritmos en hardware, campo que está en constante crecimiento hoy en día.

- Como se ha comentado anteriormente, el juego de la vida de Conway ha resultado muy útil en diversas aplicaciones prácticas, por lo que al diseñarlo en una FPGA se pueden explorar estas aplicaciones e incluso descubrir nuevos campos prácticos.

Teniendo en cuenta la descripción anterior, se llega a la conclusión de que este dispositivo puede ser una buena elección.

1.4. Estructura de la memoria

Esta memoria contiene 6 capítulos.

- El capítulo 1 introduce el tema del trabajo fin de grado, así como los objetivos y el tema a tratar en la actualidad.

- El capítulo 2 explica el entorno de desarrollo utilizado, englobando las herramientas hardware y software.

- El capítulo 3 desarrolla la implementación digital llevada a cabo, explicando uno a uno todos los bloques que conforman el proyecto, así como el funcionamiento general del mismo.

- El capítulo 4 explica cómo se ha llevado a cabo la verificación del funcionamiento del algoritmo haciendo uso del modelo de referencia en Matlab y la utilización de simulaciones y ficheros.

- El capítulo 5 hace un análisis de los resultados, así como de los recursos hardware utilizados, verificando que se cumplen las restricciones temporales.

- El capítulo 6 está dedicado a las conclusiones y a posibles mejoras que pueden ser realizadas en el futuro.

2. Entorno de desarrollo

El presente capítulo se centra en la explicación global del entorno de desarrollo elegido para implementar el proyecto, tanto a nivel de hardware, con la descripción detallada de los componentes y dispositivos utilizados, como de software, con la explicación de los programas utilizados.

2.1. Entorno Hardware

En este apartado se describe la placa utilizada para la implementación, así como el panel de LEDs en el cual se ha realizado la representación de la matriz de celdas.

2.1.1. Placa Basys 3

La placa Basys 3 de Digilent [1], es una placa de desarrollo basada en la FPGA Artix-7 de Xilinx. Se utiliza con frecuencia en proyectos de educación y desarrollo, debido a su capacidad de integración y versatilidad.

Se trata de una plataforma de desarrollo de circuitos digitales completa, que puede utilizarse para un gran rango de diseños, desde circuitos combinacionales sencillos hasta circuitos secuencias más complejos como procesadores y controladores integrados. Cuenta con una amplia gama de puertos y periféricos (LEDs, pantallas de siete segmentos, interruptores, botones…), que permiten probar el diseño, y elaborar un gran número de diseños sin necesidad de utilizar hardware adicional. Estos periféricos se muestran en la figura 6.

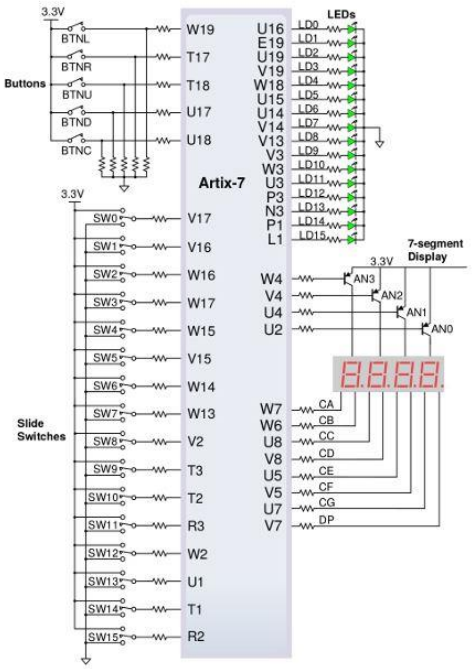


Figura 6. Esquema de conexión de la FPGA

Además, es compatible con diversas herramientas y entornos de desarrollo, y permite la conectividad con otros dispositivos (en el caso que nos ocupa con un panel de LEDs).

2.1.2. FPGA Artix-7

La placa Basys 3 está basada en la FPGA Artix-7 de Xilinx.
Cuenta con 33280 celdas lógicas, cada una de ellas con 4 LUT de entrada y 8 flip-flops, 1800 Kbits de RAM rápida, un conversor analógico-digital y velocidades de reloj superiores a 450 MHz.
Utiliza tecnología de proceso de silicio de 28nm.
En cuanto a sus componentes y características principales, son las siguientes [8]:

- Matriz lógica programable (PL), que consiste en una red de bloques lógicos y conexiones programables

- Memoria programable, que es una combinación de bloques de memoria RAM distribuida y memoria RAM que puede ser configurada.

- Gran variedad de puertos entrada/salida mediante los cuales se reciben o envían señales, estableciendo así comunicaciones con otros dispositivos por medio de algún tipo de interfaz de comunicación.

- Bloques DSP (Procesamiento Digital de Señal), que realizan operaciones matemáticas y algoritmos de procesamiento de señal, así como multiplicación, acumulación y otros cálculos específicos para procesar datos digitales.

- Controladores y periféricos, que permiten gran flexibilidad en el diseño.

- Herramientas de desarrollo proporcionadas por Xilinx como el software Vivado, del cual se hablará más adelante.

2.1.3. Panel WS2812

La visualización de la matriz de celdas se ha llevado a cabo en el panel de LEDs WS2812 [3]. Se trata de un dispositivo de iluminación programable formado por una matriz de LEDs RGB (rojo, verde, azul) con control inteligente, donde el circuito de control y el LED RGB están integrados en el mismo encapsulado conformando un píxel (ver anexo B).

El panel de LEDs utilizado para la representación gráfica es de 64 pixeles (8 por 8), conectados en serie de la forma en que se muestra en la figura 7.

Al estar conectados en serie, la señal de control se transmite de un LED al siguiente.

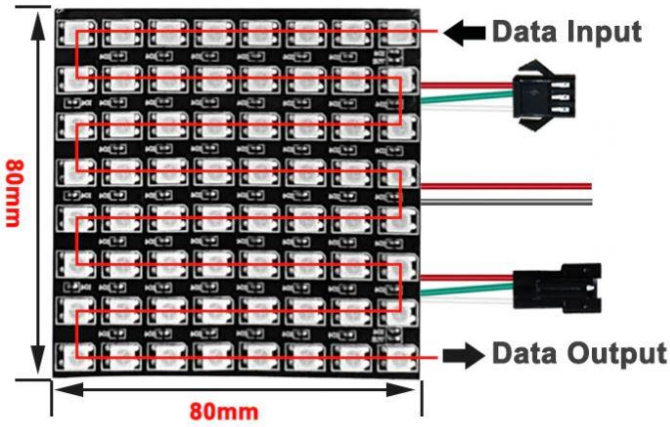


Figura 7. Conexión serie de LEDs

Incluye la interfaz serie de entrada, un registro de almacenamiento, un circuito de salida serie y un oscilador de precisión.

El protocolo de transferencia de datos es el siguiente: cada LED tiene una memoria donde almacena los primeros 3 bytes que recibe, los siguientes bytes recibidos pasarán al segundo LED y así sucesivamente.

Para resetear los LEDs, es necesario mantener la entrada a '0' durante al menos 50 microsegundos.

El control del panel se llevará a cabo en la FPGA, igual que el resto del proyecto, para lo cual es importante tener en cuenta las siguientes consideraciones:

- Para enviar un '1' al panel es necesario mantener la salida a '1' durante T1H y después a '0' durante T1L.

- Para enviar un '0', habrá que mantener la salida a '1' durante T0H y después a '0' durante T0L.

- Para resetear el panel es necesario mantener la salida a 0 durante al menos 50 µs.

Será importante respetar los tiempos de envío de datos para que la comunicación funcione correctamente.

La figura 8 y la tabla 1 muestran las formas de onda que codifican los bits y la activación del reset correspondientes a las consideraciones anteriores.

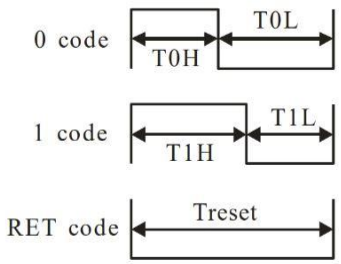


Figura 8. Formas de onda que codifican los bits [13]

Data transfer time (TH+TL=1.25μs±600ns)			Valores utilizados
T0H	0 code ,high voltage time	0.4us ±150ns	**250 ns**
T1H	1 code ,high voltage time	0.8us ±150ns	**900 ns**
T0L	0 code , low voltage time	0.85us ±150ns	**1000 ns**
T1L	1 code ,low voltage time	0.45us ±150ns	**350 ns**
RES	low voltage time Above	50 μs	**50 μs**

Tabla 1. Tiempos que codifican los bits

2.2. Herramientas Software

2.2.1. Vivado

El proyecto se ha desarrollado en el entorno de diseño integrado Vivado.

Se trata de un software producido por Xilinx para la síntesis y el análisis de lenguajes de descripción hardware que incluye características para el desarrollo de SoC y síntesis de alto nivel.
Vivado permite el diseño a nivel RTL (transferencia de registros) mediante el uso de lenguajes de descripción hardware (VHDL o Verilog).

La interfaz de usuario es el navegador de proyectos, que contiene los archivos del diseño, cuyas relaciones son interpretadas por el propio programa.

La figura 9 muestra la estructura del código desarrollado en este trabajo, como se puede observar, existe un fichero principal que engloba a otros llamados bloques o módulos. Esto constituye un diseño modular que será explicado más adelante.

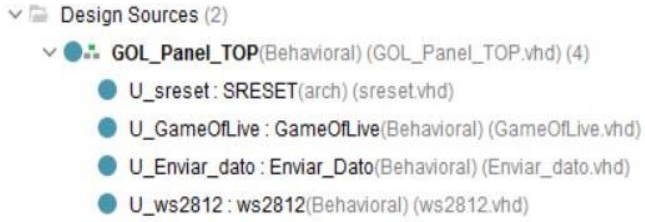


Figura 9. Estructura de ficheros en Vivado

2.2.2. VHDL

VHDL es un acrónimo que proviene de otros dos. VHSIC (Very High Speed Integrated Circuits) y HDL (Hardware Description Language). Se trata de un lenguaje de especificación utilizado para describir circuitos digitales y para la automatización de diseño electrónico utilizando distintos niveles de abstracción [11].

No se trata de un lenguaje de programación, sino de un lenguaje de descripción hardware que permite describir circuitos síncronos y asíncronos y realizar simulaciones, verificaciones y síntesis de diseño. Con este lenguaje, se pueden descubrir problemas en el diseño antes de implementarlos físicamente, y, además, permite que más de una persona trabaje en el mismo proyecto.

Existen tipos de datos (std_logic) que permiten trabajar a nivel de bit y que son los que se han utilizado en la implementación del proyecto.

Como se ha comentado anteriormente, VHDL permite una estructuración modular a la hora de describir sistemas digitales. Ofrece una descripción estructural y de comportamiento mediante la cual es posible decidir cómo se conectan los componentes entre sí y qué función realiza cada uno de ellos.

Como lenguaje de descripción hardware, todos los circuitos trabajan a la vez para obtener el resultado, es decir, todo se ejecuta en paralelo, a diferencia de los lenguajes de programación. Además, existen herramientas que transforman una descripción VHDL en un circuito real (síntesis).

2.2.3. Matlab

Matlab (MATrix LABoratory), es una plataforma de programación y cálculo numérico que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio, que es interpretado y puede ejecutarse tanto en el entorno interactivo, como a través de un fichero de script. Entre sus prestaciones básicas se encuentran la manipulación de matrices, implementación de algoritmos, representación de datos y funciones, creación de interfaces de usuario y la comunicación con programas en otros lenguajes y con otros dispositivos hardware (en el caso de este trabajo con la FPGA).

Además, Matlab dispone de dos herramientas adicionales: Simulink, que es una plataforma de simulación multidominio y Guide, que es un editor de interfaces de usuario. Matlab cuenta también con toolboxes mediante las cuales se pueden ampliar sus capacidades.
En el caso que nos ocupa, se realizará la implementación del juego de la vida en Matlab, para ser utilizada como modelo de referencia debido a la facilidad para representar gráficamente la matriz de celdas y poder comprobar su funcionamiento visualmente.

3. Implementación digital

En este apartado se va a explicar la implementación en Vivado (con la utilización de lenguaje VHDL) y en Matlab del juego de la vida, así como la descripción detallada de los bloques utilizados en el proceso y la comunicación entre ellos.

3.1. Diseño modular

La programación modular está basada en la técnica de diseño descendente, que consiste en dividir el problema original en diversos subproblemas que se pueden resolver por separado, para después recomponer los resultados y obtener la solución general al problema.
Un módulo o bloque es cada una de las partes en que se divide el proyecto final. Cada uno tiene una tarea definida y se puede diseñar y verificar de forma individual antes de ser integrado en el sistema completo, aunque en algunos casos, estos módulos necesitan comunicarse con otros para poder operar.

La modularidad del diseño tiene una serie de ventajas que se van a comentar a continuación:

- Los bloques que conforman el proyecto pueden ser diseñados individualmente y ser utilizados en otros proyectos.

- El diseño modular permite optimizar cada módulo por separado, lo que puede llevar a mejoras en el rendimiento global.

- Dividir un proyecto completo en módulos más simples facilita el desarrollo de la implementación.

- Favorece la depuración y el mantenimiento, ya que, si se produce un error, es más fácilmente localizable y solucionable. Además, las modificaciones en un módulo no afectarán a los demás siempre y cuando la interfaz de comunicación esté bien establecida.

En el diseño de este trabajo, encontramos 4 módulos que se comunican entre ellos mediante interfaz AXI4-Stream (Advanced eXtensible Interface), que será explicada más adelante. Todos los módulos están integrados dentro de un módulo general (GoL_Panel_top), mediante el cual se relacionan sus entradas y salidas. El problema global que se pretende abordar es la realización del juego de la vida, incluyendo además su visualización en el panel de LEDs. Para ello, es necesario, en primer lugar, un bloque SRESET, que se encarga de sincronizar el pulsador de reset (que es asíncrono) y además, de hacerlo

activo en bajo, lo cual es un standard para la interfaz de comunicación AXI4-Stream. El siguiente bloque GameofLife, se encargará de realizar cada iteración del juego de la vida, teniendo como entrada una disposición de la matriz de celdas inicial. En tercer lugar, es necesario un bloque Enviar_dato, que se encargue de organizar la información que devuelve el bloque GameOfLife, de manera que pueda ser enviada de una forma eficiente y sencilla al siguiente y último bloque WS2812, que se encargará de enviar los datos ya procesados de salida al panel de LEDs, teniendo en cuenta la necesidad de respetar los tiempos de envío, para su correcta interpretación. En la figura 10 se muestra un esquema de los bloques utilizados en la implementación.

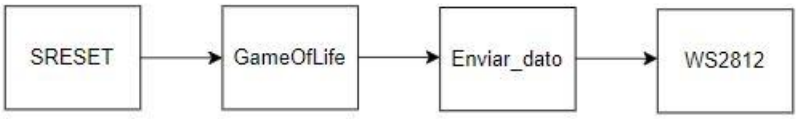


Figura 10. Esquema de bloques

A continuación, se va a detallar el funcionamiento de la interfaz de comunicación utilizada y cada uno de los bloques que componen el proyecto por separado, así como su implementación en Matlab.

3.1.1. Interfaz AXI4

El protocolo AXI [2] es un protocolo de comunicación síncrona entre módulos, muy útil a la hora de realizar transacciones de datos en las que estos se transmiten en paralelo. Está pensado principalmente para comunicaciones on-chip (entre los diferentes componentes integrados en un solo chip), ya que muchos bloques IP de Xilinx utilizan esta interfaz.

Se utiliza un protocolo handshake que funciona con un reloj global y una señal de reset general activa en bajo. En la comunicación mediante esta interfaz, un módulo actúa como maestro (el que envía los datos) y el otro como esclavo (el que recibe los datos). Existen, además, dos señales, VALID y READY, que permiten gestionar las comunicaciones.

Existen 3 tipos de interfaz AXI4:

- Full AXI4: realiza un mapeado en memoria y requiere un canal de direcciones que contiene la dirección del registro o ubicación de memoria a la que se está accediendo en la transacción. Admite operaciones en modo ráfaga (transferencia de múltiples datos consecutivos).
- AXI4-Lite: se trata de una versión simplificada de AXI4 que tiene un bus fijo de 32 bits y no admite operaciones en modo ráfaga.

- AXI4-Stream: no necesita canal de direcciones. Se centra en la transmisión de datos en secuencia continua. Se basa en un canal unidireccional para la transferencia de datos. Esta interfaz es la que se va a utilizar para la comunicación entre bloques en este trabajo y su funcionamiento se va a explicar a continuación.

El protocolo de comunicación (handshake) en la interfaz AXI4-Stream se lleva a cabo de la siguiente forma:
El maestro activa la señal VALID cuando los datos que va a enviar son válidos, indicando así al esclavo que los datos están disponibles para usarse. Por su parte, el esclavo, debe activar la señal READY cuando esté listo para recibir nuevos datos. El intercambio de datos se produce cuando ambas señales (VALID y READY) están activas en un mismo flanco de reloj. El orden en que se activan las señales VALID y READY carece de importancia.
Las figuras 11 y 12 muestran esquemas del funcionamiento del protocolo handshake.

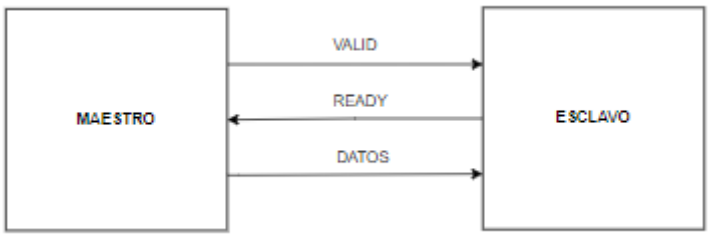


Figura 11. Interfaz AXI4-Stream [18]

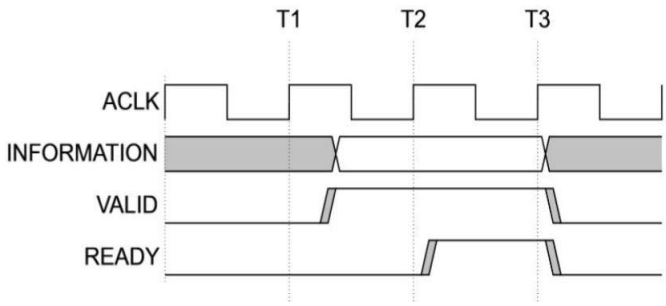


Figura 12. Protocolo AXI4-Stream [18]

3.2. Implementación en Matlab [7]

La implementación del juego de la vida en Matlab ha sido realizada para utilizarse como modelo de referencia a la hora de verificar la funcionalidad del diseño en VHDL.
Esta decisión ha sido tomada debido a que en Matlab se puede representar gráficamente la matriz de celdas en cada iteración y de esta forma comprobar visualmente el funcionamiento del algoritmo.

El código no está estructurado mediante módulos como en VHDL, sino que contiene 3 funciones y un bucle principal que se ejecuta continuamente hasta un número de iteraciones que puede ser modificado. La matriz de celdas, como se ha comentado anteriormente, es bidimensional con simetría cilíndrica, esto quiere decir que las celdas del extremo derecho están conectadas con las del extremo izquierdo y las de arriba con las de abajo, por ello, para la implementación en Matlab, ha sido necesario crear un marco ficticio en el que se replican las filas y columnas de los extremos de la matriz, para, de esta forma, poder sumar correctamente las vecinas vivas que tiene cada celda.

A continuación, se muestra una imagen (figura 13) de la matriz sin marco (borde amarillo), en la que se ha dibujado una disposición de celdas vivas (casillas con circulo amarillo) y muertas (casillas vacías) aleatoria. La matriz exterior (borde negro) constituye la matriz con marco, y en ella se han copiado las filas y columnas exteriores de la forma en que indican las flechas, de forma que los círculos negros representan las celdas vivas y las casillas vacías las celdas muertas en el marco ficticio.

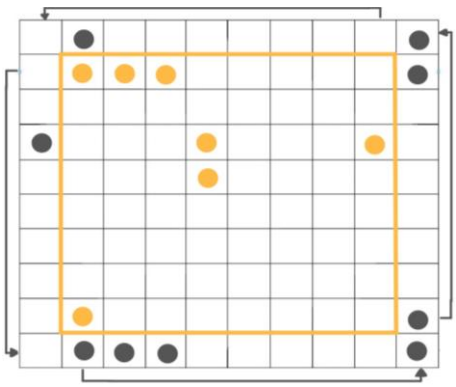


Figura 13. Matriz de celdas con marco

Uno de los puntos clave a la hora de implementarlo es conocer el número de vecinas vivas que tiene cada celda de la matriz. En cuanto a la programación, existen varias formas de llevar esto a cabo. Aquí se va a explicar la utilizada en este proyecto.

Las funciones utilizadas en el código son las siguientes:

- **F_vecinas_vivas**: tiene como argumentos de entrada una matriz 9 por 9 (a la que se ha llamado matriz con marco), y el número de fila (i) y de columna (j) en el que se encuentra la celda de la cual vamos a calcular sus vecinas vivas. En la función, se calcula la posición de la celda al norte, al sur, al este, al oeste, noroeste, noreste, suroeste y sureste de la celda actual y se comprueba el estado de todas las celdas contiguas a la que estamos analizando, con una secuencia de sentencias if, sumando un 1 a una variable X (inicialmente con valor 0) cada vez que una celda contigua está viva. De esta

forma, al comprobar el estado de todas las celdas contiguas, tendremos en la variable X el número de vecinas vivas que tiene la celda (i, j). Esta función devuelve la variable X.

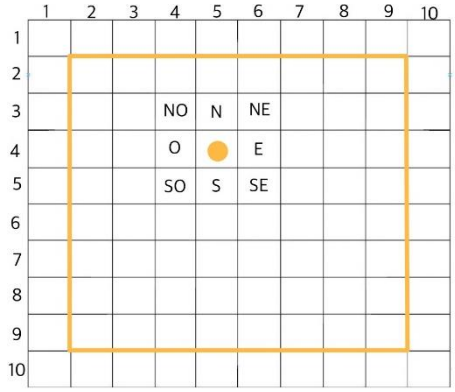


Figura 14. Celdas vecinas

En la figura 14 se observa la celda actual (punto amarillo) y sus vecinas. Los números corresponden a los índices i (vertical/fila) y j (horizontal/columna) de cada celda.

- **F_modificar_marco**: esta función actualiza el marco ficticio de la matriz en cada iteración, ya que el bucle principal recorre únicamente la matriz sin marco, por lo que es necesario actualizarlo en cada iteración. Tiene como argumento de entrada la matriz con marco y devuelve otra matriz igual que la anterior, pero con el marco modificado.

- **F_quitar_marco**: quita el marco de la matriz, dejando la matriz original 8 por 8 para poder representarla gráficamente. Tiene como argumento de entrada una matriz con marco y devuelve la misma matriz, pero sin marco.

En cuanto al bucle principal, en primer lugar, se representa gráficamente la matriz, indicando el número de iteración al que corresponde cada representación, y posteriormente se escribe en un fichero de texto con la función "writematrix", que es propia de Matlab y sirve para escribir matrices en ficheros de texto.
Este fichero se comparará con otro generado en VHDL.
Dentro del bucle principal, existen dos bucles for anidados, que se utilizan para recorrer todas las filas y columnas de la matriz (sin marco). Para cada celda se llama a la función F_vecinas_vivas, que actualiza su estado (o valor) dependiendo de cuantas vecinas vivas tenga y de su valor actual, siguiendo las reglas del juego de la vida.

Por último, se actualiza el valor de la matriz para prepararla para la siguiente iteración del bucle.

El diagrama de flujo del código descrito puede observarse en la figura 15.

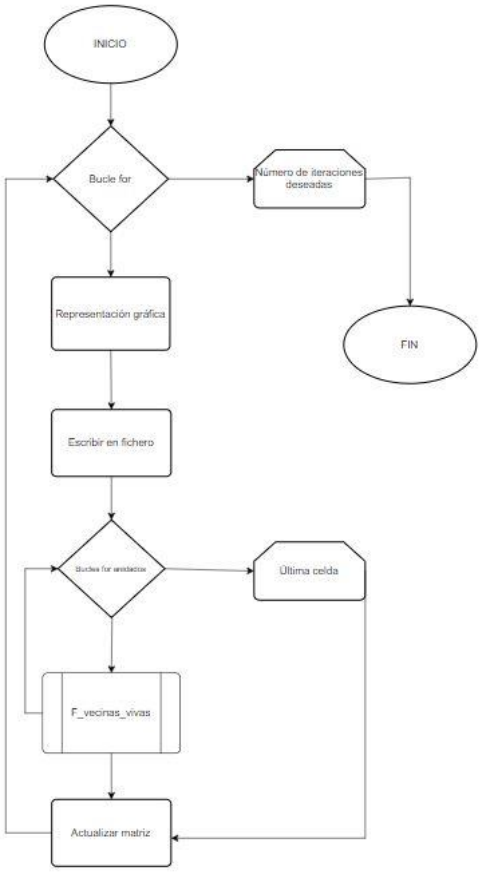


Figura 15. Diagrama de flujo de la implementación en Matlab

En cada ejecución del programa, el fichero de texto se actualiza con los nuevos valores correspondientes a las iteraciones llevadas a cabo en dicha ejecución. Posteriormente, será necesario trasladar ese fichero a un directorio en el que el proyecto VHDL pueda leerlo y compararlo con otro que se generará en la simulación.

3.3. Implementación en FPGA [19]

Como se ha mencionado anteriormente, la implementación en VHDL consta de diferentes módulos que forman parte de otro general. Cada módulo está conectado con el siguiente y se comunican mediante la interfaz AXI4-Stream. En cuanto a la estructura de cada módulo, todos ellos contienen un proceso de flip-flops, que actualiza todos los valores de los biestables del bloque siguiendo el flanco ascendente de la señal de reloj y realiza el reseteo de las señales cuando el reset se encuentra con valor ‘0’ (activo en bajo). Además, en todos los bloques (excepto en el que se encarga de sincronizar el pulsador de reset), existe una máquina de estados que modela su funcionamiento y que a continuación serán representadas y explicadas. Por último, alguno de los módulos contiene otro tipo de procesos o funciones que se explicarán individualmente más adelante.

La figura 16 representa la estructura en la que están organizados los bloques que conforman el proyecto.

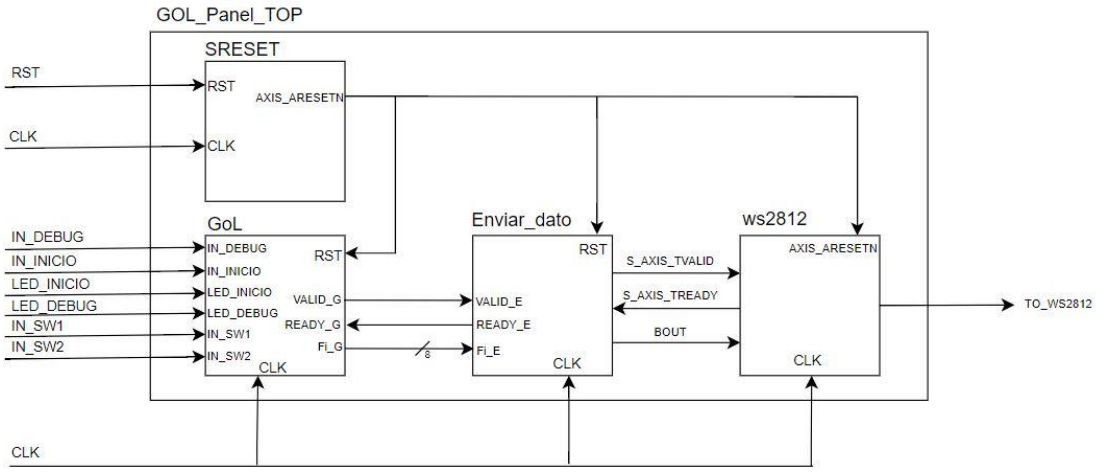


Figura 16. Esquema de los módulos que forman el proyecto

3.3.1. Módulo SRESET

Se trata de un bloque que tiene como señal de entrada el reloj, y reset asíncrono. Este bloque se encarga de sincronizar el reset por medio de dos biestables y, además, lo hace activo en bajo. Como señal de salida, tiene la señal AXIS_ARESETN, que es el reset síncrono y activo en bajo y que constituye la entrada de reset de los demás bloques (ver Anexo A, apartado A1).

3.3.2. Módulo GameOfLife

Este bloque es, como se ha comentado anteriormente, en el que se implementa el juego de la vida de Conway. Realiza las iteraciones a una velocidad que sea perceptible para el ojo humano a la hora de visualizar cada iteración en el panel de LEDs. Envía los datos al bloque Enviar_dato cuando este está preparado y con la señal READY_G en alto (ver Anexo A, apartado A2).

Como **señales de entrada**, tiene las siguientes:

- **IN_DEBUG**: pulsador que inicia el programa en modo debug. Corresponde al pulsador BTNR conectado al pin T17 de la placa (figura x).

- **IN_INICIO**: pulsador que inicia el programa de forma normal, calculando y mostrando cada iteración sin necesidad de intervención. Corresponde al pulsador BTNL conectado al pin W19 de la placa (figura x).

- **LED_INICIO**: LED que se enciende cuando el programa se ha iniciado en modo normal. Corresponde al LED0 conectado al pin U16 de la placa (figura x).

- **LED_DEBUG**: LED que se enciende cuando el programa se ha iniciado en modo debug. Corresponde al LED1 conectado al pin E19 de la placa (figura x).

- **IN_SW1** e **IN_SW2**: interruptores de dos posiciones que permiten seleccionar entre 4 disposiciones iniciales diferentes (oscilador, estático, prueba1, prueba2). Corresponden a los interruptores SW0 y SW1 conectados a los pines V17 y V16 de la placa, respectivamente.

- **CLK**: reloj de la placa (100 MHz)

- **RST**: señal de reset síncrona y activa en bajo.

- **READY_G**: señal que se recibe del esclavo en la comunicación e indica que este está preparado para recibir nuevos datos.

En cuanto a las **señales de salida**:
- **Fi_G**: son 8 vectores de 1 byte y constituyen cada una de las filas de la matriz de celdas.

- **VALID_G**: señal necesaria para la comunicación con el siguiente bloque. Se activa a '1' cuando los datos almacenados en Fi_G son válidos.

Este bloque está formado por 7 biestables: dos sincronizadores (para los pulsadores), dos detectores de flanco, un contador de pulsos de reloj y los correspondientes al estado de la máquina de estados y a la matriz de celdas (que además cuenta con habilitación enable).

La matriz de celdas es un array bidimensional de 8x8 bits.
Al declarar la matriz de esta forma, no es necesario añadirle un marco, ya que al analizar las vecinas de la celda (7)(7) y sumar un 1 a estos valores para obtener las celdas sur y este, el dato se desborda (va de 0 a 7), y se obtiene la celda (0)(0).

Este bloque cuenta con una función (F_VECINAS_VIVAS), que tiene el mismo funcionamiento que la explicada anteriormente en Matlab con la consideración de que en

este caso no existe matriz con marco, sino que se trabaja en todo momento con la matriz normal.

El funcionamiento del módulo es el siguiente: el código está formado por tres procesos que se ejecutan simultáneamente. El primero de ellos es un multiplexor, el cual selecciona la matriz inicial de entre cuatro opciones posibles según el estado de los interruptores. Por otro lado, tenemos el proceso de flip-flops, mediante el cual, los 7 biestables existentes actualizan su valor en cada flanco de subida de reloj, a excepción de la matriz de celdas, que sólo se actualizará cuando una señal de enable esté a '1'. Esta señal se activará según una máquina de estados que se va a explicar a continuación.

Por último, tenemos la máquina de estados que modela el funcionamiento general del programa. Esta máquina de estados está compuesta por 7 estados, uno de reposo, 3 dedicados al inicio normal (inicio, espera, iteración) y otros 3 dedicados al inicio en el modo debug (debug, espera_debug, iteración_debug). (Figura 13)

Al pulsar reset, la máquina de estados se encuentra en el estado reposo, en el que ambos LEDs están encendidos, a la espera de que se elija uno de los dos modos. Si se presiona el pulsador de inicio se pasa al estado inicio y si se pulsa el de debug, se pasa al estado espera_debug.

En el estado inicio, únicamente el LED de inicio normal queda encendido, y en el momento que el bloque Enviar_dato pone la señal ready a '1', la máquina de estados pasa al estado espera. En este estado existe un temporizador, que consiste en un contador de 48000000 ciclos de reloj, para que cada iteración se realice cada medio segundo aproximadamente y de esta forma sea perceptible para el ojo humano a la hora de visualizarla en la placa de LEDs. Una vez que ha pasado este tiempo, se pasa al estado iteración, donde se realiza una iteración del juego de la vida de manera similar a cómo se hacía en Matlab, se calculan las vecinas vivas de cada celda dentro de dos sentencias for-loop anidadas para recorrer las celdas de la matriz, y según su valor se actualiza el valor de cada celda. Automáticamente, una vez finalizado la sentencia for, se pasa al estado inicio, donde VALID_G se pone a '1', ya que los datos están disponibles para enviarlos.

Por otro lado, si escogemos el modo debug, pasamos al estado espera_debug, donde queda encendido únicamente el LED correspondiente a este modo y se espera hasta que se presiona el mismo pulsador, momento en el que se pasa al estado debug. Aquí, pasamos directamente a iteracion_debug donde se realiza lo mismo que en una iteración normal. Finalmente volvemos al estado espera_debug para esperar a que se pulse otra vez el pulsador.

El modo de operación del programa solo puede seleccionarse al principio de este y es necesario resetearlo para seleccionar otro modo.
Las señales de salida Fi_G son actualizadas en cada ciclo de reloj con los valores de las filas de la matriz.
La figura 17 muestra un esquema de la máquina de estados de este bloque.

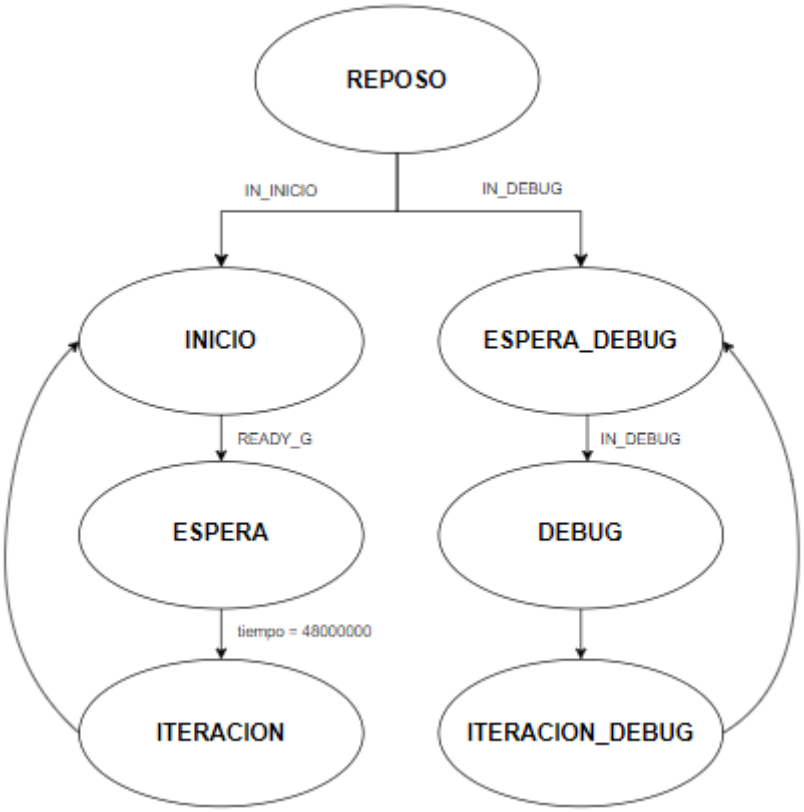


Figura 17. Máquina de estados del bloque GoL

3.3.3. Módulo Enviar_dato

En este bloque se realiza la adecuación de los datos recibidos desde GameOfLife para poder mandarlos al siguiente bloque (ver Anexo A, apartado A3).

Las **entradas** de este módulo son:

- **CLK**: reloj de la placa (100 MHz)

- **RST**: señal de reset síncrona y activa en bajo.

- **VALID_E**: señal necesaria para la comunicación con el bloque anterior. Se activa a '1' cuando los datos almacenados en Fi_G en el módulo GoL son válidos.

- **FI_E**: datos recibidos desde GoL, son 8 bytes que contienen las filas de la matriz de celdas.

- **S_AXIS_TREADY**: señal necesaria para la comunicación con el siguiente bloque. Se activa a '1' cuando el esclavo está listo para recibir datos.

En cuanto a las **señales de salida**, son las siguientes:
- **READY**_E: señal que se activa cuando el bloque está listo para recibir nuevos datos.

- **BOUT**: señal de 8 bits, que corresponde a la codificación de uno de los colores RGB de un LED de la placa, y que posteriormente será enviada bit a bit desde el siguiente bloque a la placa.

- **S_AXIS_TVALID**: señal necesaria para la comunicación con el bloque siguiente, se pone a '1' cuando los datos son válidos en BOUT.

En el este bloque, se leen los datos de las entradas Fi_E, y se guardan en un vector de 64 bits. Por cada bit del vector, se envían 3 bytes cuyos valores dependen de si este bit es un '1' o un '0'. Cuando sea un '0', el valor de los 3 bytes será 0 y el LED estará apagado, cuando el valor sea un '1', el valor será 15 y el LED estará encendido en color blanco ya que los 3 bytes tendrán el mismo valor. Esto se debe a que el panel de LEDs, como se comentó en secciones anteriores, necesita 3 bytes para cada LED, cada uno asociado a un color RGB.

En cuanto a los procesos, existe un proceso de flip-flops, como en el caso anterior, y otro correspondiente a la máquina de estados, que se ejecutan en paralelo.

Enviar_dato está formado por 6 biestables: tres contadores, un temporizador, un registro de almacenamiento, y el relacionado con el estado de la máquina de estados.
Cuenta también con un registro de desplazamiento.

Existe una función llamada DAR_VUELTA_VECTOR que se encarga de cambiar el orden de los elementos de un vector de std_logic_vector para posteriormente unirlos todos en otro vector y mandarlos por orden al panel de LEDs, ya que como se comentó en secciones anteriores, estos datos deben ser enviados de una forma específica.

La máquina de estados está formada por tres estados: inicio, crea_byte y envia_byte. En primer lugar, después de pulsar reset, la máquina de estados se encuentra en el estado inicio. En el momento en que la señal valid se pone a '1', se leen los datos de las entradas Fi_E, y se guardan en el vector de 64 bits, además, se pasa al estado crea_byte, en el que según el valor de dos contadores (cnt_bit y cont_iteraciones) se realizan diferentes

acciones. Si cnt_bit es igual a 64, quiere decir que se han enviado todos los datos de la matriz de celdas y, por lo tanto, se ha terminado de enviar una iteracion completa, el siguiente estado será inicio y se reseteará cnt_bit. Si no se han contado los 64 bits todavía y además cnt_iteraciones es '1' (lo que quiere decir que no nos encontramos en el primer bit a enviar), se activa el enable de un registro del desplazamiento que desplaza una posición a la izquierda el vector de 64 bits. Finalmente, se suma 1 a cnt_bit y se pasa al estado envia_byte. En este estado se activa la señal S_AXIS_TVALID que constituye parte del handshake con el siguiente bloque.

Si todavía no se han enviado los 3 bytes correspondientes a cada bit, el siguiente estado será envia_byte y se sumará uno a cnt_byte. Si ya se han enviado los 3 bytes, pero todavía no han sido recorridos los 64 bits, entonces el siguiente estado será crea_byte y se iniciará el contador de bytes, y si, por último, se han envíado los 3 bytes, y además se han recorrido los 64 bits, entonces se pasará a inicio para estar listo cuando lleguen nuevos datos para enviar. Todas estas condiciones están también condicionadas por el tiempo de transmisión de cada byte, que es de 10 microsegundos, por lo que la máquina de estados se encontrará en el mismo estado hasta que este tiempo haya transucurrido.

La figura 18 muestra un esquema de la máquina de estados de este bloque.

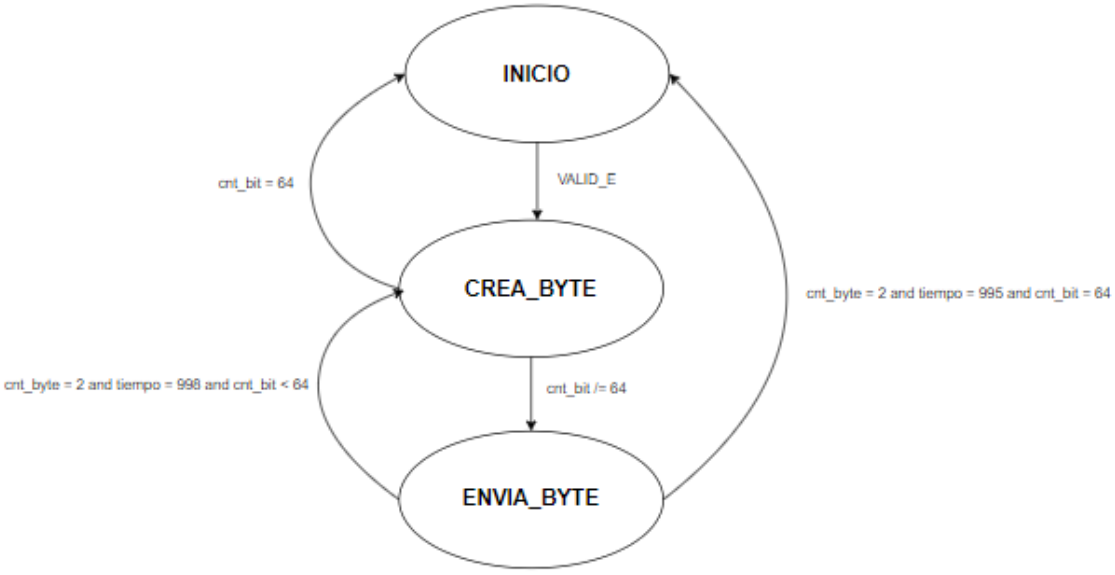


Figura 18. Máquina del bloque Enviar_dato

3.3.4. Módulo WS2812 [13]

Las **señales de entrada** de este bloque son las siguientes:

- **AXIS_ARESETN**: señal de reset síncrono, activa en bajo.

- **AXIS_ACLK**: reloj de la placa (100 MHz)

- **S_AXIS_TDATA**: datos recibidos por el bloque anterior, correspondientes al byte que configura un color RGB perteneciente a un LED de la placa.
- **S_AXIS_TVALID**: señal que permite la comunicación con el maestro y se activa cuando los datos están listos en el bloque Enviar_dato, en la salida BOUT.

En cuanto a las **salidas**:

- **S_AXIS_TREADY**: señal que permite la comunicación con el maestro y se activa cuando el esclavo está listo para recibir nuevos datos.

- **TO_WS2812**: señal de un bit que se envía al panel de LEDs.

El objetivo de este bloque es recibir un byte, e ir enviándolo bit a bit, modificado de tal forma que cumpla con las especificaciones y restricciones temporales del panel de LEDs comentadas en apartados anteriores. La forma de enviar cada bit se representó anteriormente en la figura 6 y tabla 2 (ver Anexo A, apartado A4).

De esta forma, si la señal de entrada es, por ejemplo "00000111", el primer dato a enviar será un '0', para lo cual la señal de salida deberá valer '1' durante 250 nanosegundos y posteriormente, '0' durante 1000 nanosegundos. Con el ejemplo descrito, tendremos que repetir esta operación 5 veces, que corresponden al número de 0. En el momento en que sea necesario enviar un '1', la señal de salida deberá valer '1' durante 900 nanosegundos y '0' durante 350 nanosegundos. Esta operación se repetirá 3 veces. Sumando los tiempos necesarios para enviar un byte siguiendo estas reglas, se obtiene un tiempo total de 10 microsegundos, que es exactamente el tiempo necesario para enviar un byte en el módulo Envia_dato, lo que facilita en gran medida la transmisión.

En este bloque hay dos procesos, uno correspondiente a los flip-flops y otro a la máquina de estados que modela el funcionamiento.
Existen 4 biestables: un contador, un temporizador, un registro de almacenamiento y el correspondiente al estado de la máquina de estados.

El estado en el que se encuentra la MEF tras pulsar reset es reposo, en el que la señal de salida TO_WS2812 vale '0'. En el momento en que la señal valid se activa, se guarda la señal S_AXIS_DATA en un registro de almacenamiento y se pasa al estado enviar1, en el que la señal de salida vale '1' durante el tiempo establecido (contado por un contador de ciclos de reloj) por las características del panel de LEDs, comentadas anteriormente, y que depende de si el dato a enviar es un '0' o un '1'. Finalmente, pasa al estado enviar0, donde similarmente con el estado anterior, la señal de salida vale '0' un tiempo determinado (haciendo uso del mismo temporizador, que se resetea al terminar de enviar un bit). Con un contador que cuenta bits enviados, se decide si el siguiente estado será enviar1 (para enviar

el siguiente bit), cuando todavía no se han enviado los 8 bits, o reposo (para esperar nuevos datos de entrada y la activación de la señal valid), cuando los 8 bits han sido enviados.
La figura 19 muestra un esquema de la máquina de estados de este bloque.

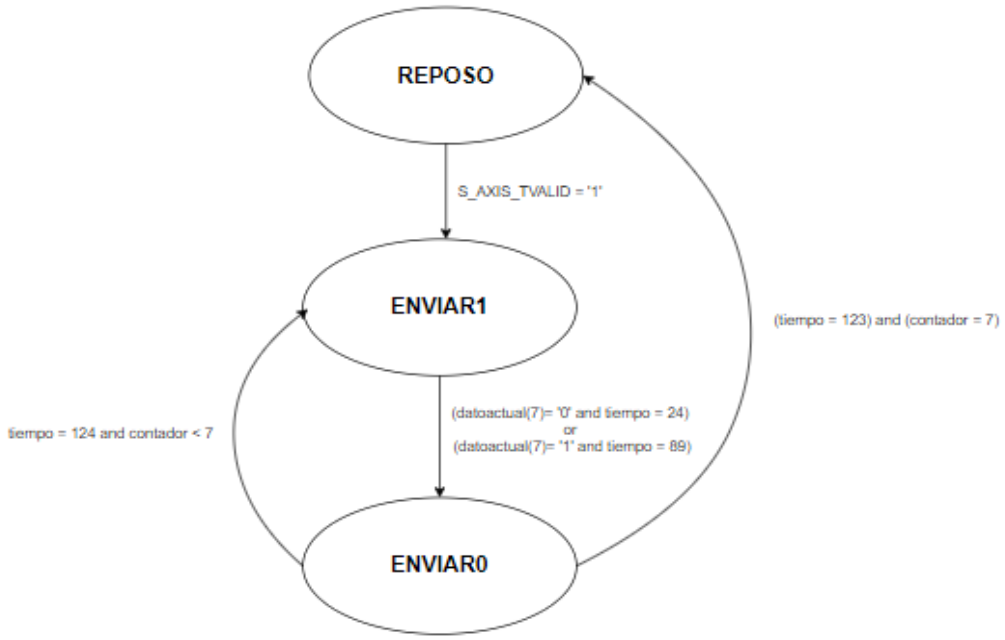


Figura 19. Máquina de estados del bloque WS2812

4. Verificación funcional

Como se ha comentado anteriormente, la verificación del funcionamiento del proyecto en VHDL se va a realizar con ayuda del diseño de Matlab, utilizando este como modelo de referencia. Para ello, se ha creado un fichero a partir de las iteraciones llevadas a cabo en Matlab, que guarda la matriz (sin marco) por filas y una iteración detrás de otra sin ningún tipo de separador.

4.1. Test bench

El test bench es un banco de pruebas diseñado para probar y verificar el comportamiento de un diseño VHDL antes de probarlo en los dispositivos hardware, también es posible aplicar estímulos al diseño a fin de analizar los resultados o comparar los mismos de dos simulaciones diferentes [19].

Tiene como objetivo realizar una simulación del sistema. Durante la simulación, se generan estímulos que servirán de entradas al diseño para verificar si sus salidas coinciden con lo que se esperaba. Esto implica que el banco de pruebas ya no es un circuito, sino que es un programa que se encargará de verificar la descripción del circuito. Es por ello, que en el test bench existen elementos del lenguaje VHDL que son más similares a los que existen en algunos lenguajes de programación, como por ejemplo manejo de archivos y de tiempo, que son los más involucrados en el caso que nos ocupa.

En cuanto a la implementación que se está desarrollando, lo primero es declarar los componentes o bloques que conforman el proyecto y que se han comentado anteriormente, para de esta forma poder dar valor a sus entradas. Por otro lado, una de las cosas que resulta más interesante es conocer el periodo de la señal de salida (TO_WS2812), ya que puede dar alguna pista sobre el correcto funcionamiento del envío de datos a la placa de LEDs. Para obtener el periodo de la señal, se ha creado un proceso, que utiliza la función now, que devuelve el tiempo actual de simulación, con lo cual se puede conocer el periodo de la señal. Además, también se realiza una reconstrucción del bit enviado. Esto es necesario para poder ver de forma más sencilla qué datos se están enviando a la placa, ya que la señal TO_WS2812, no es fácil de interpretar. Por ello, además del periodo, se calculan los tiempos en alto y en bajo de esta señal de salida, con los cuales se puede calcular si lo que se está enviando es un ‘1’ o un ‘0’.
También resulta interesante generar un fichero de texto que contenga la matriz en cada iteración llevada a cabo y que sea exactamente igual que el de Matlab para poder, de esta forma, compararlos y comprobar si se ha cometido algún error en cualquiera de las iteraciones. El lenguaje VHDL posee un manejo de archivos particular [14], más limitado que, por ejemplo, en lenguaje C. La lectura y escritura de datos a un archivo se encuentra orientada a líneas, no a caracteres sueltos, por lo que el mecanismo básico consiste en

formar una línea de texto y después escribirla al archivo. Las funciones involucradas en escribir y leer archivos se encuentran declaradas en la biblioteca std y en el paquete textio. En este TFG se utiliza el paquete std_logic_textio [15], ya que permite trabajar con datos de tipo std_logic y std_logic_vector, que son los que se han utilizado mayoritariamente en el proyecto.

En cuanto a la estructura del test bench, es la siguiente: en primer lugar, se define una señal llamada numIteraciones_des, que representa el número de iteraciones que se desean escribir en el fichero, ya que para compararlo posteriormente con el de Matlab es importante que exista el mismo número de iteraciones.

Después, se abre el fichero y se espera el tiempo necesario para que se lleve a cabo una iteración.
Para escribir cada fila en el fichero, se ha creado una sentencia for para cada una de ellas. Esto se debe a que en Matlab la única forma de escribir la matriz en el archivo es con un delimitador (espacio) entre cada caracter, por lo que es necesario insertar este mismo caracter también en el fichero generado por VHDL.

Por esto, en cada bucle for se recorre una línea insertando un espacio entre cada elemento y posteriormente, cuando se ha recorrido toda la línea, se escribe en el fichero.

La figura 20 muestra un ejemplo de cómo se escriben dos iteraciones de un patrón oscilante en un fichero.

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 1 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

Figura 20. Ejemplo de datos en fichero

Una vez que todas las iteraciones deseadas se han escrito, es necesario cerrar el fichero y volver a abrirlo, ya que no es posible abrir un fichero en modo escritura y lectura al mismo tiempo. Por ello, para compararlo con lo obtenido en Matlab es necesario cerrarlo y abrir ambos ficheros (Matlab y VHDL) en modo lectura. Se recorren ambos ficheros, omitiendo los espacios en blanco y se comparan caracter a caracter. Para conocer si se ha cometido algún fallo, existe una variable fallo que se pone a '1' solo en caso de que dos caracteres de sendos ficheros no coincidan. Además, en el momento en que existe un fallo, se crea otro fichero llamado "estado" que contiene un texto que informa de en qué iteración y en qué línea se ha cometido este error.

La figura 21 muestra el mensaje que se escribe en el fichero si se producen algún error.

```
Se ha producido un fallo en la iteracion  1, linea 2 /// Se ha producido un fallo en la iteracion  2, linea 1 ///
Se ha producido un fallo en la iteracion  2, linea 6 /// Se ha producido un fallo en la iteracion  3, linea 8 ///
```

Figura 21. Mensaje de fallo al comparar ficheros

Si por el contrario, todas las iteraciones son correctas, se deja plasmado en este mismo fichero.

La figura 22 muestra el mensaje que se escribe en el fichero si no se producen ningún error.

```
Todas las iteraciones son correctas
```

Figura 22. Mensajes de iteraciones correctas

Para verificar el funcionamiento se han probado patrones con evoluciones conocidas. En la figura 23 se observa un ejemplo de simulación llevada a cabo con un patrón que se hace estático tras 10 iteraciones. Se puede observar como las señales Fi_G (que corresponden a las filas de la matriz) mantienen su valor tras este número de iteraciones. Así mismo, se muestra la evolución de este mismo patrón en Matlab (figura 24), de forma que se corrobora que esta iteración es correcta, al ser ambos iguales.

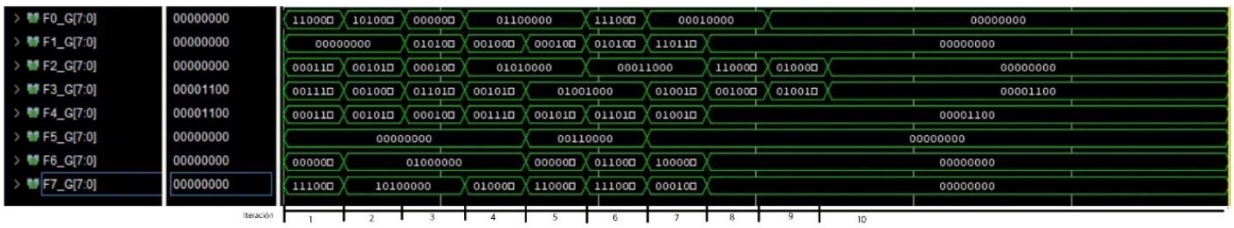


Figura 23. Simulación de patrón que se hace estático en 10 iteraciones en VHDL

Figura 24. Simulación de patrón que se hace estático en 10 iteraciones en Matlab

5. Resultados

Vivado lleva a cabo diversos análisis del diseño durante el proceso de síntesis e implementación, evaluando diferentes aspectos y proporcionando información útil sobre rendimiento, utilización de recursos y cumplimiento de restricciones [10].

En primer lugar, analiza si el diseño es sintetizable (si puede convertirse en una implementación lógica). En este análisis se identifican errores de sintaxis o latches.
También se realiza un análisis de restricciones temporales que identifica violaciones de tiempo y rutas críticas en las cuales se produce el mayor retardo y las cuales determinan la frecuencia máxima de operación.

Además, Vivado lleva a cabo un análisis de recursos utilizados por el diseño, y, por último, un análisis del consumo de potencia.

En este apartado, se van a analizar los resultados obtenidos y las prestaciones. Para ello, se analizarán los datos proporcionados por el software una vez sintetizado e implementado el proyecto (sin ningún error). También se explicarán una serie de conceptos importantes para la interpretación de estos datos.
Al sintetizar e implementar el diseño en Vivado, se obtienen los siguientes resultados en cuanto a la utilización de recursos:

Estos porcentajes proporcionan información sobre cómo se están utilizando los recursos de la FPGA, en relación con los recursos totales disponibles.

Resource	Utilization	Available	Utilization %
LUT	413	20800	1.99
FF	213	41600	0.51
IO	9	106	8.49
BUFG	1	32	3.13

Tabla 2. Recursos utilizados

La tabla 2 refleja la cantidad de recursos utilizados. Se puede observar que el porcentaje total de recursos utilizados sobre el total disponible es reducido. A continuación, se describen cada uno de los recursos:

- LUT (Look Up Table): se trata de bloques de lógica programable que se utilizan para implementar funciones lógicas en el diseño. En el contexto de la lógica combinacional, es la tabla de verdad, que define cómo se comporta la lógica combinatoria. Las FPGA

implementan la lógica combinatoria con LUT's y al configurarla, los bits de la LUT se cargan con unos o ceros.

- FF (Flip-Flop): son biestables que se utilizan para sincronizar la lógica y guardar estados lógicos entre ciclos de reloj. En cada ciclo de reloj, un flip-flop mantiene el valor en su salida.

- IO: Entradas y salidas.

- BUFG (Global Clock Buffer): distribuye señales de reloj de alta distribución (high fan-out clock signals) a través de un dispositivo PLD.

Como se observa en la figura x, el porcentaje de utilización de recursos es reducido, predominando el uso de LUT's.

A continuación, se van a analizar las prestaciones del diseño, para lo cual se ha realizado un análisis temporal del resumen obtenido en Vivado. El resumen de tiempos es una herramienta importante para evaluar si el diseño cumple con las restricciones temporales establecidas y si se alcanzan los objetivos de rendimiento deseados. Antes de analizarlo, es interesante conocer una serie de términos:

- **WNS** (Worst Negative Slack): se trata del peor valor de incumplimiento de los tiempos de establecimiento/recuperación dentro de un dominio de reloj. Este valor puede utilizarse para calcular la frecuencia máxima de reloj a la que funciona el circuito (ecuación (1)).

$$fclk, max\ (MHz)\ = \frac{1}{Tclk\ (ns)\ -\ WNS\ (ns)} \quad (1)$$

- **WHS** (Worst Hold Slack): indica el peor valor de incumplimiento de los tiempos de mantenimiento/eliminación dentro de un dominio de reloj.

- **WPWS** (Worst Pulse Width Slack): medida del peor valor de incumplimiento de los requisitos de periodo mínimo, periodo máximo, tiempo de pulso alto y tiempo de pulso bajo para cada pin de reloj de la instancia.

En el diseño se ha impuesto una restricción temporal en la frecuencia de reloj (señal de temporización periódica para restricciones temporales en diseños síncronos) de 100 MHz.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4,153 ns	Worst Hold Slack (WHS):	0,155 ns	Worst Pulse Width Slack (WPWS):	4,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	546	Total Number of Endpoints:	546	Total Number of Endpoints:	214

Figura 25. Resumen de tiempos

En la figura 25 se observa que la ruta con peor valor de WNS tiene un valor de 4.153 ns. Esto quiere decir que el diseño cumple con las restricciones temporales, ya que este valor es positivo y, además, menor que el periodo de reloj del diseño (10 ns). Para que un diseño funcione correctamente, las señales deben propagarse dentro de un ciclo de reloj.

En cuanto a la frecuencia máxima, utilizando la fórmula anterior se obtiene un valor de 171 MHz, con lo que el diseño alcanzará sin problema la frecuencia establecida (100MHz).
También resulta interesante analizar la potencia que consumirá el diseño. Para ello, se va a analizar los datos que proporciona Vivado.

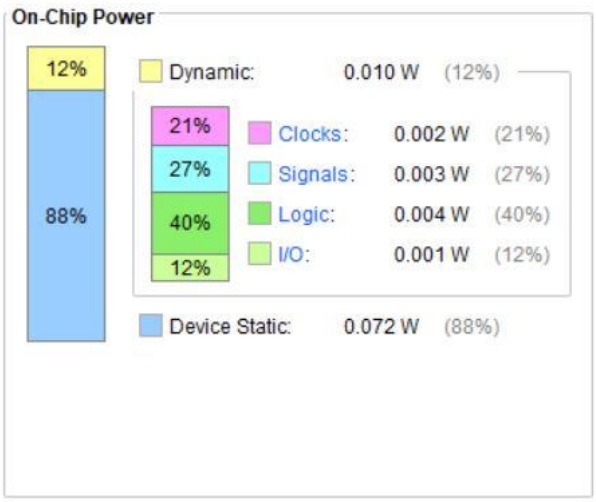


Figura 26. Consumo de potencia

En la figura 26, se observa como la potencia dinámica se encuentra en el 12%, que a su vez se divide en la consumida por el reloj (21%), la consumida por las señales (27%), la consumida por la lógica interna (40%), que es la que abarca mayor porcentaje, y la consumida por los puertos de entrada/salida (12%).
La lógica interna es la que más porcentaje abarca debido a que, como se ha observado antes, el mayor porcentaje de utilización era el de las LUT de la FPGA.

Al analizar estos datos y simular la implementación probando diferentes patrones, el proyecto se ha probado en el banco de pruebas.
Como se observa en la figura 27, dicho banco se compone de la placa Basys 3 (izquierda), conectada al circuito de acondicionamiento (centro), que a su vez está conectado al panel de LEDs (derecha).

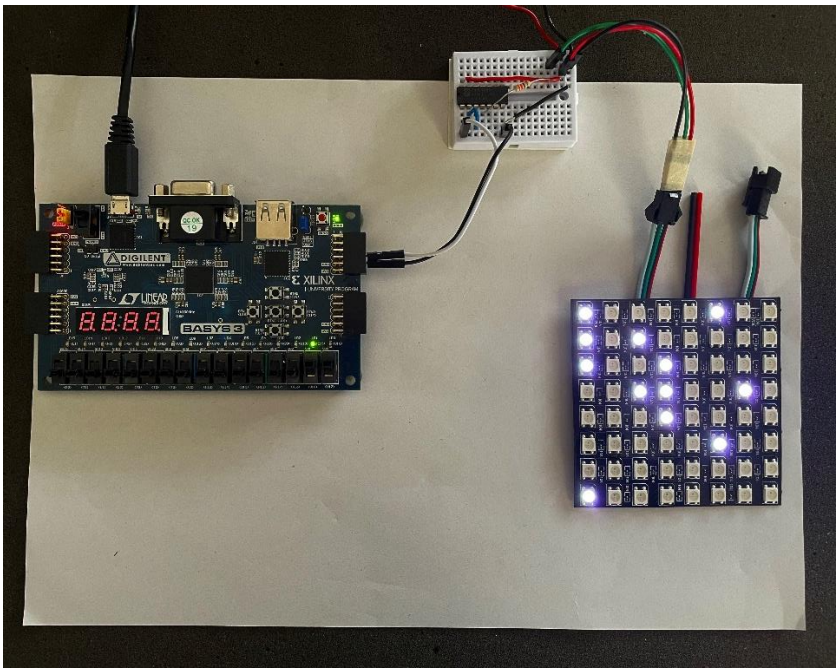


Figura 27. Banco de pruebas

Para la realización de esta prueba se ha utilizado un patrón aleatorio y el modo debug para observar la evolución de la matriz en cada iteración al presionar el pulsador.

La prueba ha sido satisfactoria ya que la representación se visualiza correctamente en el panel.

6. Conclusiones y trabajo futuro

En este apartado, se abordan las conclusiones obtenidas tras elaborar el trabajo fin de grado al completo, así como posibles acciones futuras con las cuales seguir abordando el tema de la implementación del juego de la vida de Conway.

6.1. Conclusiones

Tras la elaboración de este trabajo, cabe destacar que el objetivo propuesto (diseño, implementación y verificación del juego de la vida de Conway en FPGA), ha sido alcanzado satisfactoriamente.

El juego de la vida se trata de un algoritmo en constante evolución para el cuál se siguen descubriendo nuevos usos y variantes hoy en día, por lo que conocer de qué se trata y diferentes formas de implementarlo resulta importante.
Además, mediante la elaboración de este TFG, se corrobora la buena elección que ha supuesto realizar la implementación en una FPGA, ya que se lleva a cabo de forma exitosa y ofrece flexibilidad y capacidad de procesamiento suficientes para ejecutar eficientemente el juego de la vida.

Como se ha comentado varias veces a lo largo del trabajo, existen diferentes versiones del juego de la vida de Conway. Una posibilidad de modificación sería crear un marco de ceros alrededor de la matriz y considerar que todas las celdas laterales están muertas, en lugar de conectar una celda siempre con la siguiente como se ha hecho en este caso. También podrían incluirse nuevas reglas o diferentes patrones fácilmente sin más que modificar el bloque GameOfLife. Cabría la posibilidad de juntar varios paneles de LEDs para tener una matriz más grande y analizar cómo cambia la utilización de recursos en estos supuestos.

6.2. Líneas futuras

Aparte de todas las posibles modificaciones que podrían llevarse a cabo, comentadas en el apartado anterior, una buena idea sería realizar la implementación en HDL Coder [12].
HDL Coder es una herramienta de la empresa MathWorks (empresa desarrolladora de Matlab), que permite realizar diseños de alto nivel mediante la generación de código VHDL portátil y sintetizable a partir de funciones Matlab o Simulink. Esta generación se lleva a cabo mediante síntesis de alto nivel que transforma el diseño de Matlab en VHDL. Utilizar esta herramienta sería conveniente ya que como se ha comentado repetidas veces, se ha utilizado el modelo de Matlab como modelo de referencia, por lo que opera correctamente y ya es un modelo funcional. Podría modificarse para sintetizarlo y visualizarlo en la placa de LEDs.

Otra idea con la que seguir trabajando en este tema sería implementar el juego de la vida en un microcontrolador y observar las limitaciones reales (no solo teóricas) que ofrece en comparación con el diseño en FPGA.
Por otro lado, podría plantearse la utilización de dos relojes, uno más lento para el bloque GameOfLife y otro rápido para llevar a cabo la interfaz con los LEDs.

Bibliografía

[1] Digilent, «Basys 3 FPGA Board Reference Manual», 8 abril 2016.

[2] arm Developer, «AMBA AXI-Stream Protocol Specification», 9 Abril 2021. [En línea]
Disponible: https://developer.arm.com/documentation/ihi0051/b/?lang=en

[3] Worldsemi, «WS2812B Intelligent control LED integrated light source».

[4] Life Lexicon, 2 Julio 2018. [En línea]
Disponible: https://conwaylife.com/ref/lexicon/lex.htm

[5] Wikipedia, «Conway's Game of Life». [En línea]
Disponible: https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

[6] M. Gardner, «Mathematical Games: The fantastic combinations of John Conway's new solitaire game life», Sci. Amer., vol. 223, no. 4, pp. 120–123, 1970.

[7] MathWorks, «Conway Game of Life», 7 Mayo 2012. [En línea]
Disponible: https://es.mathworks.com/matlabcentral/fileexchange/27233-conway-game-of-life

[8] ni, «Las ventajas de los dispositivos FPGA de la serie 7 de Xilinx», 17 mayo 2023. [En línea]
Disponible: https://www.ni.com/es-es/shop/compactrio/what-are-compactrio-controllers/advantages-of-xilinx-7-series-fpga-and-soc-devices.html

[9] W. Ewert, W. Dembski and R. J. Marks, "Algorithmic Specified Complexity in the Game of Life," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 4, pp. 584-594, April 2015.

[10] AMD, «UltraFast Design Methodology Guide for FPGAs and SoCs». [En línea]
Disponible: https://docs.xilinx.com/r/en-US/ug949-vivado-design-methodology/Assessing-Post-Synthesis-Quality-of-Results

[11] H. Kaeslin, «Top-Down Digital VLSI Design. From Architectures to Gate-Level Circuits and FPGAs», Morgan Kaufmann 2015.

[12] MathWorks, «HDL Coder». [En línea]
Disponible: https://es.mathworks.com/products/hdl-coder.html

[13] Electrónica Digital (581-30315), Universidad de Zaragoza, «Control de un panel de LEDs RGB», curso 2021/2022. [PDF]

[14] A. Rushton, «*VHDL forLogicSynthesis*», John Wiley& Sons, 3ª Ed., 2011

[15] V.A. Pedroni, «Circuit Design and Simulation with VHDL», MIT Press, 2ª Ed., 2010.

[16] Fernando Sancho Cparrini, «Autómatas celulares», 30 octubre 2016. [En línea]

Disponible:
http://www.cs.us.es/~fsancho/?e=66#:~:text=Un%20aut%C3%B3mata%20celular%20es%20un,valores%20enteros%20a%20intervalos%20regulares.

[17] MCI, paguayo, «FPGA (Field Programmable Gate Array)», 18 Junio 2019. [En línea]
Disponible: https://cursos.mcielectronics.cl/2019/06/18/fpga-field-programmable-gate-array/

[18] ADM, «AXI4-Stream Interface». [En línea]
Disponible: https://docs.xilinx.com/r/en-US/pg256-sdfec-integrated-block/AXI4-Stream-Interface

[19] W. Kafig, “Howto Builda Self-CheckingTestbench”, Xcell Journal, FirstQuarter2012.

Anexo A

A1. Bloque sreset

```
library ieee ;
  use ieee.std_logic_1164.all ;
  use ieee.numeric_std.all ;

entity SRESET is
 port (
  RST : in  std_logic; -- Reset asincrono y activo en alto
  CLK : in  std_logic; -- Señal de reloj
  AXIS_ARESETN : out std_logic -- Reset sincrono y activo en bajo
  );
end SRESET ;

architecture arch of SRESET is
  signal rst_sinc0, rst_sinc1 : std_logic;
begin

  process( RST,CLK )
  begin
    if (RST='1') then
      rst_sinc0<= '0';
      rst_sinc1<= '0';
    elsif (rising_edge(CLK)) then
    -- Se sincroniza y se invierte la señal de reset
      rst_sinc0 <= '1';
      rst_sinc1 <= rst_sinc0;
    end if ;
  end process;

  AXIS_ARESETN <= rst_sinc1;

end architecture ;
```

A2. Bloque GameOfLife

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity GameOfLive is
   Port ( CLK : in STD_LOGIC;
          RST : in STD_LOGIC;
          VALID_G : out STD_LOGIC;
          READY_G : in STD_LOGIC;
          IN_DEBUG: in STD_LOGIC;
          IN_INICIO: in STD_LOGIC;
          LED_INICIO: out STD_LOGIC;
          LED_DEBUG: out STD_LOGIC;
          IN_SW1: in STD_LOGIC;
          IN_SW2: in STD_LOGIC;
          F0_G : out STD_LOGIC_VECTOR (7 downto 0);
          F1_G : out STD_LOGIC_VECTOR (7 downto 0);
          F2_G : out STD_LOGIC_VECTOR (7 downto 0);
          F3_G : out STD_LOGIC_VECTOR (7 downto 0);
          F4_G : out STD_LOGIC_VECTOR (7 downto 0);
          F5_G : out STD_LOGIC_VECTOR (7 downto 0);
          F6_G : out STD_LOGIC_VECTOR (7 downto 0);
          F7_G : out STD_LOGIC_VECTOR (7 downto 0));
end GameOfLive;

architecture Behavioral of GameOfLive is

type MATRIZ_GOL is array(0 to 7) of std_logic_vector(0 to 7);
signal GOL_MATRIZ, GOL_MATRIX_sig: MATRIZ_GOL;

constant GOL_INIT_OSCILADOR: MATRIZ_GOL:=("00000000",
                                          "00000000",
                                          "00000000",
                                          "00111000",
                                          "00000000",
                                          "00000000",
                                          "00000000",
                                          "00000000");

constant GOL_INIT_PRUEBA: MATRIZ_GOL:= ("11100001",
                                        "00000000",
                                        "01000000",
                                        "00111000",
                                        "00000000",
                                        "10000100",
                                        "00010000",
                                        "00000000");

constant GOL_INIT_PRUEBA2: MATRIZ_GOL:= ("11100001",
                                         "00000000",
                                         "01010000",
                                         "00111000",
```

```
                              "00000000",
                              "10000100",
                              "00010000",
                              "00000000");

constant GOL_INIT_ESTATICO: MATRIZ_GOL:= ("00000000",
                               "00000000",
                               "00011000",
                               "00011000",
                               "00000000",
                               "00000000",
                               "00000000",
                               "00000000");


signal GOL_INIT: MATRIZ_GOL;
type ESTADOS is (REPOSO, ESPERA, INICIO, ITERACION, DEBUG, ITERACION_DEBUG, ESPERA_DEBUG);
signal sig_ESTADO, ESTADO: ESTADOS;
signal EN_tiempo, EN_iterar, IN_DEBUG_sinc, IN_DEBUG_sig, IN_INICIO_sig, IN_INICIO_sinc : std_logic;
signal tiempo, sig_tiempo: unsigned(25 downto 0);


-- Funcion que calcula las vecinas adyacentes vivas dada la posicion de una celda y la matriz.
function F_VECINAS_VIVAS(MATRIZ: MATRIZ_GOL; i, j: integer) return unsigned is
   variable vecinas_vivas: unsigned(3 downto 0);
   variable vecina_i, vecina_j: unsigned(2 downto 0);
   variable i_S, i_N, j_E, j_O: unsigned(2 downto 0);
   begin
      vecinas_vivas := "0000";
      vecina_i := to_unsigned(i,3);
      vecina_j := to_unsigned(j,3);
      i_S := vecina_i + 1; -- Sur
      i_N := vecina_i + 7; -- Norte
      j_E := vecina_j + 1; -- Este
      j_O := vecina_j + 7; -- Oeste

      if (MATRIZ(to_integer(i_N))(to_integer(j_O))='1') then
       vecinas_vivas := vecinas_vivas + 1;
      end if;
      if (MATRIZ(to_integer(i_N))(j)='1') then
         vecinas_vivas := vecinas_vivas + 1;
      end if;
      if (MATRIZ(to_integer(i_N))(to_integer(j_E))='1') then
         vecinas_vivas := vecinas_vivas + 1;
      end if;
      if (MATRIZ(i)(to_integer(j_O))='1') then
         vecinas_vivas := vecinas_vivas + 1;
      end if;
      if (MATRIZ(i)(to_integer(j_E))='1') then
         vecinas_vivas := vecinas_vivas + 1;
      end if;
      if (MATRIZ(to_integer(i_S))(to_integer(j_O))='1') then
         vecinas_vivas := vecinas_vivas + 1;
      end if;
      if (MATRIZ(to_integer(i_S))(j)='1') then
         vecinas_vivas := vecinas_vivas + 1;
      end if;
```

```
      if (MATRIZ(to_integer(i_S))(to_integer(j_E))='1')  then
         vecinas_vivas := vecinas_vivas + 1;
      end if;

   return vecinas_vivas;
end function;


begin

-- Proceso de seleccion de la matriz inicial
process(IN_SW1, IN_SW2)

begin
   if(IN_SW1 = '0' and IN_SW2 = '1') then
      GOL_INIT <= GOL_INIT_OSCILADOR;
   elsif (IN_SW1 = '1' and IN_SW2 = '0') then
      GOL_INIT <= GOL_INIT_PRUEBA;
   elsif (IN_SW1 = '0' and IN_SW2 = '0') then
      GOL_INIT <= GOL_INIT_ESTATICO;
   elsif(IN_SW1 = '1' and IN_SW2 = '1') then
      GOL_INIT <= GOL_INIT_PRUEBA2;
   end if;
end process;

-- Proceso de F/F
process(CLK, RST)

begin

   if (CLK'event and CLK='1') then
      if (RST='0') then
         GOL_MATRIZ <= GOL_INIT;
         ESTADO <= REPOSO;
         tiempo <= (others => '0');
      else
         tiempo <= sig_tiempo;
         IN_DEBUG_sinc <= IN_DEBUG;
         IN_DEBUG_sig <= IN_DEBUG_sinc;
         IN_INICIO_sinc <= IN_INICIO;
         IN_INICIO_sig <= IN_INICIO_sinc;
         ESTADO <= sig_ESTADO;
         if(EN_iterar = '1') then
            GOL_MATRIZ <= GOL_MATRIX_sig;
         end if;
      end if;
   end if;
end process;

-- Temporizador
sig_tiempo <= tiempo + 1 when (EN_tiempo = '1') else
          (others => '0');

-- Proceso de la maquina de estados
process(tiempo,IN_INICIO,  IN_INICIO_sig,  IN_DEBUG,  IN_DEBUG_sig  ,READY_G,  ESTADO,  sig_ESTADO,
GOL_MATRIX_sig, GOL_MATRIZ)
```

```
variable suma_vecinas_vivas: unsigned(3 downto 0);

begin
   sig_ESTADO <= ESTADO;
   EN_iterar <= '0';
   GOL_MATRIX_sig <= GOL_MATRIZ;
   EN_tiempo <= '0';
   LED_INICIO <= '0';
   LED_DEBUG <= '0';
   VALID_G <= '1';

   case ESTADO is

      when REPOSO =>
         VALID_G <= '0';
         LED_INICIO <= '1';
         LED_DEBUG <= '1';
         if(IN_INICIO = '1' and IN_INICIO_sig = '0') then
            sig_ESTADO <= INICIO;
         elsif(IN_DEBUG = '1' and IN_DEBUG_sig = '0') then
            sig_ESTADO <= ESPERA_DEBUG;
         end if;


      when INICIO =>
         LED_INICIO <= '1';
         if(READY_G = '1') then
            sig_ESTADO <= ESPERA;
         end if;


       when ESPERA =>
         LED_INICIO <= '1';
         if(tiempo = 48000000) then
            sig_ESTADO <= ITERACION;
         end if;
         EN_tiempo <= '1';
         VALID_G <= '0';


      when ITERACION =>
         LED_INICIO <= '1';
         for fila_i in 0 to 7 loop
            for columna_j in 0 to 7 loop
               suma_vecinas_vivas := F_VECINAS_VIVAS(GOL_MATRIZ ,fila_i , columna_j);
               if (suma_vecinas_vivas=3) then
                  GOL_MATRIX_sig(fila_i)(columna_j) <= '1';
               elsif (GOL_MATRIZ(fila_i)(columna_j)='1') and (suma_vecinas_vivas=2) then
                  GOL_MATRIX_sig(fila_i)(columna_j) <= '1';
               else
                  GOL_MATRIX_sig(fila_i)(columna_j) <= '0';
               end if;
            end loop;
         end loop;

         VALID_G <= '0';
         EN_iterar <= '1';
```

```
        sig_ESTADO <= INICIO;


      when DEBUG =>
        LED_DEBUG <= '1';
        sig_ESTADO <= ITERACION_DEBUG;


      when ESPERA_DEBUG =>
        VALID_G <= '0';
        LED_DEBUG <= '1';
        if(IN_DEBUG = '1' and IN_DEBUG_sig = '0') then
          sig_ESTADO <= DEBUG;
        end if;


      when ITERACION_DEBUG =>
        LED_DEBUG <= '1';
        for fila_i in 0 to 7 loop
          for columna_j in 0 to 7 loop
            suma_vecinas_vivas := F_VECINAS_VIVAS(GOL_MATRIZ ,fila_i , columna_j);
            if (suma_vecinas_vivas=3) then
              GOL_MATRIX_sig(fila_i)(columna_j) <= '1';
            elsif (GOL_MATRIZ(fila_i)(columna_j)='1') and (suma_vecinas_vivas=2) then
              GOL_MATRIX_sig(fila_i)(columna_j) <= '1';
            else
              GOL_MATRIX_sig(fila_i)(columna_j) <= '0';
            end if;
          end loop;
        end loop;
        VALID_G <= '0';
        EN_iterar <= '1';
        sig_ESTADO <= ESPERA_DEBUG;

    end case;

end process;

-- Actualizacion de los valores de las salidas
F0_G <= GOL_MATRIZ(0);
F1_G <= GOL_MATRIZ(1);
F2_G <= GOL_MATRIZ(2);
F3_G <= GOL_MATRIZ(3);
F4_G <= GOL_MATRIZ(4);
F5_G <= GOL_MATRIZ(5);
F6_G <= GOL_MATRIZ(6);
F7_G <= GOL_MATRIZ(7);


end Behavioral;
```

A3. Bloque Enviar_dato

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Enviar_Dato is
   Port ( CLK : in STD_LOGIC;
          RST : in STD_LOGIC;
          VALID_E : in STD_LOGIC;
          READY_E : out STD_LOGIC;
          S_AXIS_TVALID : out std_logic;
          BOUT : out STD_LOGIC_VECTOR(7 downto 0);
          F0_E : in STD_LOGIC_VECTOR (7 downto 0);
          F1_E : in STD_LOGIC_VECTOR (7 downto 0);
          F2_E : in STD_LOGIC_VECTOR (7 downto 0);
          F3_E : in STD_LOGIC_VECTOR (7 downto 0);
          F4_E : in STD_LOGIC_VECTOR (7 downto 0);
          F5_E : in STD_LOGIC_VECTOR (7 downto 0);
          F6_E : in STD_LOGIC_VECTOR (7 downto 0);
          F7_E : in STD_LOGIC_VECTOR (7 downto 0));
end Enviar_Dato;

architecture Behavioral of Enviar_Dato is

-- Funcion que invierte el orden de los elementos de un vector
function DAR_VUELTA_VECTOR(VECTOR: std_logic_vector; dimension: integer) return std_logic_vector is
   variable vector_sol: std_logic_vector(7 downto 0);
   begin

      for i in dimension-1 downto 0 loop
         vector_sol(i) := VECTOR(-i + dimension-1);
      end loop;

   return vector_sol;
end function;


type ESTADOS is (INICIO,CREA_BYTE, ENVIA_BYTE);
signal sig_ESTADO, ESTADO: ESTADOS;
signal datos, sig_datos: std_logic_vector(63 downto 0);
signal byte : std_logic_vector(7 downto 0);
signal cont_iteraciones, cont_iteraciones_sig, EN_desp, EN_cnt_bit, EN_cnt_byte, INIT_cnt_bit, INIT_cnt_byte,
EN_tiempo, INIT_tiempo, EN_leer: std_logic;
signal cnt_bit, sig_cnt_bit: unsigned(10 downto 0);
signal cnt_byte, sig_cnt_byte: unsigned(1 downto 0);
signal tiempo, sig_tiempo: unsigned(9 downto 0);

begin

-- Proceso de F/F
process(CLK, RST)

begin
```

```
  if (CLK'event and CLK='1') then
    if (RST='0') then
      datos <= (others => '0');
      ESTADO <= INICIO;
      cnt_bit <= (others => '0');
      cnt_byte <= (others => '0');
      tiempo <= (others => '0');
      cont_iteraciones <= '0';
    else
      datos <= sig_datos;
      cnt_bit <= sig_cnt_bit;
      cnt_byte <= sig_cnt_byte;
      tiempo <= sig_tiempo;
      ESTADO <= sig_ESTADO;
      cont_iteraciones <= cont_iteraciones_sig;
    end if;
  end if;
end process;

-- Proceso de la maquina de estados
process(cont_iteraciones, VALID_E,datos,cnt_bit,cnt_byte,tiempo, ESTADO, sig_ESTADO, byte)
begin
  EN_leer <= '0';
  EN_desp <= '0';
  EN_tiempo <= '0';
  INIT_tiempo <= '0';
  EN_cnt_bit <= '0';
  INIT_cnt_bit <= '0';
  EN_cnt_byte <= '0';
  INIT_cnt_byte <= '0';
  sig_ESTADO <= ESTADO;
  sig_tiempo <= tiempo + 1;
  EN_desp <= '0';
  S_AXIS_TVALID <= '0';
  BOUT <= byte;
  cont_iteraciones_sig <= cont_iteraciones;

 if(datos(63) = '1') then
    byte <= "00001111";
 else
    byte <= (others => '0');
 end if;

  case ESTADO is

    when INICIO =>
      READY_E <= '1';
       INIT_cnt_bit <= '1'; -- Inicializa cnt_bit
      cont_iteraciones_sig <= '0';
      if(VALID_E = '1') then
        EN_leer <= '1'; -- Se leen nuevos datos
        sig_ESTADO <= CREA_BYTE;
        sig_tiempo <= (others => '0');
      end if;

    when CREA_BYTE =>
```

```
        sig_tiempo <= (others => '0');

        READY_E <= '0';

        if(cnt_bit = 64) then
          sig_ESTADO <= INICIO;
          INIT_cnt_bit <= '1'; -- Inicializa cnt_bit
        else
          if(cont_iteraciones = '1') then
            EN_desp <= '1'; -- Enalbe del registro de desplazamiento
          end if;
          EN_cnt_bit <= '1'; -- Suma 1 a cnt_bit
          sig_ESTADO <= ENVIA_BYTE;
        end if;

      when ENVIA_BYTE =>
        S_AXIS_TVALID <= '1';
        READY_E <= '0';
        if(cnt_byte < 2 and tiempo = 999) then
          sig_ESTADO <= ENVIA_BYTE;
          EN_cnt_byte <= '1'; -- Suma 1 a cnt_byte
          sig_tiempo <= (others => '0');
          S_AXIS_TVALID <= '1';
        elsif(cnt_byte = 2 and tiempo = 998 and cnt_bit < 64) then
          sig_ESTADO <= CREA_BYTE;
          INIT_cnt_byte <= '1'; -- Inicializa cnt_byte
          sig_tiempo <= (others => '0');
          S_AXIS_TVALID <= '1';
        elsif(cnt_byte = 2 and tiempo = 995 and cnt_bit = 64) then
          sig_tiempo <= (others => '0');
          sig_ESTADO <= INICIO;
          INIT_cnt_byte <= '1'; -- Inicializa cnt_byte

        end if;
        cont_iteraciones_sig <= '1';
    end case;

end process;

--Contador bit
sig_cnt_bit <= cnt_bit + 1 when (EN_cnt_bit = '1') else
          (others => '0') when (INIT_cnt_bit = '1') else
          cnt_bit;

--Contador byte
sig_cnt_byte <= cnt_byte + 1 when (EN_cnt_byte = '1') else
          (others => '0') when (INIT_cnt_byte = '1') else
          cnt_byte;

-- Almacenamiento y desplazamiento de datos
sig_datos <= F0_E & DAR_VUELTA_VECTOR(F1_E,8) & F2_E & DAR_VUELTA_VECTOR(F3_E,8) & F4_E &
DAR_VUELTA_VECTOR(F5_E,8) & F6_E & DAR_VUELTA_VECTOR(F7_E,8) when (EN_leer = '1') else
         datos(62 downto 0) & '0' when (EN_desp = '1') else
         datos;


end Behavioral;
```

A4. Bloque ws2812

```
library ieee ;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity ws2812 is
  port (
    AXIS_ARESETN : in std_logic; -- Reset asinc/sinc activo en bajo
    AXIS_ACLK : in std_logic; -- Reloj 100MHz
    S_AXIS_TDATA : in std_logic_vector(7 downto 0) ;
    S_AXIS_TVALID : in std_logic;
    S_AXIS_TREADY : out std_logic;
    TO_WS2812 : out std_logic -- Control del ws2812
     ) ;
end ws2812 ;

architecture Behavioral of ws2812 is
    type ESTADOS is (REPOSO, ENVIAR1, ENVIAR0);
    signal sig_ESTADO, ESTADO: ESTADOS;
    --Contador de pulsos de reloj:
    signal sig_tiempo, tiempo: unsigned(6 downto 0);
    --Contador de bits enviados:
    signal sig_contador, contador: unsigned(2 downto 0);
    signal dato, datoactual, sig_datoactual: std_logic_vector(7 downto 0);
    signal EN_OUT, EN_reposo : std_logic;


begin

    -- Proceso de F/F
    process(AXIS_ACLK, AXIS_ARESETN)
    begin
        if(AXIS_ARESETN = '0') then
            ESTADO <= REPOSO;
            tiempo <= (others => '0');
            contador <= (others => '0');
        elsif rising_edge(AXIS_ACLK) then
            ESTADO <= sig_ESTADO;
            tiempo <= sig_tiempo;
            contador <= sig_contador;
            datoactual <= sig_datoactual;

        end if;
    end process;

    --Almacenamiento del dato
    dato <= S_AXIS_TDATA;

    --Maquina de estados
    process(EN_reposo,ESTADO, tiempo, S_AXIS_TVALID, S_AXIS_TDATA, contador, datoactual, dato)
    begin

      sig_contador <= contador;
      sig_datoactual <= datoactual;
```

```
    EN_reposo <= '0';
    sig_ESTADO <= estado;
    S_AXIS_TREADY <= '0';

    case ESTADO is
      when REPOSO =>
        EN_OUT <= '0';
        sig_contador <= (others => '0');
        sig_tiempo <= (others => '0');
        EN_reposo <= '1';
        S_AXIS_TREADY <= '0';
        if(S_AXIS_TVALID = '1') then
          sig_ESTADO <= ENVIAR1;
          sig_datoactual <= S_AXIS_TDATA;
        end if;

      when ENVIAR1 =>
        if(EN_reposo = '1') then
          EN_OUT <= '0';
          sig_contador <= (others => '0');
          sig_tiempo <= (others => '0');
        end if;
        sig_tiempo <= tiempo +1;
        EN_OUT <= '1';
        if ((datoactual(7)= '0' and tiempo = 24) or (datoactual(7)= '1' and tiempo = 89)) then
          sig_ESTADO <= ENVIAR0;
        else
          sig_ESTADO <= ENVIAR1;
        end if;

      when ENVIAR0 =>
        sig_tiempo <= tiempo + 1;
        EN_OUT <= '0';
        if (tiempo = 124 and contador < 7) then
          sig_contador <= contador + 1;
          sig_ESTADO <= ENVIAR1;
          sig_datoactual <= datoactual(6 downto 0) & '0';
          sig_tiempo <= (others=> '0');
        elsif ((tiempo = 123) and (contador = 7)) then
          sig_contador <= (others=> '0');
          sig_datoactual <= dato;
          sig_ESTADO <= REPOSO;
          sig_tiempo <= (others=> '0');
          EN_reposo <= '1';
        else
          sig_ESTADO <= ENVIAR0;
        end if;

    end case;
  end process;

  -- Salida
  TO_WS2812 <= '1' when (EN_OUT = '1') else '0';

end Behavioral;
```

A5. Bloque GOL_Panel_TOP

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity GOL_Panel_TOP is
  Port (RST : in std_logic;
        CLK : in std_logic;
        IN_DEBUG : in std_logic;
        IN_INICIO: in std_logic;
        LED_INICIO: out STD_LOGIC;
        LED_DEBUG: out STD_LOGIC;
        IN_SW1: in STD_LOGIC;
        IN_SW2: in STD_LOGIC;
        TO_WS2812 : out std_logic);
end GOL_Panel_TOP;

architecture Behavioral of GOL_Panel_TOP is
  component sreset is
    Port ( RST : in STD_LOGIC;
           CLK : in STD_LOGIC;
           AXIS_ARESETN: out STD_LOGIC);
  end component;

  component GameOfLive is
    Port ( CLK : in STD_LOGIC;
           RST : in STD_LOGIC;
           VALID_G : out STD_LOGIC;
           READY_G : in STD_LOGIC;
           IN_DEBUG: in STD_LOGIC;
           IN_INICIO: in STD_LOGIC;
           LED_INICIO: out STD_LOGIC;
           LED_DEBUG: out STD_LOGIC;
           IN_SW1: in STD_LOGIC;
           IN_SW2: in STD_LOGIC;
           F0_G : out STD_LOGIC_VECTOR (7 downto 0);
           F1_G : out STD_LOGIC_VECTOR (7 downto 0);
           F2_G : out STD_LOGIC_VECTOR (7 downto 0);
           F3_G : out STD_LOGIC_VECTOR (7 downto 0);
           F4_G : out STD_LOGIC_VECTOR (7 downto 0);
           F5_G : out STD_LOGIC_VECTOR (7 downto 0);
           F6_G : out STD_LOGIC_VECTOR (7 downto 0);
           F7_G : out STD_LOGIC_VECTOR (7 downto 0));
  end component;

  component Enviar_dato is
    Port ( CLK : in STD_LOGIC;
           RST : in STD_LOGIC;
           VALID_E : in STD_LOGIC;
           READY_E : out STD_LOGIC;
           S_AXIS_TVALID : out std_logic;
           BOUT : out STD_LOGIC_VECTOR(7 downto 0);
           F0_E : in STD_LOGIC_VECTOR (7 downto 0);
```

```
        F1_E : in STD_LOGIC_VECTOR (7 downto 0);
        F2_E : in STD_LOGIC_VECTOR (7 downto 0);
        F3_E : in STD_LOGIC_VECTOR (7 downto 0);
        F4_E : in STD_LOGIC_VECTOR (7 downto 0);
        F5_E : in STD_LOGIC_VECTOR (7 downto 0);
        F6_E : in STD_LOGIC_VECTOR (7 downto 0);
        F7_E : in STD_LOGIC_VECTOR (7 downto 0));
    end component;

    component ws2812 is
        Port (
        AXIS_ARESETN : in std_logic;
        AXIS_ACLK : in std_logic;
        S_AXIS_TDATA : in std_logic_vector(7 downto 0) ;
        S_AXIS_TVALID : in std_logic;
        S_AXIS_TREADY : out std_logic;
        TO_WS2812 : out std_logic) ;
    end component ;

    signal F0_GE, F1_GE, F2_GE, F3_GE, F4_GE, F5_GE, F6_GE, F7_GE, BOUT_GE: std_logic_vector(7 downto 0);
    signal VALID_GE, READY_GE, S_AXIS_TVALID_GE, S_AXIS_TREADY_GE, AXIS_ARESETN_GE: std_logic;

begin

     U_sreset: sreset
        Port map( RST => RST,
                  CLK => CLK,
                  AXIS_ARESETN => AXIS_ARESETN_GE);

    U_GameOfLive: GameOfLive
        Port map( CLK => CLK,
                  RST => AXIS_ARESETN_GE,
                  VALID_G => VALID_GE,
                  READY_G => READY_GE,
                  IN_DEBUG => IN_DEBUG,
                  IN_INICIO => IN_INICIO,
                  LED_INICIO => LED_INICIO,
                  LED_DEBUG => LED_DEBUG,
                  IN_SW1 => IN_SW1,
                  IN_SW2 => IN_SW2,
                  F0_G => F0_GE,
                  F1_G => F1_GE,
                  F2_G => F2_GE,
                  F3_G => F3_GE,
                  F4_G => F4_GE,
                  F5_G => F5_GE,
                  F6_G => F6_GE,
                  F7_G => F7_GE);


    U_Enviar_dato: Enviar_dato
        Port map(CLK => CLK,
                 RST => AXIS_ARESETN_GE,
                 VALID_E => VALID_GE,
                 READY_E => READY_GE,
                 S_AXIS_TVALID => S_AXIS_TVALID_GE,
                 BOUT => BOUT_GE,
```

```
            F0_E => F0_GE,
            F1_E => F1_GE,
            F2_E => F2_GE,
            F3_E => F3_GE,
            F4_E => F4_GE,
            F5_E => F5_GE,
            F6_E => F6_GE,
            F7_E => F7_GE);

    U_ws2812 : ws2812
      Port map( AXIS_ARESETN => AXIS_ARESETN_GE,
            AXIS_ACLK => CLK,
            S_AXIS_TDATA => BOUT_GE,
            S_AXIS_TVALID => S_AXIS_TVALID_GE,
            S_AXIS_TREADY => S_AXIS_TREADY_GE,
            TO_WS2812 => TO_WS2812);

end Behavioral;
```

A6. Test Bench

```
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.numeric_std.all ;
library std;
--Librerias para leer y escribir ficheros
use std.textio.all;
use IEEE.STD_LOGIC_TEXTIO.ALL;

entity GOL_Panel_tb is
end GOL_Panel_tb;

architecture Behavioral of GOL_Panel_tb is

component sreset is
   Port ( RST : in STD_LOGIC;
          CLK : in STD_LOGIC;
          AXIS_ARESETN: out STD_LOGIC);
end component;

component GameOfLive is
   Port ( CLK : in STD_LOGIC;
          RST : in STD_LOGIC;
          VALID_G : out STD_LOGIC;
          READY_G : in STD_LOGIC;
          IN_DEBUG: in STD_LOGIC;
          IN_INICIO: in STD_LOGIC;
          LED_INICIO: out STD_LOGIC;
          LED_DEBUG: out STD_LOGIC;
          IN_SW1: in STD_LOGIC;
          IN_SW2: in STD_LOGIC;
          F0_G : out STD_LOGIC_VECTOR (7 downto 0);
          F1_G : out STD_LOGIC_VECTOR (7 downto 0);
          F2_G : out STD_LOGIC_VECTOR (7 downto 0);
          F3_G : out STD_LOGIC_VECTOR (7 downto 0);
          F4_G : out STD_LOGIC_VECTOR (7 downto 0);
          F5_G : out STD_LOGIC_VECTOR (7 downto 0);
          F6_G : out STD_LOGIC_VECTOR (7 downto 0);
          F7_G : out STD_LOGIC_VECTOR (7 downto 0));
end component;

component Enviar_Dato is
   Port ( CLK : in STD_LOGIC;
          RST : in STD_LOGIC;
          VALID_E : in STD_LOGIC;
          READY_E : out STD_LOGIC;
          S_AXIS_TVALID : out std_logic;
          BOUT : out STD_LOGIC_VECTOR(7 downto 0);
          F0_E : in STD_LOGIC_VECTOR (7 downto 0);
          F1_E : in STD_LOGIC_VECTOR (7 downto 0);
          F2_E : in STD_LOGIC_VECTOR (7 downto 0);
          F3_E : in STD_LOGIC_VECTOR (7 downto 0);
          F4_E : in STD_LOGIC_VECTOR (7 downto 0);
```

```
        F5_E : in STD_LOGIC_VECTOR (7 downto 0);
        F6_E : in STD_LOGIC_VECTOR (7 downto 0);
        F7_E : in STD_LOGIC_VECTOR (7 downto 0));
end component;

component ws2812 is
   port ( AXIS_ARESETN : in std_logic;
      AXIS_ACLK : in std_logic;
      S_AXIS_TDATA : in std_logic_vector(7 downto 0) ;
      S_AXIS_TVALID : in std_logic;
      S_AXIS_TREADY: out std_logic;
      TO_WS2812 : out std_logic);
end component ;


signal  AXIS_ARESETN_tb,  IN_SW1_tb,  IN_SW2_tb,  LED_INICIO_tb,  LED_DEBUG_tb,  IN_INICIO_tb,  RST_tb,
CLK_tb,  TO_WS2812_tb,  VALID_tb,  READY_tb,  S_AXIS_TREADY_tb,  S_AXIS_TVALID_tb,  IN_DEBUG_tb  :
std_logic;
signal BOUT_tb, F0_tb, F1_tb, F2_tb, F3_tb, F4_tb, F5_tb, F6_tb, F7_tb : std_logic_vector(7 downto 0);
constant TCLK: time:= 10ns;


constant bit_time : time := 1 us;

-- Calculo del periodo y ton de la señal TO_WS2812
signal tiempo_up : time := 0 ns;
signal tiempo_dwn : time := 0 ns;
signal periodo : time := 0 ns;
signal ton : time := 0 ns;
signal toff : time := 0 ns;


signal bit_ws : std_logic;

signal numIteraciones_des : unsigned(7 downto 0);


begin

   -- Numero de iteraciones deseadas
   numIteraciones_des <= "00000011";

   --Proceso que escribe y lee de ficheros para comprobar el funcionamiento
   Escribir_iteraciones_comprobar: process

   file datos, datos_VHDL, datos_MATLAB, estado: text;
   variable linea, linea_VHDL, linea_MATLAB, linea_fallo: line;
   variable status, status1, status2, status3: file_open_status;
   variable numIteraciones: unsigned(7 downto 0);
   variable dato_VHDL, dato_MATLAB: std_logic_vector(7 downto 0);
   variable fallo: std_logic;
   variable charm, charv: character;

   variable contador_lineas, contador_iteraciones: unsigned(7 downto 0);

   begin
      numIteraciones := (others => '0');
```

```
file_open(status, datos, "Matriz_VHDL.txt", WRITE_MODE);
  if(status = open_ok) then
    wait for 30 ns;
    loop
      -- Escribe cada carater con un espacio en medio
      numIteraciones := numIteraciones + 1;
      for i in 7 downto 0 loop
        write(linea, F0_tb(i));
        write(linea, string'(" "));
      end loop;
      writeline(datos, linea);

      for i in 7 downto 0 loop
        write(linea, F1_tb(i));
        write(linea, string'(" "));
      end loop;
      writeline(datos, linea);

      for i in 7 downto 0 loop
        write(linea, F2_tb(i));
        write(linea, string'(" "));
      end loop;
      writeline(datos, linea);

      for i in 7 downto 0 loop
        write(linea, F3_tb(i));
        write(linea, string'(" "));
      end loop;
      writeline(datos, linea);

      for i in 7 downto 0 loop
        write(linea, F4_tb(i));
        write(linea, string'(" "));
      end loop;
      writeline(datos, linea);

      for i in 7 downto 0 loop
        write(linea, F5_tb(i));
        write(linea, string'(" "));
      end loop;
      writeline(datos, linea);

      for i in 7 downto 0 loop
        write(linea, F6_tb(i));
        write(linea, string'(" "));
      end loop;
      writeline(datos, linea);

      for i in 7 downto 0 loop
        write(linea, F7_tb(i));
        write(linea, string'(" "));
      end loop;
      writeline(datos, linea);

      exit when numIteraciones = numIteraciones_des + 1;
      wait for 2.5 ms;
    end loop;
```

```
        file_close(datos);
    end if;

    file_open(status1, datos_VHDL, "Matriz_VHDL.txt", READ_MODE);
    file_open(status2, datos_MATLAB, "Matriz_MATLAB.txt", READ_MODE);
    file_open(status1, estado, "Estado.txt", WRITE_MODE);
    contador_iteraciones := (others => '0');
    contador_lineas := (others => '0');
    fallo:= '0';
    if(status1 = open_ok and status2 = open_ok) then
      while (not ENDFILE(datos_VHDL)) or (not ENDFILE(datos_MATLAB)) loop
        readline(datos_VHDL, linea_VHDL);
        readline(datos_MATLAB, linea_MATLAB);

        if (contador_lineas = 8) then
          contador_lineas := (others => '0');
          contador_iteraciones := contador_iteraciones + 1;
        end if;
        -- Lee y compara caracter a caracter de ambos ficheros
        for i in 14 downto 0 loop
          read(linea_VHDL, charv);
          read(linea_MATLAB, charm);
          if (charm /= ' ' and charv /= ' ') then
            if(charm = '1' and charv = '1') then
              dato_MATLAB(i/2) := '1';
              dato_VHDL(i/2) := '1';
            elsif(charm = '0' and charv = '0') then
              dato_MATLAB(i/2) := '0';
              dato_VHDL(i/2) := '0';
            -- Si se comete algun fallo
            else
              fallo:= '1';
              if(status1 = open_ok) then
                write(linea_fallo, string'("Se ha producido un fallo en la iteracion  "));
                write(linea_fallo, to_integer(contador_iteraciones));
                write(linea_fallo, string'(", linea "));
                write(linea_fallo, to_integer(contador_lineas +1));
                write(linea_fallo, string'(" /// "));
              end if;

            end if;
          end if;
        end loop;
        contador_lineas := contador_lineas + 1;
       end loop;
      if(fallo /= '1') then
        write(linea_fallo, string'("Todas las iteraciones son correctas"));
      end if;
      writeline(estado, linea_fallo);
      file_close(datos_VHDL);
      file_close(datos_MATLAB);
      file_close(estado);
    end if;
  wait;
end process;
```

```
RST_tb <= '1', '0' after TCLK;

-- Interruptores para una configuracion de oscilador
IN_SW1_tb <= '1';
IN_SW2_tb <= '1';

--Inicio en modo normal
IN_DEBUG_tb <= '0';

 --Proceso de pulsador de modo inicio normal
 process begin
      IN_INICIO_tb <= '0';
      wait for 100ns;
      IN_INICIO_tb <= '1';
      wait for 1us;
      IN_INICIO_tb <= '0';
      wait;
  end process;

-- Proceso de reloj
process
begin
   CLK_tb <= '0', '1' after TCLK/2;
   wait for TCLK;
end process;

--Proceso de calculo de periodo y tiempos de on y off con la funcion now
process
begin
wait until (rising_edge(TO_WS2812_tb));
periodo <= now-tiempo_up;
toff <= now -tiempo_dwn;
ton <= tiempo_dwn-tiempo_up;
tiempo_up<=now;
wait for 1 ns;
-- Reconstrucion de bit
if (ton /= 0 ns) then
   if (periodo /= 1250 ns) then
     bit_ws <= 'X', '-' after 50 us;
   elsif (ton = 900 ns) then
     bit_ws <= '1', '-' after 50 us;
    elsif (ton = 250 ns) then
     bit_ws <= '0', '-' after 50 us;
   else
     bit_ws <= 'X', '-' after 50 us;
   end if;
   if (toff > 10 us) then
     bit_ws <= '-';
   end if;
end if;
wait until falling_edge(TO_WS2812_tb);
tiempo_dwn <= now;
end process;

    U_sreset: sreset
      Port map( RST => RST_tb,
```

```
            CLK => CLK_tb,
            AXIS_ARESETN => AXIS_ARESETN_tb);

    U_GameOfLive: GameOfLive
    Port map(CLK => CLK_tb,
        RST => AXIS_ARESETN_tb,
        VALID_G => VALID_tb,
        READY_G => READY_tb,
        IN_DEBUG => IN_DEBUG_tb,
        IN_INICIO => IN_INICIO_tb,
        LED_INICIO => LED_INICIO_tb,
        LED_DEBUG => LED_DEBUG_tb,
        IN_SW1 => IN_SW1_tb,
        IN_SW2 => IN_SW2_tb,
        F0_G => F0_tb,
        F1_G => F1_tb,
        F2_G => F2_tb,
        F3_G => F3_tb,
        F4_G => F4_tb,
        F5_G => F5_tb,
        F6_G => F6_tb,
        F7_G => F7_tb);

  U_Enviar_dato: Enviar_dato
    Port map(CLK => CLK_tb,
        RST => AXIS_ARESETN_tb,
        VALID_E => VALID_tb,
        READY_E => READY_tb,
        S_AXIS_TVALID => S_AXIS_TVALID_tb,
        BOUT => BOUT_tb,
        F0_E => F0_tb,
        F1_E => F1_tb,
        F2_E => F2_tb,
        F3_E => F3_tb,
        F4_E => F4_tb,
        F5_E => F5_tb,
        F6_E => F6_tb,
        F7_E => F7_tb);
  U_ws2812: ws2812
    Port map(AXIS_ARESETN => AXIS_ARESETN_tb,
      AXIS_ACLK => CLK_tb,
      S_AXIS_TDATA => BOUT_tb,
      S_AXIS_TVALID => S_AXIS_TVALID_tb,
      S_AXIS_TREADY => S_AXIS_TREADY_tb,
      TO_WS2812 => TO_WS2812_tb);


end Behavioral;
```

A7. Código de Matlab

```
close all;
clear;
clc;

% Matriz inicial para seleccionar

oscilador  = [0 0 0 0 0 0 0 0 0 0;
              0 0 0 0 0 0 0 0 0 0;
              0 0 0 0 0 0 0 0 0 0;
              0 0 0 0 0 0 0 0 0 0;
              0 0 0 1 1 1 0 0 0 0;
              0 0 0 0 0 0 0 0 0 0;
              0 0 0 0 0 0 0 0 0 0;
              0 0 0 0 0 0 0 0 0 0;
              0 0 0 0 0 0 0 0 0 0;
              0 0 0 0 0 0 0 0 0 0];

bloque  = [0 0 0 0 0 0 0 0 0 0;
              0 0 0 0 0 0 0 0 0 0;
              0 0 0 0 0 0 0 0 0 0;
              0 0 0 0 1 1 0 0 0 0;
              0 0 0 0 1 1 0 0 0 0;
              0 0 0 0 0 0 0 0 0 0;
              0 0 0 0 0 0 0 0 0 0;
              0 0 0 0 0 0 0 0 0 0;
              0 0 0 0 0 0 0 0 0 0;
              0 0 0 0 0 0 0 0 0 0];


prueba = [0 0 0 0 0 0 0 0 0 0;
          1 1 1 1 0 0 0 0 1 1;
          0 0 0 0 0 0 0 0 0 0;
          0 0 1 0 0 0 0 0 0 0;
          0 0 0 1 1 1 0 0 0 0;
          0 0 0 0 0 0 0 0 0 0;
          0 1 0 0 0 0 1 0 0 1;
          0 0 0 0 1 0 0 0 0 0;
          0 0 0 0 0 0 0 0 0 0;
          1 1 1 1 0 0 0 0 1 1];

 prueba2 = [0 0 0 0 0 0 0 0 0 0;
          1 1 1 1 0 0 0 0 1 1;
          0 0 0 0 0 0 0 0 0 0;
          0 0 1 0 1 0 0 0 0 0;
          0 0 0 1 1 1 0 0 0 0;
          0 0 0 0 0 0 0 0 0 0;
          0 1 0 0 0 0 1 0 0 1;
          0 0 0 0 1 0 0 0 0 0;
          0 0 0 0 0 0 0 0 0 0;
          1 1 1 1 0 0 0 0 1 1];


% Eleccion de la matriz inicial:
```

```
Matriz_inicial_marco = prueba;

%Número de iteraciones:

numIteracion = 100;


matriz_marco = Matriz_inicial_marco;
matriz = F_quitar_marco(matriz_marco);
matriz_marco_sig = zeros(10,10);

writematrix([], 'Matriz_MATLAB.txt');
type Matriz_MATLAB.txt;

for iteraciones = 0:numIteracion
   %Representacion grafica de la matriz
   imagesc(matriz);
   colormap cool
   colormap([0 0 0 ; 0 1 0]);
   title(['Grid at Iteration ',num2str(iteraciones)]);
   drawnow;

   %Escribe la matriz en un fichero
   writematrix(matriz, 'Matriz_MATLAB.txt', 'Delimiter', ' ' ,'WriteMode', 'append');

   %Recorre la matriz y le asigna un valor a cada celda dependiendo de sus
   %vecinas vivas y su valor anterior
   for fila_i = 2:9
      for columna_j = 2:9
         suma_vecinas_vivas = F_vecinas_vivas(matriz_marco, fila_i, columna_j);
         if(suma_vecinas_vivas == 3)
            matriz_marco_sig(fila_i, columna_j) = 1;
         elseif (matriz_marco(fila_i, columna_j) == 1) && (suma_vecinas_vivas == 2)
            matriz_marco_sig(fila_i, columna_j) = 1;
         else
            matriz_marco_sig(fila_i, columna_j) = 0;
         end

      end
   end

   %Actualiza el valor de la matriz
   matriz_marco = matriz_marco_sig;
   matriz_marco = F_modificar_marco(matriz_marco);
   matriz = F_quitar_marco(matriz_marco);


end



%Funcion que suma las vecinas vivas de cada celda
function X = F_vecinas_vivas(matriz_marco, vecinas_i, vecinas_j)
   X = 0;
   i_S = vecinas_i + 1;
   i_N = vecinas_i - 1;
```

```
   j_E = vecinas_j + 1;
   j_O = vecinas_j - 1;

   if matriz_marco(i_N, j_O) == 1
      X = X + 1;

   end
   if matriz_marco(i_N, vecinas_j) == 1
      X = X + 1;
   end
   if matriz_marco(i_N, j_E)== 1
      X = X + 1;
   end
   if matriz_marco(vecinas_i, j_O) == 1
      X = X + 1;
   end
   if matriz_marco(vecinas_i, j_E) == 1
      X = X + 1;
   end
   if matriz_marco(i_S, j_O) == 1
      X = X + 1;
   end
   if matriz_marco(i_S, vecinas_j) == 1
      X = X + 1;
   end
   if matriz_marco(i_S, j_E) == 1
      X = X + 1;
   end

end


%Funcion que copia las filas y columnas de los extremos y añade el nuevo
%marco
function Y = F_modificar_marco(matriz_marco)
      Y(1,1) = matriz_marco(9,9);
      Y(1, 10) = matriz_marco(9,2);
      Y(10, 1) = matriz_marco(2,9);
      Y(10, 10) = matriz_marco(2,2);
      Y(1, 2:9) = matriz_marco(9, 2:9);
      Y(10, 2:9) = matriz_marco(2, 2:9);
      Y(2:9, 1) = matriz_marco (2:9, 9);
      Y(2:9, 10) = matriz_marco (2:9, 2);
      Y(2:9, 2:9) = matriz_marco (2:9, 2:9);

end

%Funcion que quita el marco a la matriz de celdas
function Z = F_quitar_marco(matriz_marco)
   Z(1:8, 1:8) = matriz_marco(2:9, 2:9);
end
```

Anexo B

B1. Hoja de características de los pixels del panel WS2812B

Worldsemi

WS2812B

Intelligent control LED integrated light source

Features and Benefits

- Intelligent reverse connect protection, the power supply reverse connection does not damage the IC.
- The control circuit and the LED share the only power source.
- Control circuit and RGB chip are integrated in a package of 5050 components, form a complete control of pixel point.
- Built-in signal reshaping circuit, after wave reshaping to the next driver, ensure wave-form distortion not accumulate.
- Built-in electric reset circuit and power lost reset circuit.
- Each pixel of the three primary color can achieve 256 brightness display, completed 16777216 color full color display, and scan frequency not less than 400Hz/s.
- Cascading port transmission signal by single line.
- Any two point the distance more than 5m transmission signal without any increase circuit.
- When the refresh rate is 30fps, cascade number are not less than1024 points.
- Send data at speeds of 800Kbps.
- The color of the light were highly consistent, cost-effective..

Applications

- Full-color module, Full color soft lights a lamp strip.
- LED decorative lighting, Indoor/outdoor LED video irregular screen.

General description

WS2812B is a intelligent control LED light source that the control circuit and RGB chip are integrated in a package of 5050 components. It internal include intelligent digital port data latch and signal reshaping ampli fication drive circuit. Also include a precision internal oscillator and a 12V voltage programmable constant curr e-nt control part, effectively ensuring the pixel point light color height consistent.

The data transfer protocol use single NZR communication mode. After the pixel power-on reset, the DIN port receive data from controller, the first pixel collect initial 24bit data then sent to the internal data latch, the other data which reshaping by the internal signal reshaping amplification circuit sent to the next cascade pixel through the DO port. After transmission for each pixel, the signal to reduce 24bit. pixel adopt auto resha -ping transmit technology, making the pixel cascade number is not limited the signal transmission, only depend on the speed of signal transmission.

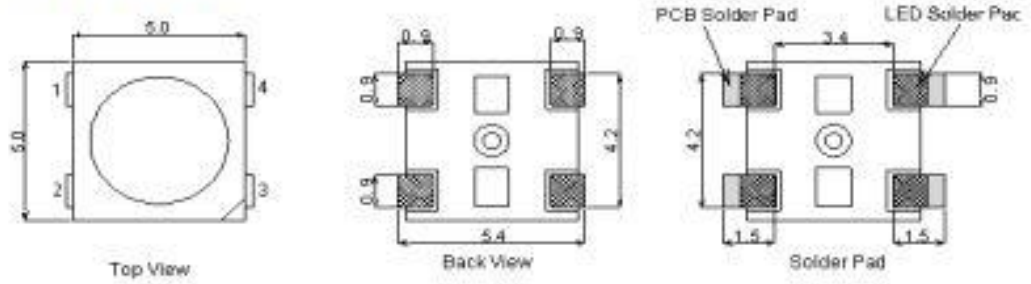
LED with low driving voltage, environmental protection and energy saving, high brightness, scattering angl e is large, good consistency, low power, long life and other advantages. The control chip integrated in LED above becoming more simple circuit, small volume, convenient installation.

WS2812B

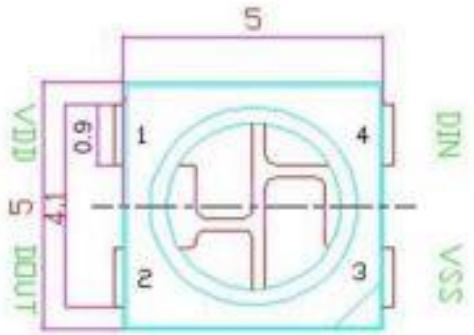
Intelligent control LED integrated light source

Worldsemi

Mechanical Dimensions



PIN configuration



PIN function

NO.	Symbol	Function description
1	VDD	Power supply LED
2	DOUT	Control data signal output
3	VSS	Ground
4	DIN	Control data signal input

Absolute Maximum Ratings

Prameter	Symbol	Ratings	Unit
Power supply voltage	V_{DD}	+3.5~+5.3	V
Input voltage	V_I	-0.5~VDD+0.5	V
Operation junction temperature	Topt	-25~+80	℃
Storage temperature range	Tstg	-40~+105	℃

Electrical Characteristics（T_A=-20~+70℃, V_{DD}=4.5~5.5V,V_{SS}=0V,unless otherwise specified）

Prameter	Smybol	conditions	Min	Tpy	Max	Unit
Input current	I_I	V_I=V_{DD}/V_{SS}	——	——	±1	μA
Input voltage level	V_{IH}	D_{IN}, SET	0.7V_{DD}	——	——	V
	V_{IL}	D_{IN}, SET	——	——	0.3 V_{DD}	V
Hysteresis voltage	V_H	D_{IN}, SET	——	0.35	——	V

Switching characteristics（T_A=-20～+70℃, V_{DD}=4.5～5.5V,V_{SS}=0V,unless otherwise specified）

Prameter	Symbol	Condition	Min	Tpy	Max	Unit
Transmission delay time	t_{PLZ}	CL=15pF,DIN→ DOUT,RL=10KΩ	——	——	300	ns
Fall time	t_{THZ}	CL=300pF,OUTR/OUTG/OUTB	——	——	120	μs
Input capcity	C_I	——	——	——	15	pF

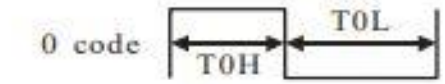
RGB IC characteristic parameter

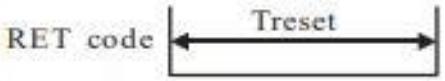
Emitting color	Model	Wavelength(nm)	Luminous intensity(mcd)	Voltage(V)
Red	13CBAUP	620-625	390-420	2.0-2.2
Green	13CGAUP	522-525	660-720	3.0-3.4
Blue	10R1MUX	465-467	180-200	3.0-3.4

Data transfer time(TH+TL=1.25μs±600ns)

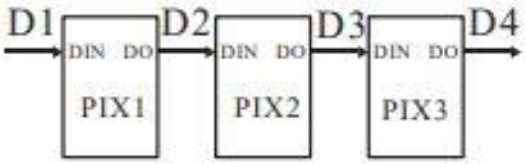
T0H	0 code ,high voltage time	0.4us	±150ns
T1H	1 code ,high voltage time	0.8us	±150ns
T0L	0 code , low voltage time	0.85us	±150ns
T1L	1 code ,low voltage time	0.45us	±150ns
RES	low voltage time	Above 50μs	

Sequence chart:

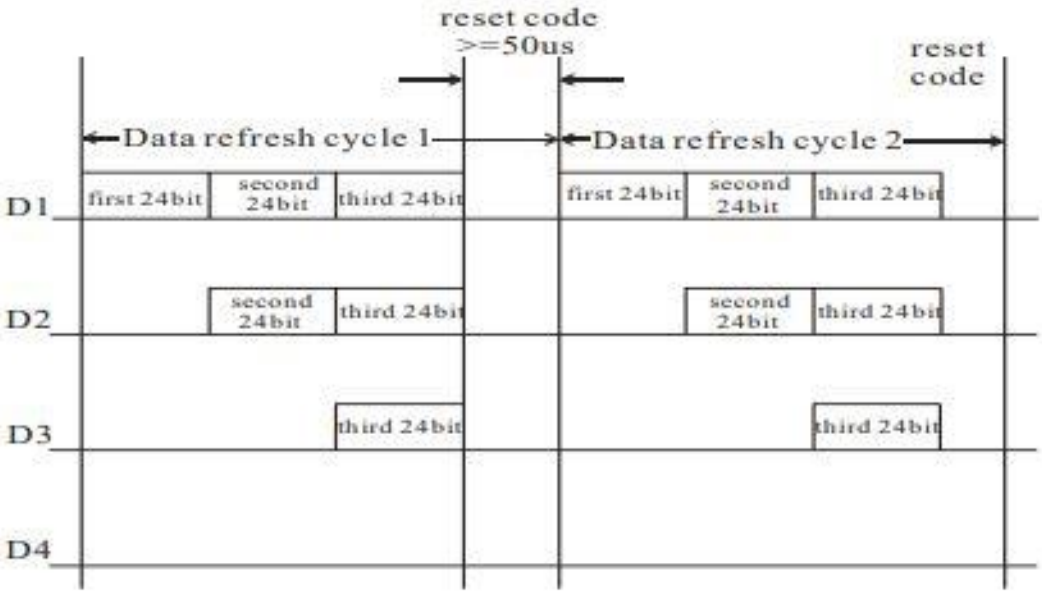




Cascade method:



Data transmission method:



Note: The data of D1 is send by MCU,and D2, D3, D4 through pixel internal reshaping amplification to transmit.

Composition of 24bit data:

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

Note: Follow the order of GRB to sent data and the high bit sent at first.

Typical application circuit: