Trabajo fin de Máster
Ingeniería de Electrónica, robótica y automática

Generador Automático de Drivers y Monitores para la verificación de circuitos digitales en VHDL

Autor: Alex Vladimir Pilatasig Escobar
Tutor: Hipólito Guzmán

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo fin de Máster
Ingeniería Electrónica, robótica y automática

Generador Automático de Drivers y Monitores para la verificación de circuitos digitales en VHDL

Autor:
Alex Vladimir Pilatasig Escobar

Tutor:
Hipólito Guzmán

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Trabajo fin de máster: Generador Automático de Drivers y Monitores para la verificación de circuitos digitales en VHDL

Autor: Alex Vladimir Pilatasig Escobar

Tutor: Hipólito Guzmán

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Agradezco a Dios y a la Santísima Virgen María por cuidarme y protegerme a lo largo de toda mi vida y haberme dado habilidades para superar los obstáculos que se me presentaron en la vida y de mi carrera estudiantil.

Agradezco a mis padres y a mis hermanas por todo su apoyo incondicional en todos los ámbitos de mi vida y haberme dado la oportunidad de tener una educación de excelencia en el transcurso de mi vida. Especialmente quiero agradecer a mi tío, mi tía y mi prima quienes me acogieron en su hogar mientras mi estancia universitaria, me dieron su apoyo sentimental y moralmente Gracias por ser las mejores familias del mundo.

Agradezco por el apoyo, el tiempo dedicado y por los conocimientos brindados por parte de todos mis maestros, especialmente agradezco al profesor Hipólito Guzmán Tutor de mi trabajo de titulación de fin de Máster por toda la colaboración brindada a lo largo de este proyecto.

Alex Vladimir Pilatasig escobar

Sevilla, 2020

Resumen

En este trabajo se desarrolla una herramienta la cual genere de manera automática los módulos Driver y Monitor en VHDL de diferentes circuitos electrónicos, para de esta manera tener la capacidad de verificar dichos diseños mediante la arquitectura de un testbench TLM (modelado a nivel de transacciones).

Lo primero en realizar es buscar las formas de onda de los circuitos digitales, ahora mediante la aplicación de wavedrom dibujamos estas gráficas y las guardamos por defecto esta aplicación las guarda en formato JSON, con lo cual se puede añadir más información a esta transacción. Esta información adicional es la encarga de especificar de mejor manera el funcionamiento de los circuitos digitales a diseñar.

Posteriormente se desarrolla un algoritmo que sea capaz de leer estos datos y generar los módulos Driver y Monitor del circuito en VHDL. Para esto se utiliza el lenguaje de programación Python el cual cuenta con librerías que ayudan a integrar los datos de la aplicación wavedrom en la base de datos de Python en forma de diccionarios. El algoritmo que se desarrollo es capaz de tomar los cambios de estado que se observan en las formas de onda de cada señal y convertirlos a un circuito en VHDL denominado Driver, Adicionalmente a esto se crea un módulo el cual usara las formas de onda generadas por el Driver para verificar el funcionamiento del circuito y comprobar que los datos enviados sean los correctos.

Finalmente se tiene los módulos VHDL tanto del Driver como del Monitor del circuito deseado y se procede a realizar la simulación de estos y así verificar el correcto funcionamiento, para esto se utiliza el software Xilinx en cual se desarrolla los testbench para cada uno de los módulos y se realiza su simulación, con lo que se verifica visualmente que efectivamente los circuitos cumplen con las especificaciones y funcionan de manera correcta.

Abstract

In this work, a tool is developed which automatically generates the Driver and Monitor modules in VHDL of different electronic circuits, in order to have the ability to verify said designs through the architecture of a TLM testbench (transaction level modeling) .

The first thing to do is to look for the waveforms of the digital circuits, now through the Wavedrom application we draw these graphs and save them by default this application saves them in JSON format, with which more information can be added to this transaction. This additional information is responsible for specifying in a better way the operation of the digital circuits to be designed.

Subsequently, an algorithm is developed that can read this data and generating the Driver and Monitor modules of the circuit in VHDL. For this, the Python programming language is used, which has libraries that help to integrate the data from the Wavedrom application into the Python database in the form of dictionaries. The algorithm that was developed can take the changes of state that are observed in the waveforms of each signal and converting them to a VHDL circuit called a Driver. In addition to this, a module is created which will use the waveforms generated by the Driver to verify the operation of the circuit and verify that the data sent is correct.

Finally, we have the VHDL modules for both the Driver and the Monitor of the desired circuit and proceed to simulate them and thus verify the correct operation, for this the Xilinx software is used in which the testbench is developed for each of the modules and their simulation is carried out, with which it is visually verified that the circuits actually comply with the specifications and work correctly.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

:	Asignación de un valor en JSON
“ ”	Definir una cadena de caracteres en JSON
<=	Asignar un valor a una variable en VHDL
=	Igualar 2 variables
[]	Definir una lista
{}	Definir un Diccionario

1 INTRODUCCIÓN

La persistencia es muy importante. No debes renunciar al menos que te veas obligado a renunciar.

- Elon Musk -

En el presente capitulo se define los objetivos que se pretende alcanzar y la justificación para el desarrollo de este proyecto, en donde se explica además el contexto de los HDL (lenguajes de descripción hardware), en concreto VHDL en el diseño y verificación de drives y Monitores para protocolos de comunicación, se indica también las fases del desarrollo del proyecto y la estructura del documento.

1.1. Motivación

En la actualidad el uso de los lenguajes de descripción hardware para diseñar circuitos electrónicos y digitales de gran complejidad se está convirtiendo en una tarea más necesaria e imperativa, esto se debe a que hoy en día existen circuitos integrados muy potentes y de altísimas prestaciones que están compuestos por millones de transistores y los lenguajes de descripción hardware son capaces de describir de una manera sencilla y muy precisa del comportamiento de estos circuitos, de esta manera se facilita el diseño de estos circuitos.

El Ingeniero Gordon Moore cofundador de INTEL, por el año 1965 fue director de los laboratorios Fairchild semiconductor, es ahí en los auges de la microelectrónica donde observo una tendencia en su fabricación de los circuitos integrados, por lo tanto predijo que cada año se duplicara el número de transistores por unidad de superficie de circuitos integrados, a pesar que 10 años después en 1975 módico su ley a que los transistores se duplicaran cada 2 años, desde entonces a esta ley se la conoce como la ley de Moore.[1]

Hoy en día la ley de Moore sigue efectuando su predicción, lo que conlleva a que cada día se fabrica y se presenta circuitos integrados mucho más potentes y con mejores prestaciones que sus predecesores, esto nos lleva a que la capacidad de diseño y por consiguiente la capacidad de verificación de estos circuitos también aumente, sin embargo la capacidad de diseño crece más lenta que la de fabricación debido a que se necesita muchas más personas diseñando, con respecto a la capacidad de verificación su crecimiento es aún menor ya que en la verificación se requiere el doble de tiempo del usado en el diseño de los circuitos.

Es por eso, por lo que se requiere de softwares y técnicas automatizas que faciliten el diseño y posteriormente simplifiquen la verificación de los circuitos integrados, he aquí la importancia del proyecto la cual es crear un programa digital que genere los Drivers y Monitores en lenguaje VHDL de diferentes tipos de protocolos de comunicación y así agilizar el proceso de diseño y verificación.

1.2. Objetivos

Para el desarrollo del trabajo de fin de máster aquí propuesto se establecen algunos objetivos los cuales se indican a continuación:

- Elaboración de las transacciones para cada protocolo de comunicación en formato JSON.
- Diseño de un algoritmo capaz de crear los Drivers VHDL de diferentes protocolos de comunicación.
- Diseño de un algoritmo capaz de crear los Monitores VHDL de diferentes protocolos de comunicación.
- Diseño de un algoritmo capaz de crear los package VHDL de diferentes protocolos de comunicación.
- Diseño de un algoritmo capaz de crear el archivo SVG para cada protocolo de comunicación.
- Elaboración de los testbench para los Drivers creados.
- Elaboración de los testbench para los Monitores creados.
- Simulación tanto de los Drivers como de los Monitores VHDL.
- Verificación del correcto funcionamiento de los protocolos de comunicación creados en VHDL.

1.3. Justificación

Debido a la veracidad de la predicción de la ley de Moore en la actualidad la capacidad de fabricación crece a pasos agigantados en comparación con la capacidad de diseño y verificación, cada día se fabrican circuitos integrados mucho más potentes y de mejores prestaciones por tal motivo se requiere de mucho más tiempo y más personas para la etapa de diseño de estos circuitos además la etapa de verificación es aún más importante y conlleva un tiempo mayor aproximadamente el doble de tiempo que consume la etapa de diseño.

Por tal motivo se produce una gran brecha entre la habilidad de fabricación, diseño y verificación. En la siguiente imagen se puede apreciar de mejor manera la evolución de cada habilidad, se crean un par de brechas de déficit como es el Design GAP el cual es el retraso que tiene la habilidad de diseño con respecto a la fabricación, la siguiente brecha es el Verification GAP que es retraso de la habilidad de verificación con respecto a la fabricación. Entonces tanto la evolución del diseño como la verificación de circuitos integrados está muy por debajo de la capacidad de fabricación que hoy en día se tiene.

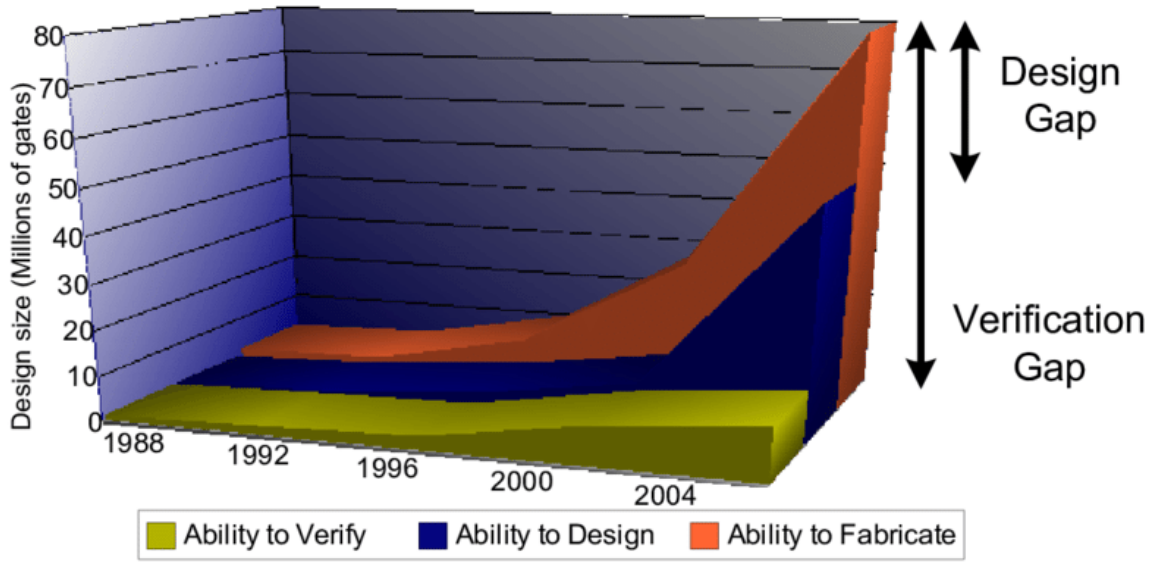


Figura 1-1. Design GAP y Verification GAP [1]

Hoy en día no existe ninguna herramienta que facilite o algún software que realice el diseño y verificación de protocolos de comunicación en lenguaje de descripción hardware, para realizar un protocolo de comunicación con HDL lo que comúnmente se hace en la actualidad es describirlos a mano e irlos diseñando con algún lenguaje de descripción hardware para posteriormente realizar su simulación y verificar su correcto funcionamiento introduciendo a mano diferentes tipos de estímulos.

Por lo descrito anteriormente se plantea realizar un algoritmo que genere de forma automática tanto los package como los drives y los Monitores en lenguaje VHDL de algunos protocolos de comunicación, para esto se requiere definir algunos detalles como son las transacciones de cada protocolo para lo cual se utilizara el formato JSON, el cual es un formato jerárquico para almacenamiento de datos. Para la elaboración del algoritmo se utiliza el lenguaje de programación Python porque hoy en día es uno de los lenguajes más utilizados y además cuenta con un sinfín de librerías relacionadas con los HDL, con lo cual hace que este proyecto tenga mucha escalabilidad en futuros trabajos. También se utiliza wavedrom el cual es un software encargado de renderizar la información del archivo JSON para que sea compatible con Python.

La metodología de verificación que se utiliza para el proyecto es el modelado a nivel de transacciones, con esto se trata de elevar el nivel de abstracción para la verificación, esto se realiza disgregando los datos que se mueven por las interfaces de los movimientos de pines y señales de control. Debido a esta metodología se requiere que el algoritmo genere tanto Drivers como Monitores para cada protocolo y así verificar su correcto funcionamiento.

Finalmente se requiere de la creación de los testbench para los Drivers y los Monitores, para esto se utiliza el software Xilinx, en el cual se puede diseñar y cambiar muy fácilmente las transacciones para las pruebas, además el mismo software permite la simulación de toda la metodología de verificación que es la integración tanto del Driver como del Monitor.

1.4. Fases de desarrollo

En este apartado se describe las fases para el adecuado desarrollo del presente trabajo de fin de máster, el cual se compone principalmente de 3 fases que se las detalla a continuación:

- Fase 1: Planificación
    - Estudio del funcionamiento de diferentes protocolos de comunicación.
    - Estudio del proceso de verificación en VHDL
    - Estudio de la sintaxis y diseño de los Drivers, Monitores y package en VHDL
    - Estudio de un formato de almacenamiento de datos para la elaboración de las transacciones.
    - Estudio de un lenguaje de programación para el diseño del algoritmo y además sea compatible con el formato de las transacciones.
- Fase 2: Desarrollo
    - Diseño de las transacciones para cada protocolo de comunicación.
    - Diseño del algoritmo que genere los Drivers, Monitores y package.
    - Elaboración de los testbench para los Drivers y Monitores generados.
    - Simulación de los archivos VHDL
    - Verificación del adecuado funcionamiento de los protocolos.
- Fase 3: Documentación
    - Elaboración de la memoria del TFM
    - Elaboración de la presentación

1.5. Estructura del documento

A continuación, se pretende describir de manera rápida y concisa la estructura del presente documento, indicando a que hace referencia cada uno de sus capítulos:

Capítulo 1:

Introducción. – En este capítulo se describe de manera rápida el contexto del trabajo de fin de máster, en donde se indican apartados como la motivación, objetivos y la justificación para el desarrollo del proyecto, además se muestra las fases de desarrollo y la estructura del documento.

Capítulo 2:

Estado del arte y Solución propuesta. – En este capítulo se describe los antecedentes de las metodologías aplicadas en el proceso de verificación y la forma en la que actualmente se realizan los Drivers y Monitores para los protocolos de comunicación, además se presenta la solución propuesta que es en si el proyecto desarrollado en el TFM, se indica la descripción y el alcance que el proyecto tiene.

Capítulo 3:

Metodología. - En el presente capitulo se detalla el marco teórico y los principios fundamentales que requiere el proyecto para su correcto análisis y posteriormente elaboración. Se explique puntos fundamentales como son los lenguajes de descripción hardware, el modelado a nivel de transacción con la explicación de cada uno de sus componentes y los protocolos de comunicación que se utilizaran para la verificación del correcto funcionamiento del algoritmo.

Capítulo 4:

Análisis. – En este capítulo se presenta los diferentes formatos de texto, lenguajes de programación y software que se utilizan para el desarrollo del proyecto, como son el formato JSON para describir el protocolo de comunicación, formato SVG con el cual se observara de forma gráfica del protocolo de comunicación, wavedrom que renderiza el formato JSON para que sea compatible con Python el cual es el lenguaje de programación utilizado para el diseño del algoritmo de generación de los Driver VHDL y el Monitor VHDL, se describe el software Xilinx en el cual se diseña los test-bench tanto para los Drivers y Monitores, además en el mismo software se simula y verifica el correcto funcionamiento de los protocolos de comunicación.

Capítulo 5:

Diseño y funcionamiento. – En este capítulo se explica el diseño de las transacciones que son la descripción de los protocolos de comunicación en formato JSON, el diseño del algoritmo el cual es descrito con el lenguaje de programación Python y se describe como se genera cada uno de los elementos requeridos, finalmente se explica el funcionamiento del algoritmo.

Capítulo 6:

Pruebas y resultados. - En el presente capitulo se indican los diferentes protocolos de comunicación que se utilizan para la verificación del algoritmo, aquí se describe la transición que se utiliza para cada protocolo. A partir de esto se muestra los diferentes resultados obtenidos para cada uno de los protocolos y se explica si el funcionamiento es el que se desea obtener y es el adecuado.

Capítulo 7:

Conclusiones y líneas futuras. – En este capítulo se indican las conclusiones que se obtuvieron con la elaboración de este proyecto, además se describen posibles trabajos futuros o la continuación del mismo proyecto con el propósito de validar su uso para más protocolos de comunicación.

2 ESTADO DEL ARTE Y SOLUCIÓN

Si quieres encontrar los secretos del universo, piensa en términos de energía, frecuencia y vibración

- Nikola Tesla -

EN el presente capitulo se estudia el ámbito en el cual se encuentra el presente trabajo de fin de máster. Primeramente, se analiza el estado del arte del proyecto lo que conlleva estudiar las distintas metodologías de verificación y las alternativas o métodos que existen en la actualidad para desarrollar Drivers y Monitores para protocolos de comunicación. Posteriormente se presenta la solución que proponemos en este proyecto en donde se indica la descripción y el alcance que tendrá este proyecto.

2.1. Estado del arte

En el apartado de este capítulo se hace una pequeña revisión de lo que son las metodologías de verificación y cuál es el estado en que se encuentra en ámbitos de Investigación y desarrollo. Además, se indica cual es el método de verificación que en la actualidad se utiliza para la verificación de los protocolos de comunicación en lenguaje VHDL, se explica también la forma en la que hoy en día se realizan los Drivers y Monitores para los protocolos de comunicación.

2.1.1 Metodologías de verificación

Debido a la gran evolución que ha tenido el proceso de fabricación de circuitos digitales los procesos tanto de diseño como de verificación han tenido la necesidad de crear nuevas técnicas y métodos para estar al nivel que el proceso de fabricación demanda.

En los últimos años el proceso de verificación ha ido tomando gran relevancia con los lenguajes de descripción hardware ya que se han ido implementando diversas metodologías de verificación. Cuyo diseño está basado en una arquitectura modular para que se acople a los testbench y a los estímulos que son necesarios en la verificación de circuitos digitales con lenguajes de descripción hardware. La utilización del lenguaje VHDL es de gran utilidad tanto para la etapa de diseño como para la etapa de verificación de los circuitos digitales ya que este lenguaje permite integrar construcciones en diseño RTL que para la etapa de verificación es muy importante ya que es un nivel de abstracción mucho mayor que en el diseño estructural. Este tipo de diseño es siempre sintetizable es decir que se puede crear los circuitos esquemáticos equivalentes, a continuación, se muestra algunas de las metodologías de verificación más utilizadas y cómo fue su evolución hasta la metodología más avanzada que hoy en día existe.[1]

Las primeras metodologías que se desarrollaron estaban escritas para el lenguaje Verilog, sin embargo, por el esfuerzo adicional que conllevaba verificar con VHDL, en el año 2016 se desarrolla una librería de código abierto que mitiga estas deficiencias.

2.1.1.1 VMM (Verification Methodology Manual)

Esta es la primera metodología exitosa, la cual es ampliamente implementada y utilizada para la creación de entornos de verificación reutilizables, esta metodología fue creada por synopsys y especialmente fue diseñada para trabajar con SystemVerilog. Esta metodología de verificación utiliza las características del lenguaje como es la programación orientada a objetos, restricciones, aleatorización y con la cobertura funcional con lo cual tanto principiantes como expertos tienen la capacidad de elaborar entornos de verificación de muy alto nivel. La contribución de esta metodología de verificación es un pilar fundamental en la creación de la metodología de verificación UVM.[2]

2.1.1.2 OVM (Open Verification Methodology)

Esta metodología fue desarrollada en el año 2008 por Mentor graphics y Cadence, está basada en código abierto. Es una biblioteca de procedimientos y objetos, para la elaboración de pruebas aleatorias y dirigidas mediante la generación de estímulos, recolección de datos y control en el proceso de la verificación utilizando cobertura funcional y comunicación a nivel de transacción. Al igual que la biblioteca anteriormente mencionada esta es una de las principales bases para la elaboración de la metodología UVM.[2]

2.1.1.3 UVM (Universal Verification Methodology)

En el año 2010 a partir de las 2 metodologías anteriormente mencionadas se desarrolla la metodología UVM, la cual es una biblioteca basada en la semántica y sintaxis de System Verilog y creada en base a código libre, mediante esta librería es posible crear elementos de verificación reutilizables y flexibles, además la creación de banco de pruebas muy potentes que se pueda implementar en múltiples proyectos.

Esta metodología de verificación proporciona utilidades tan básicas y genéricas como son la jerarquización de componentes, modelamiento a nivel de transacciones, bases de datos configurables y funciones para la automatización de datos. También posee la etapa de abstracción en la cual cada parte del entorno de verificación tiene su respectiva función de trabajo, por ejemplo, el Driver se encarga de enviar señales al diseño mientras que el Monitor se únicamente de Monitorizar el correcto funcionamiento de este diseño.[3]

2.1.1.4 Verificación IP

A partir de todas estas metodologías mencionadas anterior y especialmente de la metodología UVM se desarrolla la metodología de verificación IP, sin embargo, esta metodología ya no se basa en código abierto lo que conlleva a que para su distribución y utilización se requiere del pago de licencias costosas. Pese a lo anteriormente mencionado esta metodología es muy utilizada debido al aumento de la complejidad de diseño en los sistemas embebidos y en los System on chip, ya que lo que la verificación IP hace es dividir el diseño en pequeños bloques mucho más simples y reutilizables y conectarlos entre sí.

La verificación IP (VIP) funciona mediante la utilización de bloques de descripción de hardware en los cuales se busca normalizar el diseño de los circuitos más usados como son sumadores, memorias, registros etc. Por lo tanto, estos bloques son reutilizables ya que cuentan con un alto grado de confiabilidad, llevándolos a que se tenga un notable aceleramiento en el desarrollo de los entornos de verificación y así reducir el tiempo que se consume en el proceso de verificación de los circuitos digitales. Esta metodología posee un vasto catálogo de posibilidades de verificación por lo tanto muchas empresas han desarrollado sus softwares de verificación IP a continuación se detalla algunas de ellas.[4]

- **Cadence. -**Esta empresa posee un software de verificación IP muy basto y potente ya que consta con alrededor de 60 interfaces de memoria y 40 protocolos de comunicación, pero la principal ventaja es que permite la utilización de algunos lenguajes de programación como son: VHDL, Verilog, System Verilog e incluso permite el uso de C++.
- **Synopsys VC. -** Esta metodología está desarrollada en el lenguaje System Verilog y está fundamentada en la metodología de verificación UVM, su módulo de conexión es totalmente configurable y posee bloques o modelos de verificación a nivel de sistema de algunos de los más importantes protocolos de comunicación como son: USB, FLASH, DRAM, etc.

- **VIP de Mentor. -** Es muy utilizada para la verificación de diseños avanzados, con la cual se puede implementar bancos de pruebas en diferentes lenguajes de los más importantes que son VHDL y Verilog, además esta metodología de mentor es la que ofrece una arquitectura de lo más similar a UVM.

2.1.1.5 UVVM (Universal VHDL Verification Methodology)

Las primeras metodologías mencionadas se diseñaron específicamente para trabajar con el lenguaje de descripción Verilog y no es hasta el 2010 cuando se desarrolla UVM donde se integran una metodología más flexible y reutilizable, además en base a esta metodología se desarrolla la verificación IP en la cual ya se puede verificar diseños que estén desarrollados en VHDL el inconveniente es que la verificación IP no está basada en código abierto con lo cual para su utilización es necesario la compra de costosas licencias, en el año 2016 se desarrolla UVVM la cual es una metodología de verificación universal para VHDL que está basada en código abierto por lo tanto es gratuita y de libre distribución y utilización, con la que se puede elaborar bancos de pruebas muy estructurados y en lenguaje VHDL.

Algunos de los principales aspectos que UVVM maneja es la capacidad de mantenimiento, la reutilización de sus diseños, la legibilidad, la extensibilidad entre otros, consta además de un conjunto muy basto de elementos que son: un marco VVC que hace referencia a los componentes de verificación VHDL, BFM que son los modelos funcionales de bus, estas 2 elementos VVC y BFM se pueden combinar y utilizar en conjunto para la verificación de cualquier diseño, también posee un extensa biblioteca de utilidades. En el caso de los protocolos de comunicación esta metodología posee una amplia librería de protocolos algunos de ellos son UART, SPI, I2C, etc. Y está conformada por 2 partes fundamentales que se indican a continuación.[5]

- **UVVM Utility Library. –** Es una librería la cual está conformada por funciones básicas para la utilización de diferentes testbench en VHDL, con lo cual el tiempo de desarrollo de los bancos de pruebas es mucho más ágil y además con un buen control de alertas y Monitorización. lo más importante que en esta librería se integra el soporte para los BFM.
- **UVVM VVC. –** Esto es la verificación de componentes VHDL, esta parte de UVVM permite el diseño de bancos de pruebas mucho más avanzados y estructurados, con lo cual se verifica diseños mucho más grandes y con muchos estímulos.

Gracias a la elaboración de todas estas metodologías anteriormente mencionadas se ha logrado estandarizar los modelos de verificación y la elaboración de los bancos de pruebas, pese a las características de calidad y eficiencia que ofrece UVVM la cual es la metodología más avanzada y la que integra las mejores cualidades y lenguajes de descripción de sus metodologías predecesoras, aún le falta integrar más componentes y protocolos de verificación. Sin embargo, para la elaboración de protocolos para aplicaciones más específicas se termina desarrollando tanto los Drivers como los Monitores a mano.

2.1.2 Alternativas existentes

En la actualidad para el diseño y verificación de protocolos de comunicación se realiza haciendo a mano el diseño y luego utilizando UVVM para su verificación si es el caso de un protocolo estándar. Sin embargo, en muchas de las ocasiones se requiere modificar los protocolos básicos o diseñar protocolos específicos para aplicaciones concretas, en estos casos se realiza el diseño y la verificación a mano.

Lo primero que se realiza es buscar la forma de onda del protocolo que se desea implementar o dibujar las formas de onda en papel o en algún software de acuerdo con las necesidades y requerimientos que la aplicación necesite, lo más común es utilizar wavedrom esta aplicación permite dibujar las formas de onda de los protocolos a manera de un plan de pruebas, en donde se describe lo que se va a simular y genera las formas de onda que se espera como resultado de la simulación.

En la siguiente ilustración se muestra como ejemplo las formas de onda del protocolo I2C una vez que se tiene estas formas de onda con la ayuda de algún software de descripción de hardware, se va diseñando la transacción y el movimiento de pines que tiene que realizar el dispositivo con el propósito de obtener las formas de onda esperadas.

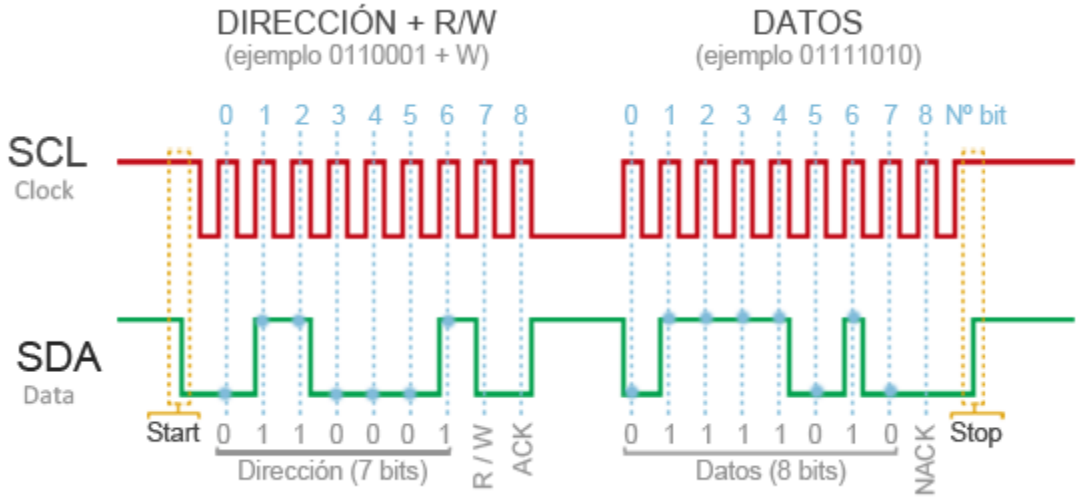


Figura 2-1. Formas de onda Protocolo I2C [17]

Una vez que se tiene diseñado el VHDL del protocolo de comunicación, se procede a realizar la verificación de su correcto funcionamiento para ello se escoge una metodología de verificación, la más comúnmente utilizada es la verificación a base del modelado a nivel de transacciones. Para eso en la siguiente figura se detalla cómo es el proceso de verificación y las partes que esta metodología requiere.

Una vez que se tiene diseñado el protocolo en VHDL que se lo denomina Driver, se procede a desarrollar el Monitor para dicho Driver, la ventaja de trabajar con transacciones al momento de realizar los planes de pruebas es que aumenta el nivel de abstracción por lo tanto el diseñador no tiene que preocuparse de todos los aspectos de las señales a lo largo de la verificación.

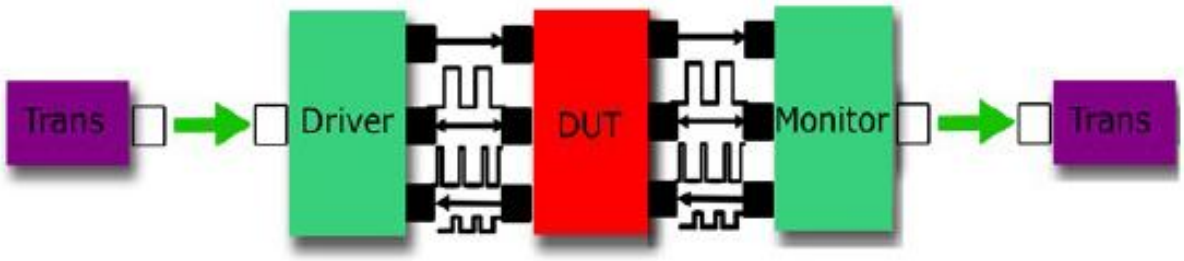


Figura 2-2. Testbench modelado a nivel de transacciones [15]

El proceso de funcionamiento de verificación de este método es el siguiente una vez que se ingresa la transacción del protocolo, el Driver genera el movimiento de pines a partir de la transacción ingresada, luego de esto las señales ingresan a la DUT (device under test) y esta una vez que responde pasa al Monitor, el cual toma estos movimientos de pines y los transforma a transacciones. De esta manera se puede verificar el correcto funcionamiento del protocolo realizado en VHDL, en el cual se ingresa una transacción y como resultado al final de este proceso de verificación se obtiene una transacción de salida la cual tiene que ser la misma que la de entrada y así se verifica que el protocolo funcione de manera correcta.

2.2. Solución propuesta

En vista de las carencias de algunos protocolos en las metodologías de verificación y que la forma de diseño y verificación tradicional es a mano, lo cual conlleva un esfuerzo adicional y mayor tiempo de trabajo se propone realizar el presente trabajo el cual se lo detalla a continuación.

2.1.3 Descripción

En el presente trabajo se pretende desarrollar una aplicación o algoritmo el cual genere de forma automática los Drivers, Monitores y package VHDL de diferentes protocolos de comunicación y así diseñar y verificar el correcto funcionamiento de los protocolos que se desea implementar en alguna aplicación determinada.

Lo primero es generar las transacciones para ello a partir de la utilización del software wavedrom en el cual se diseña las formas de onda que se espera obtener como resultado del protocolo que se desea implementar, se aprovecha que el formato de funcionamiento que utiliza wavedrom es JSON en el cual se puede ingresar las transacciones de manera jerarquizada y además Python posee una librería para renderizar los datos JSON de wavedrom con lo cual dicha transacción ya se puede integrar en Python.

Por lo tanto, el lenguaje de programación que se utilizara para la elaboración del algoritmo de generación de los Drivers y Monitores es Python ya que posee las librerías necesarias para el acoplamiento con wavedrom, entonces primeramente se diseña en wavedrom las formas de onda que se espera obtener del protocolo a implementar, una vez que se tiene este archivo JSON se lo integra a Python y se procede a la elaboración tanto de los Drivers, de los protocolos y de los package VHDL además el algoritmo también genera la forma de onda que tendrá el protocolo con lo cual se verifica la correcta implementación del protocolo.

Una vez que se tiene los VHDL de los protocolos, con la ayuda de software Xilinx se procede a desarrollar los testbench para cada protocolo y posteriormente se realiza la simulación de estos así se verifica el adecuado funcionamiento de cada protocolo de comunicación. Al tener un algoritmo completo de generación Driver y Monitor se consigue obtener varias características y ventajas que no se tienen cuando se realiza el diseño manualmente.

- **Mínimo Error. –** Al generar de forma automática tanto los Drivers, como los protocolos y los package VHDL se elimina el error de sintaxis que se produce al momento de diseñar de forma manual los protocolos en VHDL.
- **Sincronización. –** La sincronización queda resuelta ya que al diseñar las formas de ondas del protocolo en wavedrom se asegura que el resultado que se obtendrá en VHDL será el que se desea obtener.
- **Tiempo. –** Como el algoritmo genera de manera inmediata los Drivers, los Monitores y los package VHDL se reduce el tiempo de elaboración de estos con respecto al diseño tradicional que se lo realiza a mano.
- **Escalable. – S**e puede crear nuevos protocolos o combinar 2 o más protocolos y así obtener uno más avanzado y robusto. Es se lo realiza de manera inmediata basta con modificar las formas de onda de la transacción que se realiza en wavedrom.
- **Modificable. –** Se puede modificar la forma de onda de la transacción del protocolo y de manera inmediata el algoritmo genera los nuevos Drivers, Monitores y package VHDL, con esto se puede crear y verificar muchos protocolos en menos tiempo.

La filosofía de este proyecto es mediante un solo algoritmo de generación se ingresen diferentes transacciones de acuerdo con el protocolo que se desea implementar. Para ello se utiliza el formato de trabajo que utiliza wavedrom para crear sus formas de ondas y luego a partir de esto integrarlo a Python con la ayuda de sus librerías y poder generar los protocolos en VHDL. Para luego con la ayuda de los ficheros VHDL generados poder verificar el correcto funcionamiento de cada protocolo de comunicación.

2.1.4 Alcance

El objetivo de este proyecto no es competir con las grandes metodologías de verificación que existen en el mercado, ni tampoco el crear un algoritmo de generación solo para 1 o 2 protocolos de comunicación específicos. El objetivo de este proyecto es generar un sistema completo de verificación Driver – Monitor con el cual mediante el diseño de las formas de onda en la transacción el algoritmo de generación sea capaz de crear los Drivers, Monitores y package VHDL para diferentes protocolos de comunicación.

A un futuro se pretende que se implementen en el código de generación de los VHDL todas las funcionalidades que tiene el software wavedrom para la elaboración de las formas de onda y así tener un campo más amplio para la elaboración de más protocolos de comunicación. Además, con la integración de todas las funciones de wavedrom en el algoritmo se podrá desarrollar nuevos protocolos que se requieran de acuerdo con el campo de aplicación que se los utilizará.

Las estructuras para la elaboración de los módulos VHDL estarán definidas en el algoritmo por lo tanto la continuación de este trabajo para un futuro es la de integrar las funciones del lenguaje Verilog a este algoritmo y así poder generar de manera simultánea los módulos Driver y Monitor en estos 2 lenguajes VHDL y Verilog.

3 METODOLOGÍA

Mi padre me explicó que la educación y el conocimiento es lo que les permitirá a los niños mejorar el mundo

- Steve Wozniak-

EN el presente capitulo se detalla y explica los conocimientos fundamentales que se requieren para la realización de este trabajo, como son: una descripción de lo que son los HDL y algunos de sus lenguajes de programación más conocidos. También se aborda el TLM que es el modelado a nivel de transacción utilizado para la verificación funcional en circuitos digitales y como es el funcionamiento y modo utilización de los protocolos de comunicación.

3.1 Hardware Description Languages (HDL)

Cada día se construyen microprocesadores más avanzados y de mayores prestaciones los cuales están compuestos por miles e incluso millones de unidades lógicas funcionales, por tal motivo la necesidad de diseñar y elaborar circuitos digitales más complejos es notoria, como ya lo predijo Moore cada año la capacidad de fabricación aumentara, por lo tanto, la capacidad de diseño también tiene que buscar formas y alternativas para crecer es ahí que nacen los HDL o lenguajes de descripción hardware y ya con ellos se puede diseñar circuitos digitales más complejos y de mayores prestaciones.

Alrededor de los años setenta ocurre una evolución en la fabricación de los circuitos integrados, es ahí en donde aparece la tecnología MOS en concreto y la más utilizada la NMOS, las cuales contribuyeron al desarrollo y fabricación de los circuitos digitales hasta los años 80. A partir de este año el departamento de defensa de los estados unidos empieza con un proyecto al cual denominaron VHSIC que hace referencia a los circuitos integrados de muy alta velocidad, con el cual se trataba de desarrollar un lenguaje que describiera a los circuitos digitales de una mejor manera y ayude a resolver el problema que se tenía de modificar el hardware que se tenía para una aplicación y poder utilizarlo en otra. Entonces se contrata a 3 ingenieros de las empresas IBM, Intermetrics y Texas Instrument para el desarrollo de un lenguaje de descripción hardware estándar el cual la IEEE en el año 1984 publica el primer estándar, el cual después con la colaboración de universidades y otras empresas se logró desarrollar más funcionalidades para que finalmente en 1987 la IEEE publique el estándar STD 1976-1987 el cual es el punto de partida de los HDL.

Los HDL son lenguajes de alto nivel, los cuales cuentan con un estándar y sintaxis definida con lo cual se facilita el diseño y modelamiento de los circuitos digitales, gracias a esta estandarización se puede describir los circuitos a diferentes niveles de abstracción. En la siguiente figura se indica los niveles de abstracción que existen para el hardware como para el software.

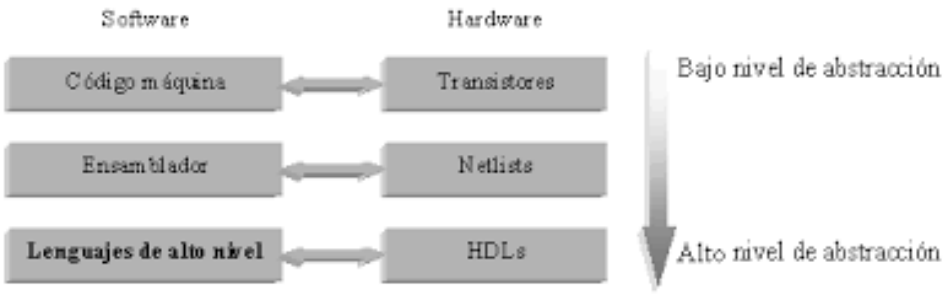


Figura 3-1. Niveles de abstracción [12]

En la figura 3-1 se puede apreciar los diferentes niveles de abstracción, con lo cual se observa que los HDL tienen un alto nivel de abstracción, algunas de las características más importantes que estos lenguajes poseen se las detallan a continuación:

- **HDL Multinivel. –** es decir estos lenguajes posee un rango muy amplio de utilización de los posibles niveles de especificación que tenga el circuito digital y puede combinar estos niveles en una única descripción.
- **Fácil lectura y comprensión. –** son lenguajes que tienen una extensa biblioteca de documentación por lo tanto su lectura y comprensión de la forma de diseño de los circuitos digitales se hace muy fácil y rápida.
- **descripción de componentes. –** Gracias a las bibliotecas y al ser lenguajes de descripción de alto nivel la descripción de los diferentes componentes de los circuitos digitales se simplifica mucho y su integración en la simulación es muy rápida.
- **Universalidad. –** estos lenguajes son muy compatibles con la mayoría de las herramientas de automatización de diseño, por lo tanto, son utilizados por muchas empresas de diseño y proveedores de software.

Estos lenguajes pueden ser sintetizados en diferentes librerías por lo tanto es muy independiente a la tecnología que se utiliza para su funcionamiento, esto hace que estos diseños sean reutilizables y la portabilidad hacia otra tecnología sea más ágil y sencilla, esto hace que se pueda utilizar un diseño en componentes muy diferentes como pueden ser una FPGA y ASIC utilizando un mínimo esfuerzo de diseño para cambiarlas entre ellas.

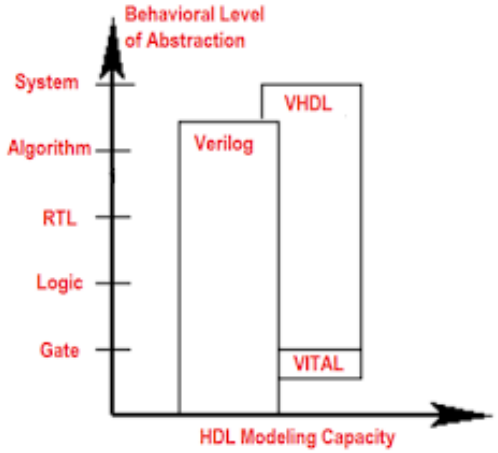


Figura 3-2. Capacidad de modelado en HDL [12]

Existen diferentes lenguajes para describir los HDL, los 2 lenguajes más importantes se indican en la figura 3-2, los cuales son Verilog y VHDL. En dicha figura se indica la capacidad de modelado HDL que tienen estos 2 lenguajes y los diferentes niveles de abstracción que tienen cada uno de ellos en el diseño de los circuitos digitales.

3.1.1 Verilog

Es un lenguaje de descripción hardware que permite realizar el modelamiento de circuitos electrónicos tanto digitales como analógicos y señales mixtas. Verilog aparte del diseño también permite la prueba e implementación de los circuitos a diferentes niveles de abstracción.

Este lenguaje fue inventado y desarrollado por Phil Moorby por el año de 1985 cuando trabajaba en la empresa Automated Integrated Design System, al momento que se desarrollaba este lenguaje se lo hacía como un lenguaje de modelo de hardware. Por el año de 1990 Cadence compra la empresa anteriormente mencionada quedándose así con los derechos de los simuladores de Verilog. Gracias a esto el mercado de Verilog creció muchísimo en la década de los 90 sin embargo en esta misma época VHDL empieza a crecer, lo cual obligo a Cadence hacer que su lenguaje sea de código abierto y de libre distribución, luego para estandarizar este lenguaje se lo envía a la IEEE, quienes en el año 1995 saque el estándar para Verilog denominado STD IEEEE 1364-1995, el cual luego fue llamado Verilog 95.

Esta versión de Verilog poseía algunas deficiencias por lo cual se enviaron algunas extensiones a la IEEE, para que finalmente se obtuviera un nuevo estándar de este lenguaje denominado estándar IEEE 1364-2001 al cual se le dio el nombre de Verilog 2001. En esta versión de Verilog se incluyeron algunas nuevas funcionalidades como la generación de declaraciones, variables automáticas y arreglos multidimensionales, hoy en día una gran mayoría de diseño de circuitos electrónicos trabajan con este estándar. Sin embargo, en el año 2005 la IEEE saca un nuevo estándar el cual presenta leves cambios con respecto a la anterior versión. En este mismo año se lanza al mercado el estándar System Verilog el cual además de ser utilizado como lenguaje de diseño también se lo utiliza como lenguaje de verificación, por lo tanto, este lenguaje se agrega a la clasificación de los HDVL que significa lenguaje de verificación y descripción de hardware.

Cuando se estaba diseñando Verilog se trataba de que este lenguaje tenga una sintaxis de programación muy similar a la de C, para que de esta manera los ingenieros o desarrolladores de los circuitos electrónicos se familiaricen muy rápido con este nuevo lenguaje, haciendo que sea más fácil empezar a utilizar este lenguaje de descripción y que tuviera mayor aceptación en comparación a otros lenguajes descripción hardware existentes en el mercado.

El diseño de un circuito en Verilog está basado en una jerarquía de módulos, estos módulos por puertos IN, OUT o INOUT que hace referencia a puertos bidireccionales entrada – salida. Posee sentencia secuenciales y concurrentes las cuales definen el comportamiento del módulo y son las encargadas de describir la relación que existen entre los puertos, registros y líneas de conexión. Las sentencias secuencias son utilizadas dentro de los bloques de diseño y funcionan secuencialmente, mientras que las sentencias concurrentes funcionan de forma paralela. Otra funcionalidad es que se puede definir un módulo que este dentro de otro modulo y así definir un sub-comportamiento.

3.1.2 VHDL

Es un lenguaje de descripción hardware cuyas siglas significan lenguajes de descripción de hardware de circuitos integrados de alta velocidad, este lenguaje se empezó a desarrollar por el año 1980 por el departamento de defensa de los estados unidos quienes iniciaron un proyecto denominado VHSIC en el cual se buscaba desarrollar circuitos integrados de muy altas prestaciones en tecnologías de 0.5 micras, con este proyecto se vio la necesidad de desarrollar un lenguaje de descripción hardware. Entonces se pone en marcha una segunda etapa del proyecto VHSIC en el cual se desarrolló lo que en la actualidad se lo conoce como VHDL.

El desarrollo de este lenguaje empezó por parte del departamento de defensa de los estados unidos, quienes luego con la integración de empresas como IBM, Texas Instrument y Intermetrics a este proyecto lograron que la IEEE empezara con la estandarización de este lenguaje el cual en el año 1987 se aprueba su primer estándar con el número de registro 1076. A partir de este momento diversas empresas empiezan a sacar compiladores y simuladores al mercado, luego de esto empiezan aparecer las herramientas de síntesis con lo cual la IEEE publica una revisión de este estándar y lo llama IEEE STD 1076-1993 y la cual es la versión que en la actualidad se utiliza en el diseño de circuitos digitales.

Los VHDL poseen una serie de características, una de sus principales ventajas es que es basado en código libre y de libre distribución aparte de esta posee otras características que se las enlista a continuación:

- Diseño en alto nivel
- Alto nivel de abstracción
- Verificación
- Síntesis
- Simulación
- Control temporal
- Descripción estructural
- Integración de subprogramas

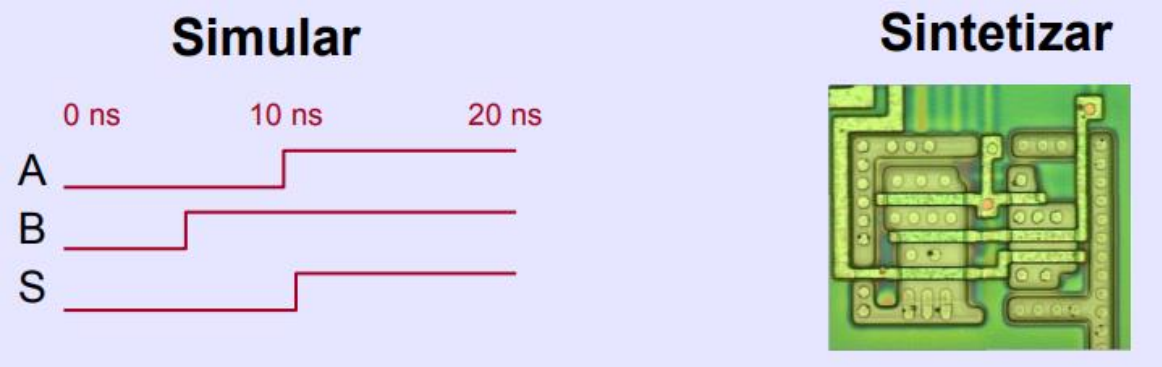


Figura 3-3. Características de VHDL [7]

En la figura 3 – 3 se indica algunas de las características con las que cuenta VHDL como son la capacidad de sintetizar, esto hace referencia a la creación de un circuito que funcione de la misma manera que se diseñó el modelo con el leguaje de descripción hardware utilizado. Otra de las características es la capacidad de simular esto quiere decir que el modelo tiene la funcionalidad deseada, para esto se genera formas de onda del modelo descrito en VHDL y así observar si cumple con especificaciones que se requiere para la aplicación en la que se desea utilizar.

VHDL no es un lenguaje de programación, este es un lenguaje de descripción hardware que permite el diseño de circuitos digitales tanto síncronos como asíncronos esto implica que no se utiliza la misma terminología que en la programación normal en VHDL se tienen otro tipo de componentes como son las puertas y los biestables en lugar de variables y funciones, etc.

La utilización de estos lenguajes en el diseño de circuitos digitales es muy importante ya que con ellos se puede tener algunas ventajas, las cuales se indican a continuación.

- Debido al gran número de fabricación de circuitos integrados se requiere de diseños mucho más avanzados y de mejores prestaciones por lo cual es imperativo tener herramientas que trabajen con el computador es ahí donde los VHDL nos facilitan el diseño de estos circuitos.
- Con los VHDL se puede descubrir fallas en diseño antes que se los implemente de manera real en las aplicaciones a ser usados.
- Son reutilizables y permiten que más de una persona trabajen con estos modelos sobre el mismo proyecto o en otro diferente.

Lo más importante de los VHDL es que se puede realizar la descripción de la estructura de los circuitos así también como la descripción de la funcionalidad del mismo circuito utilizando métodos similares a los lenguajes de programación convencionales.

3.2 Transaction-Level Modeling (TLM)

Es modelo consigue elevar el nivel de abstracción en la verificación de circuitos digitales ya que separa los datos que se envían por la interfaz del modelo de las señales de control y de los movimientos de los pines del circuito. Con esto se trata de desarrollar de una manera independiente por una parte la funcionalidad del circuito y por otra la comunicación de este. [6]

La principal utilización de este modelo es que facilita la transferencia de datos o secuencias de control con diferentes componentes que estén funcionando dentro de la misma aplicación. El nivel de abstracción que presenta este modelo ha sido de mucha ayuda y es por eso por lo que este modelo ha ido tomando cierta envergadura al momento de la verificación de circuitos digitales. Esto es debido a la complejidad que cada día los circuitos digitales van teniendo, es fundamental reducir el esfuerzo de diseño y el tiempo que se utiliza entre el diseño y verificación.

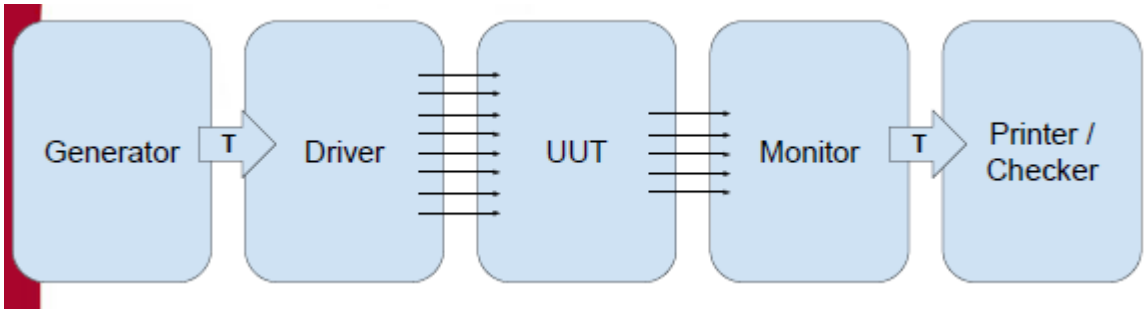


Figura 3-4. Arquitectura de un testbench TLM [7]

En la figura 3 – 4 se presenta la arquitectura del testbench que tiene el modelado a nivel de transacciones, se puede observar las partes que esta metodología de verificación tiene y su funcionamiento es el siguiente. Primeramente, se generan las transacciones las cuales ingresan al Driver el cual convierte estas transacciones en movimiento de pines y los pasa a la DUT y cuando este módulo responde a las señales ingresadas, un módulo denominado Monitor convierte estos movimientos de pines en transacciones las cuales se pueden observar en el módulo checker y verificar si la transacción de salida es similar a la transacción de entrada y así es como esta metodología verifica el correcto funcionamiento de los circuitos digitales.

3.2.1 Transacciones

La principal ventaja de utilizar transacciones para la verificación es que se puede elevar el nivel de abstracción en el diseño y verificación de los circuitos digitales y así el diseñador no tienen que preocuparse de todos los detalles que tiene la señal. En la transacción se encapsula todos los datos que tiene el modelo como por ejemplo una transacción simple se presenta a continuación

write (data,address)

En esta transacción de ejemplo se describe un circuito para escritura la cual cuenta con una dirección de memoria y el dato que se desea escribir en dicha dirección. Para el proyecto se utilizará el formato de texto JSON el cual es un formato de jerarquización de datos en este formato se describirá los diferentes parámetros que cada protocolo de comunicación tendrá.

3.2.1.1 Package VHDL

Este es un archivo con un tipo de dato compuesto que contiene los datos de la transacción. Una vez que el generador reciba la transacción este generara el package, el cual tendrá el formato de un VHDL el cual describe un **récord** y contendrá los datos de cada protocolo ya sea dirección de memoria, dirección de dispositivos, el dato que se requiere enviar etc. Todo dependerá del tipo de protocolo que se desea implementar.

3.2.2 Movimiento de pines

El movimiento de pines es el cambio que las señales sufren, estos cambios pueden ser de nivel alto a bajo y viceversa otro tipo de movimiento es él envió de un dato por la línea o la búsqueda de una dirección de memoria como se puede apreciar en la figura 3-5, estos son algunos de los cambios que pueden ocurrir en una línea de señal.

Una vez que el módulo Driver reciba la transacción está la convierte en movimiento de pines de acuerdo a los datos que la transición contenga, como por ejemplo la transacción presentada anteriormente produce el movimiento de pines que se presenta en la figura 3 – 5, en donde se puede observar cómo se mueven y cambian de estados las diferentes líneas con el propósito de realizar la acción que la transacción contenga, en este caso la línea cs y la wren cambian en el momento en el que la acción de las líneas de data y address se activan.[7]

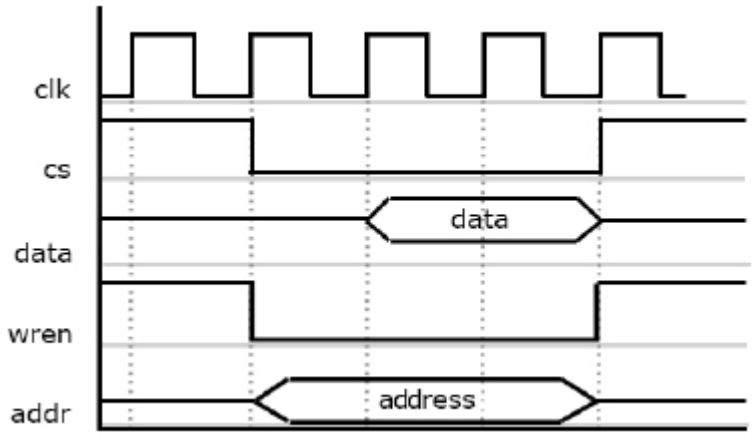


Figura 3-5. Movimiento de pines [15]

3.2.3 Driver VHDL

Este es un modelo descrito en lenguaje VHDL, es el encargo de recibir la transición la cual es una instrucción que tiene que realizar el dispositivo a probar. Entonces este módulo recibe esa transacción y la convierte a movimiento de pines de acuerdo con el propósito de la transacción, esto lo hace mediante el cambio de estados de los pines que tiene asociado el dispositivo. En el caso de este proyecto la transacción que se da es él envió de un paquete de datos por lo tanto el Driver toma esta transacción y la transforma en el movimiento de pines de acuerdo con el tipo de funcionamiento del protocolo que se esté utilizando.

3.2.4 DUT

Este es el dispositivo al cual se le está poniendo a prueba. Es el dispositivo al cual el Driver envía el movimiento de pines que se genera esto se lo realiza con el propósito de realizar la acción que la transacción requiere realizar. En el caso de la transacción mencionada que se ingresó con anterioridad al Driver, la DUT recibe el movimiento de pines generado con el propósito de enviar un paquete de datos.

3.2.5 Monitor VHDL

Este modelo esta descrito en VHDL, es el encargado de censar los movimientos de pines que entran y salen a la DUT. Toman la información más relevante, la encapsulan y la envía en forma de transacción por la salida de este módulo. En el ejemplo que se tomó en cuenta tanto para el Driver como para la DUT, lo que hace el Monitor es tomar el movimiento de pines producido y lo traduce a datos los cuales tienen que ser o mejor dicho son los datos de transmisión que se ingresan en la transacción.

3.3 Protocolos de comunicación

Los protocolos de comunicación son medios por los cuales se realiza en envió de datos o paquetes de datos de una forma ordenada, sincronizada y siguiendo ciertos pasos, se conoce el dispositivo al cual se le va a enviar los datos y el momento en el cual se está realizando la transmisión. Existe un sinfín de protocolos de comunicación de acuerdo con la aplicación en la cual se los quiere utilizar, pueden ser tanto sincrónicos como asincrónicos a continuación se detalla algunos de los importantes y los que más se utiliza en la comunicación de datos.

Los protocolos definen como tiene que ser la comunicación entre diferentes dispositivos, para realizar esta comunicación se tiene que seguir un conjunto de reglas con los cuales se puede asegurar el correcto envió y recepción de los datos y así asegurar el formato y la secuencia con la cual se van a transmitir los datos por la línea de señal.

Estos protocolos se mencionan debido a que son unos de más conocidos y usados en la comunicación, además tiene diferentes cambios de estado y formas de onda, por lo tanto, son ideales para la verificación del correcto funcionamiento del algoritmo que se va a desarrollar.

3.3.1 UART

Este protocolo realiza una comunicación serie y es asíncrona como lo indica es su nombre el cual es: transmisor receptor asíncrono universal. Este protocolo necesita de una sola línea para la transmisión de los datos, por cada tramo de transmisión envía un byte de información lo cual quiere decir que por cada vez se activa este protocolo enviar 8 bits de datos, los cuales siguen una secuencia especifica la cual está determinada por el protocolo, la velocidad de transmisión normal que este protocolo utiliza es de 9600 baudios, que quiere decir que el tiempo aproximado en transmitir un bit es de 104 microsegundos.

Este protocolo transmite los datos de forma serie lo cual quiere decir que para la transmisión de datos se requiere de una sola línea y los datos se van enviando uno detrás de otro a diferencia que la comunicación en paralelo la cual envía sus datos de forma simultánea por diferentes líneas. Al utilizar la comunicación en serie se tiene una gran ventaja la cual es se eliminan el número de puertos y líneas que se utilizan para la transmisión ya que con la comunicación en serie se envía los datos por una sola línea.

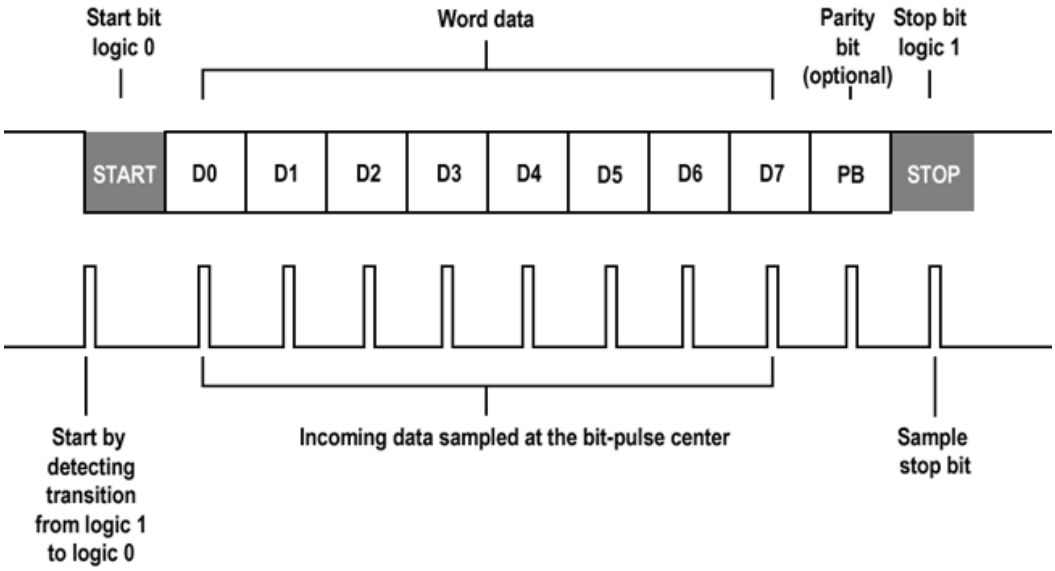


Figura 3-6. Protocolo UART [16]

Un protocolo quiere decir que se necesitan seguir un conjunto de reglas para poder realizar él envió de datos correctamente y el dispositivo de recepción de estos datos los reciba correctamente y los valores de los datos adecuados, en la figura 3 – 6 se presenta las formas de onda de este protocolo y las reglas y pasos que este necesita para realizar la transmisión de datos a continuación se describe estas reglas y como es el funcionamiento de este protocolo.

- **Start bit. –** Este es el bit de inicio con el cual este protocolo indica que se va a iniciar la transmisión de datos, normalmente la señal de la línea está en alto, pero si se quiere iniciar a transmitir un dato este bit se coloca en estado lógico bajo 0 y es el momento en el cual se inicia el funcionamiento de transmisión de datos de este protocolo.
- **Word Data . –** esta sección es la que lleva el paquete de datos que se desea enviar, normalmente por cada transmisión se envía 1 byte de datos, de los cuales se los envía bit a bit, la secuencia en que se las envía es en forma ordenada empezando por el bit menos significativo, por la tanto la transmisión empezara desde el bit 0 e ira hasta el bit 7.
- **Parity bit. –** Este es el bit de paridad el cual consiste en tomar todos los datos transmitidos previamente por la línea y de acuerdo con estos datos se enviar un bit adicional con un estado lógico, para asegurarse de la correcta recepción del paquete de datos. Existen 2 tipos de bit de paridad los cuales se lo indica a continuación.

Tabla 3-1. Bit de paridad

DATOS	SUMA DE BITS	BP PAR	BP IMPAR
10101011	5	1	0
0111001	4	0	1
010011	3	1	0
0110	2	0	1

- **Stop Bit . –** Este es el bit que indica la finalización de la transmisión de este protocolo, en el cual para indicar que ha terminado la transmisión esta se pone a un valor lógico alto 1 y si termina el funcionamiento de este protocolo.

3.3.2 I2C

Este protocolo de comunicación se denomina inter Integrated circuit, se lo desarrollo por el año 1982 por la empresa Philips, este es un protocolo de comunicación serie y síncrono lo que quiere decir que para comunicarse y enviar los datos hacia otro dispositivo se requiere de una señal de reloj y así sincronizar todos los módulos, tiene gran aplicación en la industria ya que es muy utilizado para la comunicación en microcontroladores y sus puertos en sistemas embebidos

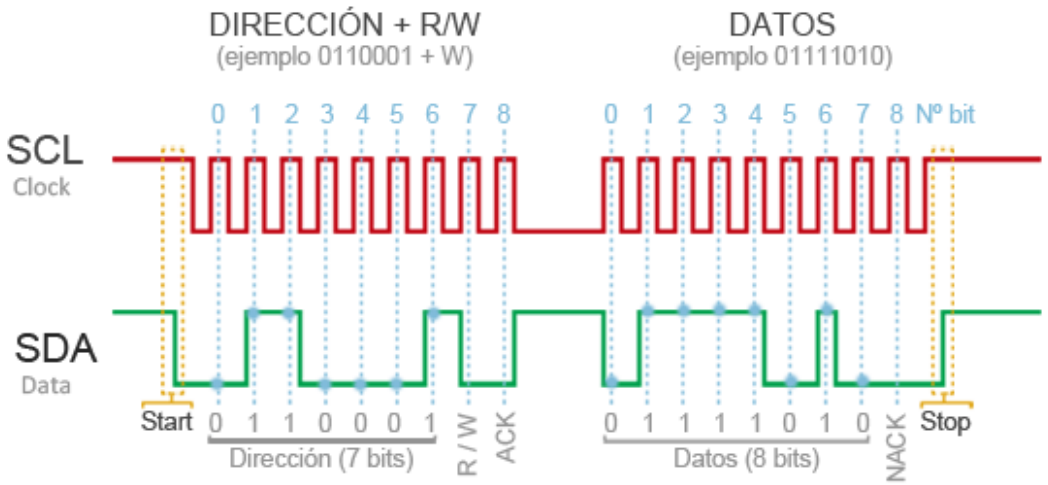


Figura 3-7. Protocolo I2C [17]

Para el funcionamiento de este protocolo solo se necesita de dos líneas una denominada **scl** la cual es la encargada de transmitir la señal de reloj con la cual se sincronizan los dispositivos que intervienen en la transmisión, la otra línea se denomina **sda** y es la encargada de transmitir tanto la dirección del dispositivo y de registro y el paquete de datos y otras funciones que este dispositivo requiere para su funcionamiento. Estas líneas son bidireccionales esto quiere decir que funcionan como entrada o salida de datos con lo cual se reduce el número de líneas necesarias para la transmisión, pero se incrementa la complejidad al enviar y recibir datos por las mismas líneas.

Este protocolo envía los datos bit a bit y normalmente tanto los datos como las direcciones son enviadas desde el bit más significativo, las direcciones de los dispositivos conectados a este bus varían con un tamaño de 7 a 10 bits, con lo cual se pueden conectar hasta 128 dispositivos en el mismo bus y cada uno de ellos puede ser tanto maestro como esclavo. En la figura 3 – 7 se puede observar el funcionamiento y la forma de transmisión que utiliza este protocolo el cual se lo detalla a continuación.

Scl. – Esta señal es generada por el dispositivo maestro y es una señal de reloj que este protocolo utiliza para sincronizarse con el dispositivo esclavo con el que se desea establecer una comunicación, esta no es más que una señal que va cambiando de estados lógicos entre alto y bajo a una determinada velocidad la cual establece el dispositivo maestro

Sda. - Esta señal es la encargada de transmitir la dirección del dispositivo con el cual se desea establecer la comunicación, también es la que envía el paquete de datos y algunas funcionalidades que este protocolo requiere para su correcto funcionamiento, a continuación, se describe la secuencia de funcionamiento que este protocolo utiliza para la transmisión de datos.

- **Start. –** Este es el bit de inicio con el cual este protocolo indica que se va a iniciar la transmisión de datos, normalmente la señal de la línea está en alto, pero si se quiere iniciar a transmitir un dato este bit se coloca en estado lógico bajo 0 y es el momento en el cual se inicia el funcionamiento de transmisión de datos de este protocolo.
- **dirección. –** En esta etapa se envían la dirección del dispositivo con el cual se desea establecer la comunicación, esta dirección puede tener una longitud de entre 7 a 10 bits, la dirección se envía bit a bit empezando por el bit más significativo
- **R/W. –** Después de enviar la dirección del dispositivo se envía este bit el cual determina la acción a realizar la cual puede ser de lectura o escritura. Si la acción es escritura se envía un valor lógico bajo, pero si la acción es de lectura se envía un valor lógico alto.
- **ACK/NACK. –** Luego de lo anterior descrito se envía este bit al cual se lo conoce como bit de reconocimiento, este bit lo envía el esclavo y es el que determina si la dirección se ha recibido correctamente, cuando la dirección se recibe correctamente se envía un bit en estado lógico bajo 0 al cual se denomina ACK y En caso que no se recibiera la dirección se envía un bit con estado lógico alto que se lo denomina NACK y la transmisión termina.

Después de lo descrito anteriormente en ocasiones se suele enviar la dirección de registro del dispositivo donde se va a leer o escribir el dato, se envía bit a bit una dirección de 8 bits empezando por el bit más significativo, luego de esto el esclavo envía el bit de reconocimiento dependiendo si la dirección se ha recibido correctamente se envía un valor lógico alto o bajo.

- **Datos. –** Ahora se envía el paquete de datos, normalmente este protocolo envía un byte de datos, que son 8 bits. Él envió se lo realiza bit a bit empezando por el bit más significativo.
- **ACK/NACK. –** Luego de lo anterior descrito se envía este bit al cual se lo conoce como bit de reconocimiento, este bit lo envía el esclavo y es el que determina si la dirección se ha recibido correctamente, cuando la dirección se recibe correctamente se envía un bit en estado lógico bajo 0 al cual se denomina ACK y En caso que no se recibiera la dirección se envía un bit con estado lógico alto que se lo denomina NACK y la transmisión termina.
- **Stop. –** Este es el bit que indica la finalización de la transmisión de este protocolo, en el cual para indicar que ha terminado la transmisión esta se pone a un valor lógico alto 1 y si termina el funcionamiento de este protocolo.

Hay que tomar en cuenta algunas consideraciones importantes que este protocolo requiere para la transmisión de datos como son. El cambio de estado de la señal sda solo se puede realizar cuando la señal scl se encuentre a nivel lógico bajo. Existe una excepción a esta regla la cual es en el inicio y fin de la transmisión de datos en donde los cambios de estado de la señal sda se producen cuando la señal scl se encuentra a nivel lógico alto, en la figura 3- 8 se puede apreciar de mejor manera esta regla descrita y se observa como es el cambio de estado al inicio y final de la transmisión.

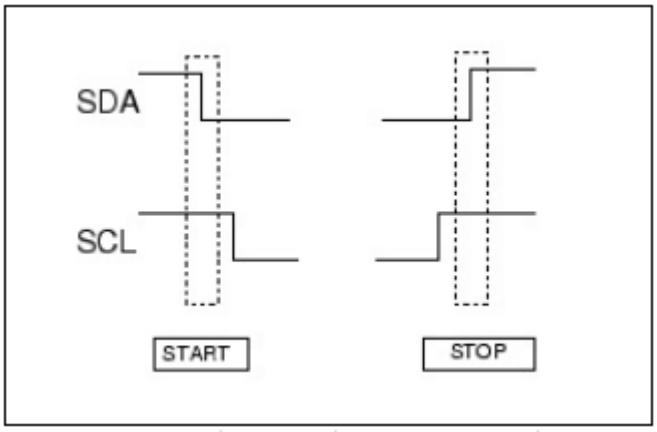


Figura 3-8. Condiciones de Inicio y parada del protocolo I2C [17]

3.3.3 SPI

Este protocolo se denomina interfaz periférica serie, el cual fue desarrollado por el 1980 por la empresa Motorola y es comúnmente usado para la transferencia de datos en circuitos integrados, este es un protocolo de comunicación serie y síncrono lo que quiere decir que para comunicarse y enviar los datos hacia otro dispositivo se requiere de una señal de reloj y así sincronizar todos los módulos.

Para su funcionamiento este protocolo requiere de 4 líneas las cuales son: la línea **sclk** es la encargada de generar la señal de reloj, es utilizada para la sincronización con los demás dispositivos, esta señal la genera el dispositivo maestro y es la activa el funcionamiento de las demás líneas, la siguiente señal se denomina **mosi** es la señal que transmite los datos desde el maestro hacia es esclavo , la siguientes señal se denomina **miso** es la que transmite los datos desde el dispositivo esclavo hacia el dispositivo maestro y finalmente la señal **ss** la cual es la encargada de seleccionar el dispositivo con el cual se está realizando la comunicación a continuación se describe el funcionamiento de este protocolo.

- **Sclk . –** Esta señal es generada por el dispositivo maestro y es una señal de reloj que este protocolo utiliza para sincronizarse con el dispositivo esclavo con el que se desea establecer una comunicación, esta no es más que una señal que va cambiando de estados lógicos entre alto y bajo a una determinada velocidad la cual establece el dispositivo maestro
- **Mosi . –** Esta señal es la encargada de transmitir los datos desde el dispositivo maestro al dispositivo esclavo, se transmiten los datos bit a bit empezando por el bit más significativo y por lo general tiene una longitud de 16 bits de datos, solo se envía los datos cuando la señal **ss** esta activada.
- **Miso . –** Esta señal es la encargada de transmitir los datos desde el dispositivo esclavo al dispositivo maestro, se transmiten los datos bit a bit empezando por el bit más significativo y por lo general tiene una longitud de 16 bits de datos, solo se envía los datos cuando la señal **ss** esta activada.
- **ss . –** Esta señal selecciona el dispositivo con el cual se desea realizar la comunicación, normalmente para que la señal se active se coloca a un valor lógico bajo y permanece activa durante todo el tiempo en el que se está realizando la comunicación con algún dispositivo.

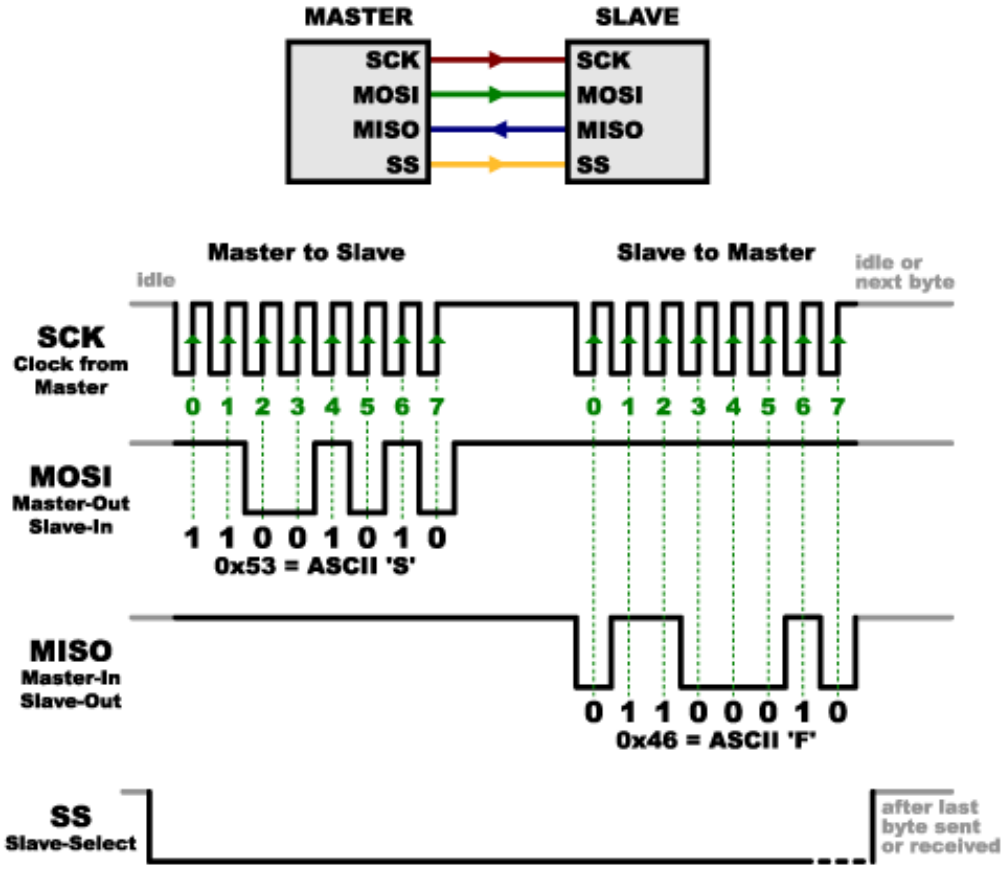


Figura 3-9. Protocolo SPI [17]

En la figura 3 – 9 se puede apreciar de forma gráfica como es el funcionamiento de este protocolo y como son los cambios de estado de cada una de las señales. Además, existen 4 modos de funcionamiento de este protocolo los cuales se indican en la figura 3 – 10, 2 de estos modos se lo realiza con CPOL la cual modifica la señal **sclk** haciendo que se invierta la señal, la otra variable es el CPHA la cual crea 2 modos de funcionamiento más haciendo que la señal de transmisión de datos miso y mosi se adelante un poco.

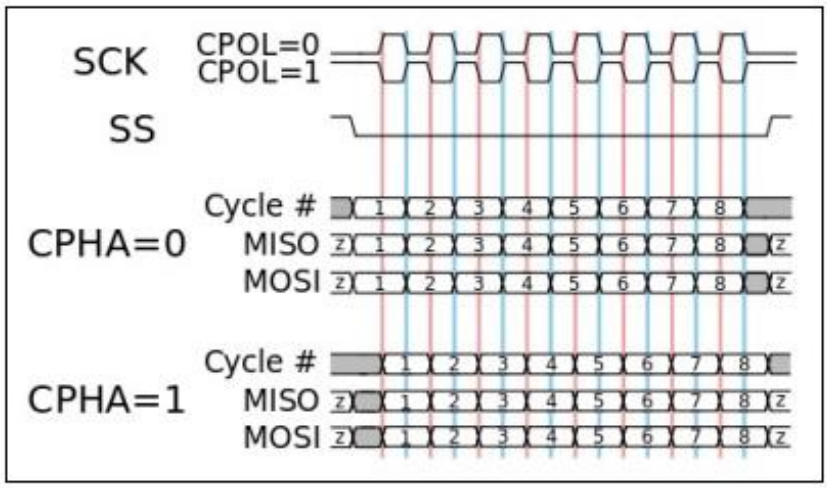


Figura 3-10. Modos de funcionamiento SPI [17]

3.3.4 Protocolo ficticio

Este protocolo se lo diseño modo de prueba para verificar el funcionamiento de este algoritmo, en el cual se envía paquetes en vectores y además se integra otras señales de utilidad para él envió de los datos.

- **Clk. –** Esta señal es generada por el dispositivo maestro y es una señal de reloj que este protocolo utiliza para sincronizarse con el dispositivo esclavo con el que se desea establecer una comunicación, esta no es más que una señal que va cambiando de estados lógicos entre alto y bajo a una determinada velocidad la cual establece el dispositivo maestro
- **Ena. –** Esta señal selecciona el dispositivo con el cual se desea realizar la comunicación, normalmente para que la señal se active se coloca a un valor lógico alto y permanece activa durante todo el tiempo en el que se está realizando la comunicación con algún dispositivo.
- **Startp . –** Este es el bit de inicio con el cual este protocolo indica que se va a iniciar la transmisión de datos, normalmente la señal de la línea está en bajo, pero si se quiere iniciar a transmitir un dato este bit se coloca en estado lógico alto 1 en un estado de tiempo y es el momento en el cual se inicia el funcionamiento de transmisión de datos de este protocolo.
- **Dout. –** esta es la línea por la cual se transmiten los datos, esta será una señal de tipo vector de 4 bits de longitud, por lo tanto, al momento se enviarán los datos en paquetes de 4 bits.
- **Endp. –** Este es el bit que indica la finalización de la transmisión de este protocolo, en el cual para indicar que ha terminado la transmisión esta se pone a un valor lógico alto 1 y si termina el funcionamiento de este protocolo.

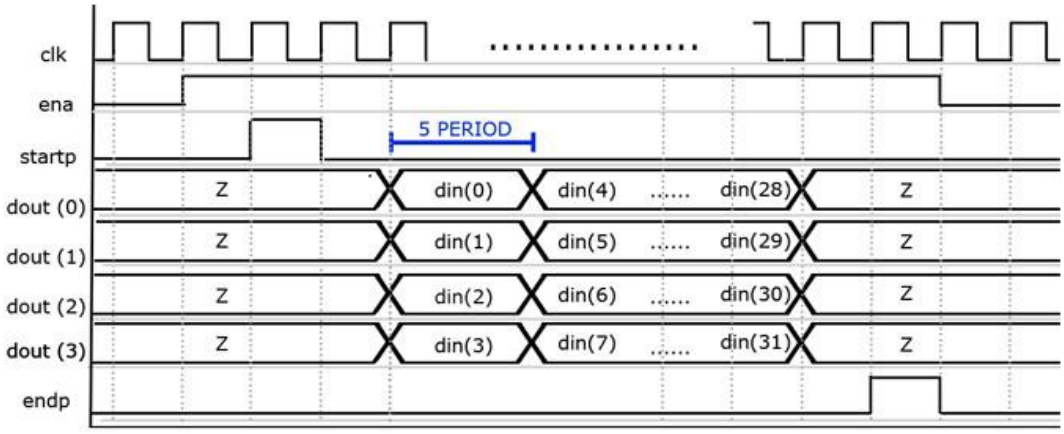


Figura 3-11. Protocolo Ficticio [15]

4 ANÁLISIS DEL SOFTWARE

El laboratorio de una fábrica es el mejor lugar para aprender sobre el fracaso

- Soichiro Honda -

EN este capítulo se presenta una revisión tanto de los formatos como de los programas informáticos que se han utilizado para el desarrollo de este proyecto, los formatos descritos son JSON y SVG en donde se describe la transacción y posteriormente se observa las formas de onda. Se detalla también el lenguaje de programación utilizado para desarrollar el algoritmo y se describe el software en el cual se realiza la simulación de los módulos VHDL.

4.1 Formato Json

Es un formato de texto en el que se presentan los datos estructuradamente utilizando la sintaxis de objetos de JavaScript de ahí viene el nombre de este formato JSON que significa JavaScript object notation. A pesar de que inicialmente se lo diseño para trabajar con JavaScript en la actualidad es muy utilizada por diferentes entornos de programación en donde ya se tiene la capacidad tanto de generar como de leer los datos que este formato tiene y con los cuales se puede trabajar.

El formato Json es una cadena, la cual tiene una similitud con los objetos de JavaScript, debido a esta similitud se puede describir en Json los mismos tipos de datos que se pueden describir en un objeto de JavaScript como son: números, cadenas, booleanos y arreglos, gracias a esto se puede describir los datos de una manera jerarquizada como la que se muestra en la figura 4 – 1.

```
1  {
2      "Tipo":"class",
3      "Nombre":"persona",
4      "Operaciones": ["consultar","insertar","actualizar"],
5      "Vistas":["PDF","XML","cargo"]
6  }
```

Figura 4-1. Formato JSON

Json nació como una alternativa a XML y es en el año 2019 cuando se lo define como un formato independiente del lenguaje con el que se desea utilizar, ya a partir de este año se puede utilizar y trabajar el formato Json en diferentes tipos de lenguajes de programación. Este formato es constituido por una colección de pares denominados nombre/valor, en donde se coloca un nombre cualquiera y a continuación se coloca el valor para dicho nombre este valor puede tomar diferentes valores que pueden ser cadenas, números, arreglos, etc.

Este es el formato en el que se describen las transacciones, existen varios formatos que ofrecen las mismas características que este formato tiene, sin embrago se utiliza este formato ya que la aplicación wavedrom utiliza este formato para describir los diagramas de tiempo digitales, además existe una librería la cual renderiza estos datos y los convierte a un diccionario de Python, con los cuales ya se puede trabajar con este lenguaje de programación.

4.2 Formato SVG

Los archivos en formato SVG son imágenes vectoriales, la principal ventaja de estas imágenes es que se pueden redimensionar y la memoria que utilizan para guardar sus datos es mucho menor que en las imágenes convenciones que se basan en mapas de bits como son (JPEG, PNG, etc.).

Se utiliza este formato para almacenar las imágenes de las formas de onda que produce la transacción ingresada, los SVG son utilizados por las características anteriormente mencionadas y además la librería wavedrom utilizada en Python trabaja con este formato. Con este archivo se verifica si la transacción ingresada cumple con las formas de onda y cambios de estado que se desea obtener.

4.3 Wavedrom

Esta aplicación grafica las formas de onda de las señales a partir de una descripción de su comportamiento mediante la utilización de las funciones y códigos con las que cuenta este software, tiene su propio lenguaje de descripción el cual esta descrito en formato Json.

Esta aplicación es utilizada para la realización del plan de pruebas de los circuitos digitales, en ella se describe el comportamiento que se desea obtener de algún circuito. Este es el punto de partida por el que se eligió utilizar esta aplicación ya que en esta se describe inicialmente la forma de onda que se desea obtener del diseño de un circuito en específico. Además, se provecha la funcionalidad de que esta aplicación utiliza el formato Json para describir el comportamiento de las señales en donde se pueden añadir datos adicionales que sirven para complementar la transacción y para que al algoritmo le sea más fácil desarrollar los módulos Driver y Monitor.

Cuenta con un motor de renderizado y además esta aplicación cuneta con una librería en Python con la cual se puede describir la forma de onda de un circuito y luego con la utilización de esta librería estos datos se pueden pasar a Python y trabajar con ellos en este lenguaje de programación, a continuación, en la figura 4 – 2 se las formas de onda que esta aplicación puede crear.

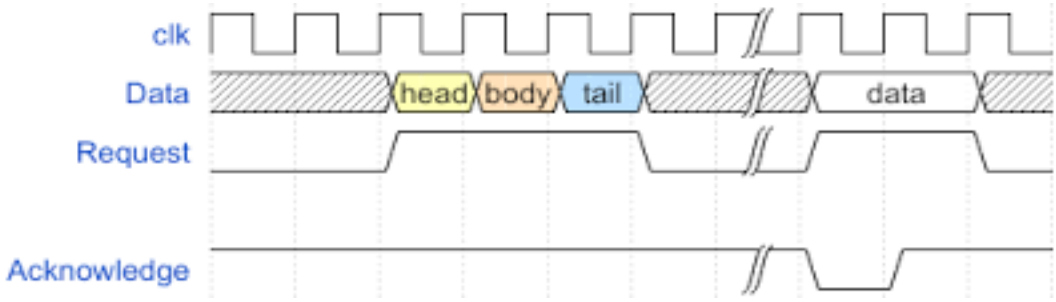


Figura 4-2. Ejemplo de Formas de ondas generadas por wavedrom [10]

4.4 Python

Es un potente lenguaje de programación muy utilizado por muchos programadores alrededor del mundo ya que es un lenguaje multiparadigma y multiplataforma, pero la principal razón por la que tantas personas lo utilizan es porque es desarrollado bajo código abierto, con lo cual se lo puede utilizar en cualquier ámbito y la comunidad desarrolladora puede ir subiendo y añadiendo sus librerías y trabajos creados con el propósito de seguir complementando este lenguaje de programación.

Python fue desarrollado a inicios de los años 90 en el centro de investigación de ciencias de la computación de la ciudad de Ámsterdam por un ingeniero holandés llamado Guido Van Rossum. Este lenguaje de programación es ideal para trabajar con grandes cantidades de información y con diferentes tipos de formatos por lo cual es muy utilizado por desarrolladores de big data. Una de las principales características de este lenguaje es que su sintaxis es muy limpia y legible, con lo que se puede diseñar algoritmos muy complejos con la utilización de pocas líneas de código.

Ventajas de utilizar Python

- Lenguaje limpio y ordenado
- Fácil programación
- Muy portable – se lo puede utilizar en cualquier sistema
- Flexibilidad de programación
- Simplificación y rapidez
- Gran comunidad desarrolladora

Para la realización de este trabajo se Elige este lenguaje por las características y ventajas anteriormente mencionadas, principalmente se lo escoge porque cuenta con la librería wavedrom con la cual se puede integrar los archivos para la generación de las formas de onda de wavedrom en el lenguaje de programación Python, y este renderiza estos datos y los convierte en un diccionario, el cual es un conjunto de tuplas y listas con los datos de los archivos de wavedrom. A partir de estos datos se diseña el algoritmo de generación de los módulos VHDL.

4.5 Xilinx

Es una herramienta Informática que permite el diseño y simulación de circuitos digitales. Así también como el diseño de circuitos mediante la utilización de los HDL. La empresa que proporciona este software lleva su mismo nombre y son muy conocidos por inventar e introducir al mercado las FPGA que significan arreglos de compuertas lógicas programables. En este proyecto esta herramienta es utilizada para la integración de todos los módulos creados y para su posterior simulación y verificación del correcto funcionamiento de los circuitos que se desea desarrollar. [8]

4.5.1 Xilinx ISE

Este es el software informático en donde se realiza el diseño de los circuitos ya sea mediante la utilización de los diferentes componentes que posee o mediante la descripción de los circuitos con el uso de HDL, este es el software que permite añadir los módulos VHDL creados y en donde se diseñan los módulos testbench para la posterior simulación tanto del Driver como del Monitor. En la figura 4 – 3 se observa la interfaz de este software y las ventanas que posee las cuales se enumera a continuación.

- Ventana de procesos
- Ventana de información
- Ventana de edición
- Ventana de ficheros fuente

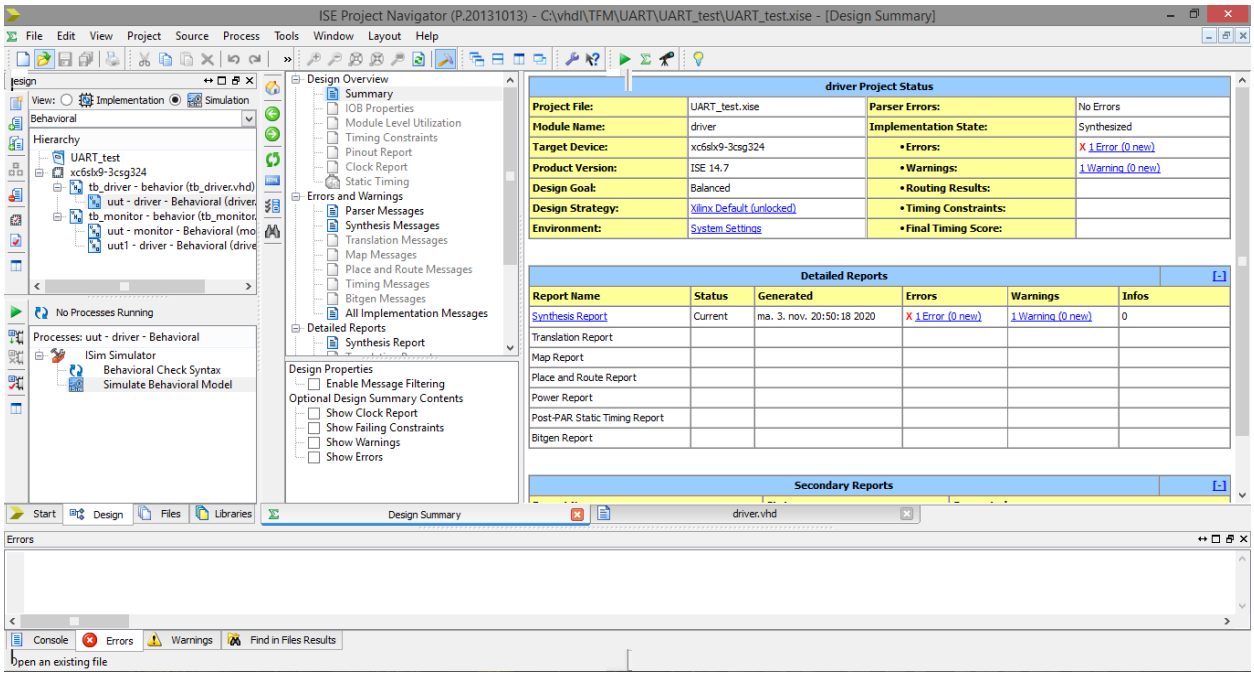


Figura 4-3. Interfaz Xilinx ISE

4.5.2 Xilinx ISIM

Este es el software informático el cual muestra la simulación de los módulos VHDL creados en el anterior software. Aquí se puede observar las formas de onda que genera el circuito diseñado en VHDL y los diferentes cambios de estado que este produce. En la figura 4 – 4 se muestra la interfaz gráfica que tiene el software Xilinx ISIM.

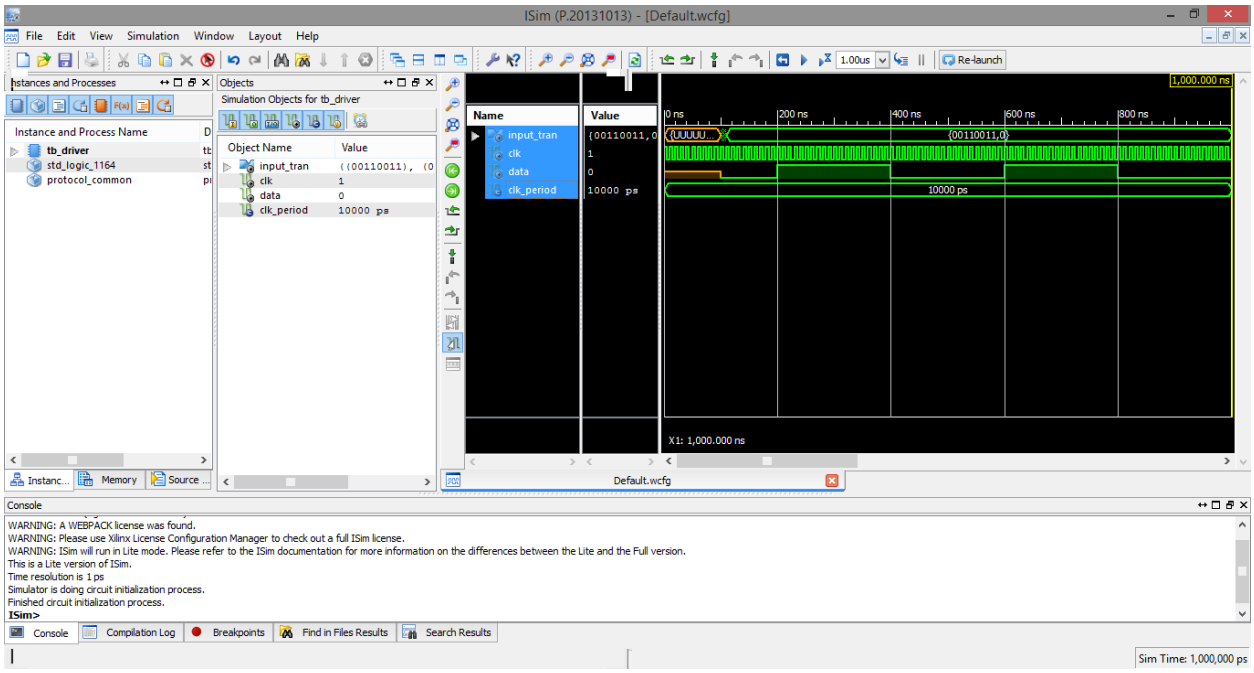


Figura 4-4. Interfaz Xilinx ISIM

5 DISEÑO Y FUNCIONAMIENTO

La mayoría de las personas gastan más tiempo y energía en rondar los problemas que en tratar de resolverlos

- Henry Ford-

EN este capítulo se detalla cómo se diseñó del algoritmo y las transacciones, además se revisa como es la forma en la que este algoritmo analiza las transacciones para crear los módulos VHDL que genera, finalmente se explica cómo es la forma de funcionamiento de este algoritmo.

5.1 Diseño

En este apartado se explica la forma en la que está estructurada las transacciones y como el algoritmo de generación las utiliza para realizar la creación del Driver, Monitor y package VHDL. Estos archivos son necesarios para la verificación del correcto funcionamiento del circuito digital. Por lo tanto, se explica el diseño de la transacción y las partes que esta lleva con el propósito de que el algoritmo tenga todas las herramientas necesarias para la creación de todos los módulos VDHL, posteriormente también se describe la forma en la que el algoritmo crea los diferentes módulos y archivos.

5.1.1 Transacción

Es un archivo en la que se encapsulan ciertos datos y especificaciones que posteriormente son utilizados por el algoritmo para generar los distintos módulos VHDL. Para esto se utiliza un formato en que se permita encapsular los datos de forma ordenada y jerarquizada, existen diversos formatos con estas características sin embargo se utiliza el formato JSON ya que es el formato que utiliza wavedrom para la elaboración de los planes de prueba y así graficar las formas de onda que se desean simular. La transacción estará compuesta por 2 partes las cuales se las describe a continuación:

Trans. –

Esta es la primera parte y es la encargada de describir y encapsular los datos que se desean enviar, es donde se ingresan las diferentes variables que encapsularan los datos a enviar en los modelos VHDL. Además, estos datos el algoritmo utiliza estos datos para la elaboración del módulo package VHDL.

Signal.-

En esta parte se describen los pines que contendrá el dispositivo y se describe las formas de onda que estos producirán al ponerse en funcionamiento. Utilizando las funcionalidades que wavedrom tiene se describe el comportamiento de las señales. Estos datos son los que utilizara el algoritmo para generar los modules Driver y Monitor.

En la siguiente tabla se muestran los comandos wavedrom que se utilizaron. Estos comandos se utilizan para generar las formas de onda que tienen las diferentes señales, además son los comandos que están integrados en el algoritmo de generación esto quiere decir que estos son los comandos que el algoritmo reconoce para la generación de los distintos módulos

Tabla 5-1. Valores Wavedrom

SIMBOLO	SIGNIFICADO O EQUIVALENCIA
H	Valor lógico alto
L	Valor lógico bajo
P	Señal de reloj positiva
N	Señal de reloj negativa
Z	Alta impedancia
X	Bit de paridad
3	bit de lectura o escritura
4	Bit de reconocimiento ACK/NACK
=	Señal para transferencia de bit de datos
.	Señal para extensión de la señal anterior

5.1.1.1 Archivo JSON

En este archivo se encapsulan todos los datos del circuito digital a ser diseñado y verificado, se utiliza este formato ya que es el que wavedrom utiliza para el desarrollo de los planes de prueba y para graficar las formas de onda de las señales que se va a implementar. Como se explicó anteriormente consta de 2 partes que son **trans** y **Signal.** A continuación, se muestra como ejemplo una de las transacciones utilizadas, en este caso es la transacción que describe el funcionamiento del protocolo de comunicación UART. Como se observa este formato describe los datos de una forma ordena y jerarquizada. Consta de diferentes datos de los cuales algunos son generales para todas las señales y otros que detallan la forma de las señales y el funcionamiento de manera más específica.

```
{
 "trans":
       [
         {"name": "data", "type":"vector","len": 8, "start": "LSB"}
       ],

 "signal":
       [
         {"name": "data", "wave": "l========xh", "type":"logic","len": 1,
         "ciclos": 10}
       ]
}
```

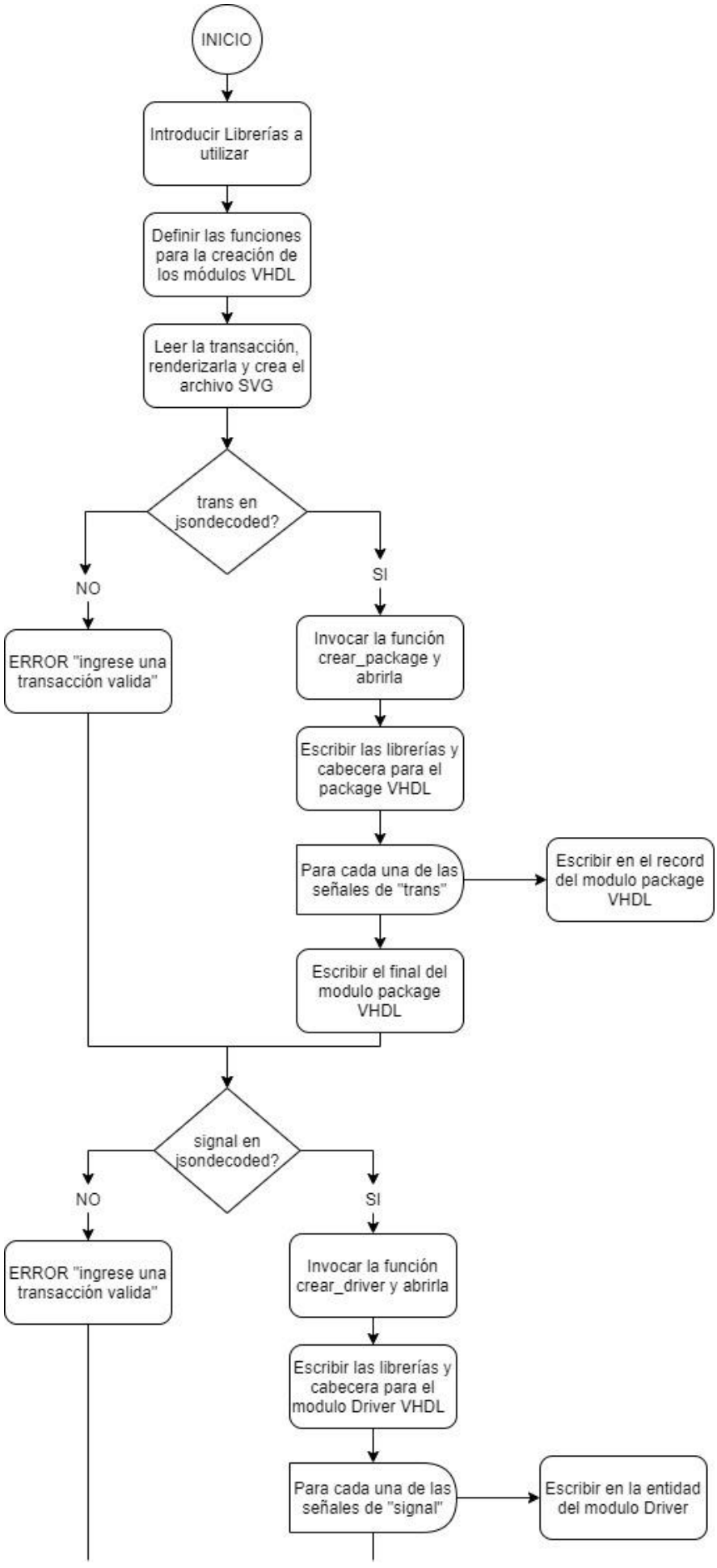
- **Name. –** En este apartado se describe el nombre que el Puerto o señal va a tener. Se puede asignar cualquier nombre.
- **Type. –** En este apartado se describe el tipo de las señales, existen cargados 2 tipos: **logic** para referirse a señales de tipo lógico que enviaran los datos bit a bit o cambian su estado lógico y **vector** quienes envían los datos en paquetes de bits de diferentes tamaños.
- **Len. –** En este apartado se describe la longitud que tendrá la señal por ejemplo si se trata de una señal de tipo lógico la longitud es de 1. Pero si es de tipo vector la longitud puede ser cualquier número mayor que 1.
- **Start. –** En este apartado se describe en caso de que la señal sea de tipo vector el orden en el que se envía los datos ya sea empezando por el menos significativo **LSB** o por el bit más significativo en donde se coloca la palabra **MSB**.
- **Wave. –** En este apartado se describe el comportamiento que tendrá la señal es decir con la utilización de los comandos mostrados en la tabla 5 – 1 se va diseñando las formas de onda que se quiere obtener de acuerdo con la aplicación en donde se la quiera utilizar.
- **Ciclos. –** En este apartado se describe en número de ciclos de reloj que la señal permanecerá en cada estado, este número de ciclos son utilizados por el software de simulación para determinar la velocidad con la que se quiere que el circuito trabaje.
- **phase. -** En este apartado se describe la fase de la señal es decir se introduce el desfase que la señal tendrá con respecto a la señal normal, el valor que se puede ingresar es de -1 a 1. Estos valores son el porcentaje de desface de la señal si el valor es positivo la señal se retrasa y si el valor es negativo la señal se adelanta.
- **R/W. –** En este apartado se describe si la función que la señal realiza en su funcionamiento es de lectura o es de escritura, colocando una **R** para lectura y una **W** para escritura.
- **ACK/NACK. –** En este apartado se describe el bit de reconocimiento, **ACK** para indicar que el bit se ha recibido y **NACK** para indicar que el bit no se ha recibido.

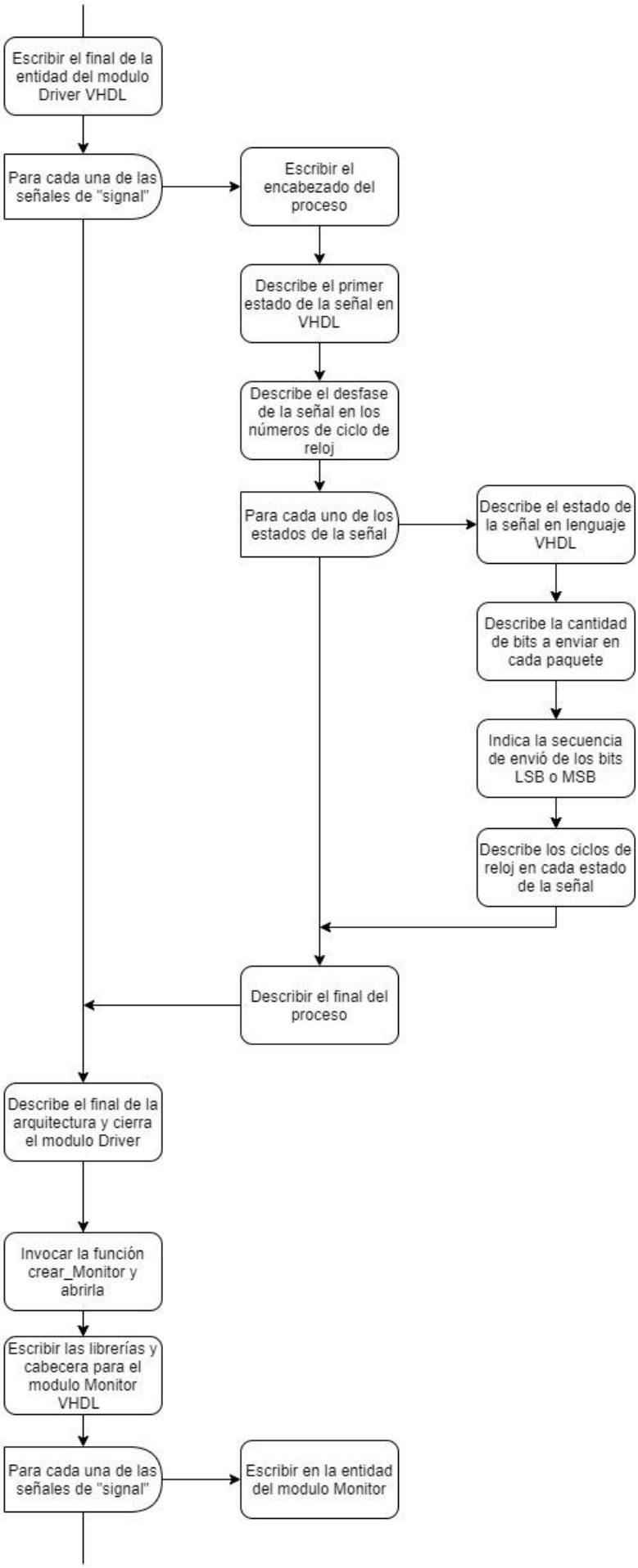
5.1.2 Algoritmo

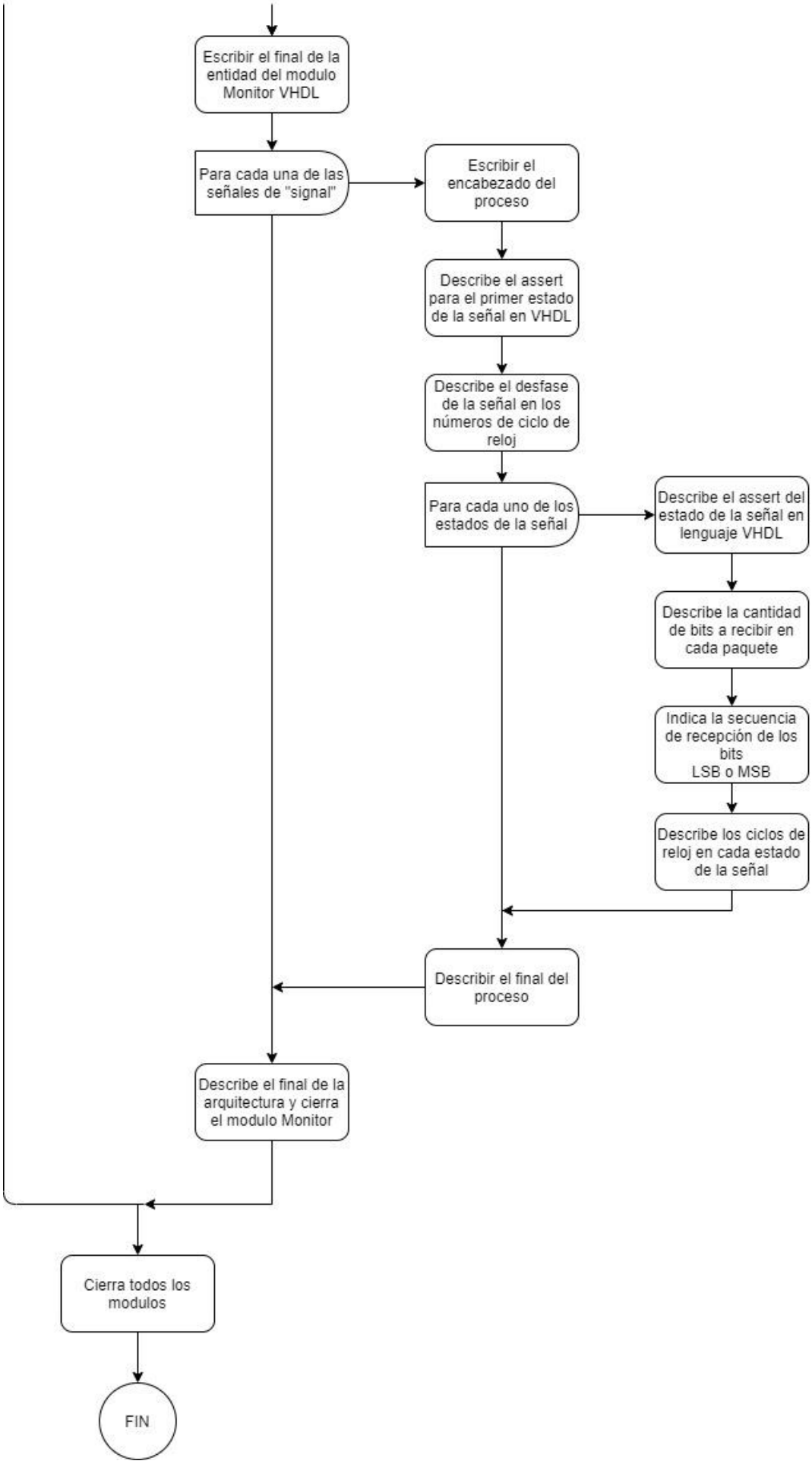
En este apartado se describe la forma en la que el algoritmo funciona y como realiza la generación de los módulos VHDL, a continuación, se describe la forma en la que se generan los módulos y el archivo SVG con el cual se observa las formas de onda que se están diseñando.

Este algoritmo lee las transacciones que anteriormente se describieron en wavedrom, esta aplicación guarda las transacciones en formato JSON y mediante las librerías **wavedrom** que posee Python se renderiza este archivo con el cual ya Python puede trabajar con los datos que contiene la transacción, la librería anteriormente mencionada convierte el archivo JSON en un diccionario de Python con el cual los datos que contiene ya se pueden utilizar para el desarrollo de los módulos VHDL.

A continuación, se presenta el diagrama de bloques del algoritmo que se desarrolló, en donde se describe como está diseñado y sus diferentes segmentos que este tiene para realizar para la generación de los módulos VHDL.


INICIO
Introducir Librerías a utilizar
Definir las funciones para la creación de los módulos VHDL
Leer la transacción, renderizarla y crea el archivo SVG
trans en jsondecoded?
NO
SI
ERROR "ingrese una transacción valida"
Invocar la función crear_package y abrirla
Escribir las librerías y cabecera para el package VHDL
Para cada una de las señales de "trans"
Escribir en el record del modulo package VHDL
Escribir el final del modulo package VHDL
signal en jsondecoded?
NO
SI
ERROR "ingrese una transacción valida"
Invocar la función crear_driver y abrirla
Escribir las librerías y cabecera para el modulo Driver VHDL
Para cada una de las señales de "signal"
Escribir en la entidad del modulo Driver


Escribir el final de la entidad del modulo Driver VHDL
Para cada una de las señales de "signal"
Escribir el encabezado del proceso
Describe el primer estado de la señal en VHDL
Describe el desfase de la señal en los números de ciclo de reloj
Para cada uno de los estados de la señal
Describe el estado de la señal en lenguaje VHDL
Describe la cantidad de bits a enviar en cada paquete
Indica la secuencia de envió de los bits LSB o MSB
Describe los ciclos de reloj en cada estado de la señal
Describir el final del proceso
Describe el final de la arquitectura y cierra el modulo Driver
Invocar la función crear_Monitor y abrirla
Escribir las librerías y cabecera para el modulo Monitor VHDL
Para cada una de las señales de "signal"
Escribir en la entidad del modulo Monitor


Escribir el final de la entidad del modulo Monitor VHDL
Para cada una de las señales de "signal"
Escribir el encabezado del proceso
Describe el assert para el primer estado de la señal en VHDL
Describe el desfase de la señal en los números de ciclo de reloj
Para cada uno de los estados de la señal
Describe el assert del estado de la señal en lenguaje VHDL
Describe la cantidad de bits a recibir en cada paquete
Indica la secuencia de recepción de los bits
LSB o MSB
Describe los ciclos de reloj en cada estado de la señal
Describir el final del proceso
Describe el final de la arquitectura y cierra el modulo Monitor
Cierra todos los modulos
FIN

Lo primero que realiza el algoritmo es generar un archivo SVG en el cual se indica las formas de onda que dicha transacción genera, esto se lo realiza a modo de verificación para observar si la forma de onda generada cumple con las especificaciones que se pide para la realización de los módulos VHDL.

A partir de esto el algoritmo empieza a generar los distintos módulos VHDL que se necesitan generar para el diseño y verificación de los circuitos digitales, para esto de acuerdo al módulo que se desea generar el algoritmo se un archivo VHDL y va introduciendo las diferentes secciones como son inicialmente las librerías y la descripción de la entidad de dicho modulo, a continuación toma los datos del apartado wave de la transición de cada señal y los transforma a un proceso por cada señal de 1 puerto. En estos procesos se describe el comportamiento de las señales para ello se toma los diferentes cambios de estados que tiene las señales y las describe en código VHDL, mediante la utilización de los diversos valores que puede tomar una señal en VHDL los cuales se muestran en la tabla 5 – 2.

Tabla 5-2. Valores de tipo STD_logic VHDL

SIMBOLO	SINIFICADO O EQUIVALENCIA
U	Sin inicializar: usado como valor por defecto
X	Valor desconocido (fuerte)
0	Cero (fuerte): transistor puesto a tierra
1	Uno (fuerte): transistor puesto a alimentación
Z	Alta impedancia
W	Valor desconocido (débil)
L	Cero (débil): resistencias pull - Down
H	Uno (débil): resistencias pull - up
-	Tipo especial útil para síntesis

5.1.2.1 Archivo SVG

Una de las primeras acciones que realiza el algoritmo es generar el archivo SVG de la transacción ingresada para que el usuario pueda comprobar visualmente que la forma de onda de dicha transacción cumple con las especificaciones que se requieren para el diseño y verificación del circuito digital.

Para generar este archivo lo que se hace es cargar la transacción que está en formato JSON en una variable y leemos sus datos, ahora mediante la utilización de la librería JSON des encapsulamos los datos del archivo y lo guardamos en un archivo diccionario de Python, finalmente con la utilización de la librería wavedrom se renderiza los datos de dicho diccionario y se los guarda en un archivo con formato SVG en el cual se podrá visualizar de forma gráfica las formas de onda que la transacción genera. A continuación, se muestra el código utilizado para la generación de este archivo y finalmente en la gráfica 5 – 1 se muestra la forma de onda generada de la transacción ingresada en el apartado 5.1.1.1.

```
f = open("trans.json", "r")
content = f.read()
jsondecoded = json.loads(content)
print(jsondecoded)
a=json.dumps(jsondecoded,indent=4)
print(a)
svg = wavedrom.render(a)
svg.saveas("demo1.svg")
```

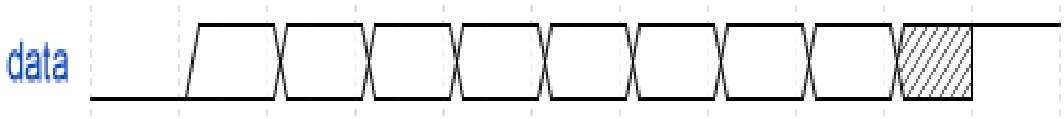


Figura 5-1. Archivo SVG generado de una transacción

5.1.2.2 Package VHDL

En este apartado se trata de definir un tipo de dato compuesto que tenga los datos de la transacción para eso lo primero que realiza el algoritmo es comprobar si la transacción contiene los apartados de Trans y Signal, que son fundamentales para la elaboración de los módulos VHDL, para la generación del package VHDL el algoritmo toma los datos que contiene el apartado trans, entonces:

Lo primero es crear un fichero denominado protocol_common.vhdl en el cual se describe el paquete de datos se quiere enviar. Primeramente, se ingresa las librerías necesarias para que este paquete funciona, se utiliza la librería IEEE.STD_LOGIC_1164.all la cual describe el estándar VHDL. A partir de esto se describe el paquete en sí, en donde se define el tipo de paquete y los datos que este contendrá, aquí el algoritmo coloca todas las señales que están descritas en el apartado trans y además introduce una señal denominada Valid, la cual determinan el inicio del funcionamiento del módulo Driver. A continuación, se indica el módulo package VHDL que se genera con la transacción mostrada en el apartado 5.1.1.1.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package protocol_common is

	type protocol_type is
		record
			data : STD_LOGIC_VECTOR (7 downto 0);
			valid : STD_LOGIC;
		end record;

end protocol_common;

package body protocol_common is

end protocol_common;
```

5.1.2.3 Driver VHDL

Para la generación del módulo Driver el algoritmo toma los datos que están encapsulados en el apartado Signal de la transacción ingresada. Entonces se genera un archivo denominado Driver.vhdl En el cual se describe el módulo. Lo primero es ingresar las librerías necesarias para el funcionamiento y generación de las señales que contiene cada módulo, se utiliza la principal librería de VHDL que es la IEEE.STD_LOGIC_1164.all la cual describe el estándar VHDL además se utiliza work.protocol_common.ALL la cual permite integrar el módulo package VHDL al módulo Driver.

Posteriormente se describe la entidad del módulo o componente utilizando todas las señales que tiene el apartado Signal de la transacción y finalmente se describe los cambios de estado que tienen cada señal, para esto por cada señal se crea un proceso y dentro de este se describe los cambios de estado que tienen dichas señales. Se utiliza los códigos descritos en el apartado wave y el algoritmo va mirando los cambios de estado que se muestran en la tabla 5 – 1 y los convierte a cambios de estado en lenguaje VHDL los cuales se muestran en la tabla 5 – 2 , continuación se muestra las librerías y la entidad del módulo Driver que se generó al introducir la transacción mostrada en el apartado 5.1.1.1.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use work.protocol_common.ALL;

entity Driver is
        Port (
                input_tran : in protocol_type;
                clk : in  STD_LOGIC;
                data : out STD_LOGIC);
end Driver;
```

5.1.2.4 Monitor VHDL

En la generación del módulo Monitor en VHDL el algoritmo utiliza los datos del apartado Signal de la transacción ingresada, entonces se crea un archivo al que se denomina Monitor.vhdl En el cual se describe el módulo. Lo primero es ingresar las librerías necesarias para el funcionamiento y generación de las señales que contiene cada módulo, se utiliza la principal librería de VHDL que es la IEEE.STD_LOGIC_1164.all la cual describe el estándar VHDL además se utiliza work.protocol_common.ALL la cual permite integrar el módulo package VHDL al módulo Monitor.

Posteriormente se describe la entidad del módulo o componente utilizando todas las señales que tiene el apartado Signal de la transacción que en este caso serán tomadas como señales de entrada al módulo entonces el algoritmo va tomando todos los cambios de estado que generan las señales y las va convirtiendo a funciones de verificación como son los Asserts y si se trata del envió de datos crea funciones que guardaran estos datos que se envían para luego verificar si los datos obtenidos concuerdan con los datos enviados inicialmente. A continuación, se observa las bibliotecas y la descripción de la entidad del módulo Monitor .

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use work.protocol_common.ALL;

entity Monitor is
        Port (
                output_tran : inout protocol_type;
                clk : in  STD_LOGIC;
                data : in STD_LOGIC);
end Monitor;
```

5.2 Uso del algoritmo

1. Lo primero es obtener las formas de onda del circuito que se desea diseñar y verificar su funcionamiento en VHDL
2. Lo segundo es utilizar la aplicación wavedrom para dibujar estas gráficas y añadir los datos necesarios para la creación de los módulos VHDL, para el correcto desarrollo de la transacción se utiliza las funciones y características que se explicaron en este capítulo.
3. Ahora estos datos se los carga en el archivo trans.json que se encuentra en la misma carpeta del algoritmo de generación.
4. Con la ayuda del IDE de Python se pone en funcionamiento el algoritmo el cual de manera automática genera el archivo SVG con el cual se puede observar de manera gráfica las formas de onda del circuito diseñado en VHDL.
5. Además, se crean los archivos protocol_common, Driver y Monitor a los cuales ya se les puede realizar sus respectivos testbench e ingresar los estímulos que se desean aplicar.
6. Como resultado se puede observar las formas de onda que estos circuitos VHDL generan y la verificación del correcto funcionamiento de los módulos generados.

En la Tabla 5.3 se indica un resumen del proceso que se debe seguir para la elaboración de un protocolo de comunicación describiendo su diseño en VHDL, mediante la utilización del algoritmo elaborado en este trabajo.

Tabla 5-3. Tabla Resumen del funcionamiento

ENTRADA	PROCESO	SALIDA
Búsqueda de las formas de onda	Buscar las formas de onda del protocolo que se desea realizar los módulos en VHDL	Imágenes de las formas de onda
descripción JSON	Describir la forma de onda en la aplicación Wavedrom y generar la imagen vectorial de la forma de onda	Imagen SVG de la forma de onda y transacción JSON
transacción JSON	La transacción diseñada en Wavedrom se la carga en el archivo trans.json y se ejecuta el algoritmo el cual genera de forma automática los módulos Driver, Monitor, Package VHDL y un archivo SVG con la forma de onda generada.	Módulos VHDL
Módulos VHDL	Con la ayuda del software Xilinx ISE se realiza los módulos Testbench para cada uno de los módulos VHDL generados anteriormente.	Testbench de los módulos VHDL
Testbench de los módulos VHDL	Simular los Testbench de los módulos VHDL generados, para esto se utiliza el software Xilinx ISIM	Graficas de las formas de onda de los protocolos de comunicación.
Imágenes iniciales de las formas de onda.	Verificar el correcto funcionamiento de las formas de onda generadas de los protocolos de comunicación comparándolas con las imágenes iniciales de las formas de onda las cuales se buscaron en la red.	Graficas de las formas de onda generadas

6 PRUEBAS Y RESULTADOS

Una máquina puede hacer el trabajo de 50 hombres ordinarios, pero ninguna máquina puede hacer el trabajo de un hombre extraordinario

- Elbert Hubbard-

EN este capítulo se realizarán diferentes pruebas de funcionamiento para determinar el nivel de generación de los módulos VHDL que tiene el algoritmo diseñado, para esto se implementaran las transacciones para algunos protocolos de comunicación. Luego de esto se tomarán estos módulos generados y se crearán sus respectivos módulos de simulación testbench y así verificar el correcto funcionamiento tanto de los Drivers como de los Monitores VHDL creados. Con esto se comprueba que el algoritmo diseñado es capaz de crear un completo sistema Driver – Monitor para el diseño y verificación de circuitos digitales descritos en VHDL

6.1 Protocolos Usados

Lo primero es determinar las formas de onda de salida que tiene los protocolos o circuitos digitales que se desea implementar. Para ello de acuerdo a la aplicación en la que se desea utilizar el circuito a diseñar se va elaborando las formas de onda con sus respectivos niveles, para esto se utiliza una herramienta denominada wavedrom la cual es una aplicación en la que se permite dibujar las formas de onda de cualquier diseño, esta aplicación es muy utilizada en la elaboración de los planes de prueba en donde se describe lo que se va a simular y se observa las formas de onda que produce tal diseño.

Las diferentes pruebas que se van a realizar se las van a hacer en concreto con algunos protocolos de comunicación conocidos los cuales se los describa a continuación en este capítulo. Las pruebas se realizan con protocolos de comunicación ya que en estos circuitos se integran diferentes valores o estados que una forma de onda puede tener, como son valores lógicos altos y bajos, transmisión de datos, detección de bit de paridad, direccionamiento tanto de dispositivos y registros, selección de modo lectura o escritura, bit de reconocimiento , etc. A continuación, se presenta las transacciones que se implementan para la generación de los módulos VHDL de cada protocolo y además se observa las gráficas SVG de las formas de onda que producen estas transacciones. Esto se consigue gracias a la aplicación wavedrom y también el algoritmo es capaz de generar estas graficas SVG esto se lo realiza como forma de comprobación y asegurarnos de las formas de onda que la transacción genera son iguales tanto de la aplicación wavedrom como del algoritmo de generación que se diseñó para este trabajo.

Cabe mencionar que este algoritmo diseñado para la generación de los módulos VHDL no es específicamente para protocolos de comunicación, se puede diseñar y verificar cualquier circuito digital que cuente con los cambios de estado con los que cuenta el algoritmo y los cuales se describió en capítulos anteriores, sin embargo, los protocolos elegidos cuentan con diferentes cambios de estado por lo que son ideales para verificar el funcionamiento del algoritmo del trabajo.

6.1.1 UART

En el capítulo 3 ya se describió el funcionamiento de este protocolo junto con la forma de onda y los distintos cambios de estados que este protocolo genera al ponerse en funcionamiento. Por lo tanto, ahora mediante la ayuda de la aplicación wavedrom, de su formato de desarrollo y de las diferentes herramientas que posee, se procede a desarrollar la transacción para este protocolo. Esta transacción se la desarrolla en formato JSON y se la divide en 2 partes las cuales se las describe a continuación.

Trans. –

En este apartado se describe las características de la transacción de este protocolo, las cuales servirán para la generación del Package VHDL. Se describen datos como el nombre del puerto del package que en este caso es **data**, también se describe el tipo de dato que en este protocolo es **STD_logic_vector** con una longitud de **8** bits, además se indica también el orden con el que se transmiten los datos que en este caso **LSB** lo que significa que la transmisión de datos iniciara por el bit menos significativo del mensaje.

Signal. –

En esta parte se describe el movimiento de los pines o cambio de estados de cada uno de los puertos que tiene el circuito digital a diseñar. Para el protocolo UART solo se requiere de un PIN al que de acuerdo con esta transacción se lo denomina **data**, una de las funciones que tiene wavedrom es que mediante la utilización de **wave** se puede describir el comportamiento que tendrá la señal para lo cual se utiliza los comandos descritos en el capítulo 4. Este módulo comienza con un valor lógico en bajo representado por la letra **l**, para luego empezar con la transmisión de datos la cual se hace enviando 8 paquetes de datos y se lo representa con él =, después de esto se determina el bit de paridad de los datos enviados para esto se utiliza la letra **x** y finalmente se pone la línea a un valor lógico alto para eso se utiliza la letra **h**.

También se indica otros elementos como es el tipo de dato con el que trabajara la señal que en este caso al ser **logic** nos indica que es un dato **STD_logic**, cuya longitud será de **1** bit. Otro elemento que se indica en esta transacción es **ciclos** el cual hace referencia a la velocidad con la que trabajara este protocolo, en este caso se indica que por cada estado de la señal al momento de la simulación pasaran 10 ciclos de reloj del simulador. A continuación, se muestra la transacción que es utilizada para la elaboración de los módulos del protocolo UART.

```
{
 "trans":
       [
         {"name": "data", "type":"vector","len": 8, "start": "LSB"}
       ],

 "signal":
       [
         {"name": "data", "wave": "l========xh", "type":"logic","len": 1,
         "ciclos": 10}
       ]
}
```

La transacción mostrada anteriormente se la puede cargar en la aplicación wavedrom la cual genera una forma de onda como la que aparece en la figura 6 – 1. Además, dicha transacción al cargarla en el algoritmo de generación también genera un archivo SVG en el cual se puede observar la forma de onda que produce tal transacción, esto se realizó gracias a las librerías con las que cuenta Python y además se lo hizo como forma de verificación visual de las formas de onda que se están generando.

Figura 6-1. Grafica SVG del protocolo UART

6.1.2 I2C

En el capítulo 3 ya se describió el funcionamiento de este protocolo junto con la forma de onda y los distintos cambios de estados que este protocolo genera al ponerse en funcionamiento. Por lo tanto, ahora mediante la ayuda de la aplicación wavedrom, de su formato de desarrollo y de las diferentes herramientas que posee, se procede a desarrollar la transacción para este protocolo. Esta transacción se la desarrolla en formato JSON y se la divide en 2 partes las cuales se las describe a continuación.

Trans. –

En este apartado se describe las características de la transacción de este protocolo, las cuales servirán para la generación del Package VHDL. En el caso de este protocolo se requiere de 3 señales en la transacción las cuales se describen a continuación:

- La primera tiene por nombre **Slave_address** y es de tipo **STD_logic_vector** con una longitud de **7** bits y empieza a transmitir los bits desde el más significativo **MSB.**
- La segunda tiene por nombre **Reg_address** y es de tipo **STD_logic_vector** con una longitud de **8** bits y empieza a transmitir los bits desde el más significativo **MSB.**
- La tercera tiene por nombre **data** y es de tipo **STD_logic_vector** con una longitud de **8** bits y empieza a transmitir los bits desde el más significativo **MSB.**

Signal. –

En esta parte se describe el movimiento de los pines o cambio de estados de cada uno de los puertos que tiene el circuito digital a diseñar. Las señales descritas en este apartado son las que se utilizan para la elaboración tanto del módulo Driver como del Monitor, Para el protocolo I2C se requiere de 2 señales de salida las cuales se les explica a continuación:

- La primera señal de salida tiene por nombre **scl**, y su forma de onda esta descrita en **wave** en la cual inicia con 2 estados con valor lógico alto representado por la letra **h**, después de esto tiene 9 estados representados con la letra **n** la cual hace referencia a los ciclos de reloj que empiezan con valor lógico bajo y luego cambian a valor lógico alto, luego tiene 2 estados lógicos bajos **l**, 9 ciclos de reloj **n**, 2 valores lógicos bajos **l,** 9 ciclos de reloj **n** y un valor lógico alto **h**. Es de tipo **STD_logic** con longitud de 1 bit, duración de 10 ciclos de reloj por cada estado y una fase de 0.2 esto se lo realiza para retrasar un 20% la señal y así sincronizarla con la forma de funcionamiento de este protocolo.
- La segunda señal de salida tiene por nombre **sda**, y su forma de onda esta descrita en **wave** en la cual se utiliza las funciones de wavedrom que se detallaron en el capítulo 5 para que cambie de valor sus estados. Es de tipo **STD_logic** con longitud de 1 bit, duración de 10 ciclos de reloj por cada estado y cuenta con un apartado **R/W** en el cual se indica si la señal es de lectura o escritura y también se puede indicar el reconocimiento en **ACK/NACK.**

```
{
 "trans":
  [
    {"name": "slave_address", "type":"vector", "len": 7, "start": "MSB"},
    {"name": "reg_address", "type":"vector", "len": 8, "start": "MSB"},
    {"name": "data", "type":"vector", "len": 8, "start": "MSB"}
  ],
 "signal":
  [
      {"name": "scl", "wave": "hhnnnnnnnnnllnnnnnnnnnllnnnnnnnnnnh",
      "type":"logic","len": 1, "ciclos": 10, "phase": 0.2},

      {"name": "sda", "wave": "hl=======34ll========4ll========4lh",
      "type":"logic","len": 1, "ciclos": 10, "R/W": "W","ACK/NACK":
      ["ACK","ACK","ACK"]}
  ]
}
```

La transacción mostrada anteriormente se la puede cargar en la aplicación wavedrom la cual genera una forma de onda como la que aparece en la figura 6 – 2. Además, dicha transacción al cargarla en el algoritmo de generación también genera un archivo SVG en el cual se puede observar la forma de onda que produce tal transacción, esto se realizó gracias a las librerías con las que cuenta Python y además se lo hizo como forma de verificación visual de las formas de onda que se están generando.

En la figura 6 – 2 se puede apreciar de forma gráfica la transacción diseñada, se puede apreciar como el cambio de estados solo se realiza cuando la señal **scl** está a nivel bajo con excepción de los estados de inicio y fin en el cual cambian cuando la señal **scl** está a nivel alto, se observa cómo van cambiando los estados y enviando los datos por la señal.

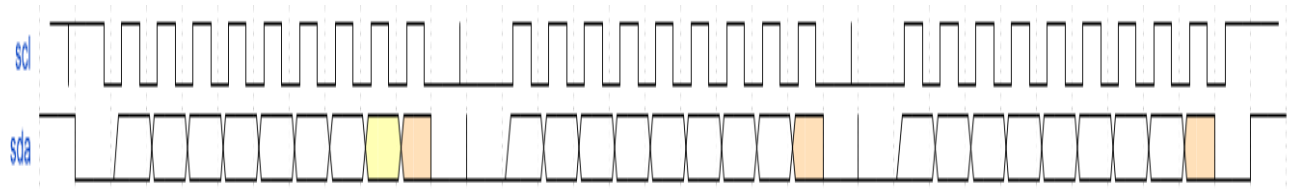


Figura 6-2. Grafica SVG del protocolo I2C

6.1.3 SPI

En el capítulo 3 ya se describió el funcionamiento de este protocolo junto con la forma de onda y los distintos cambios de estados que este protocolo genera al ponerse en funcionamiento. Por lo tanto, ahora mediante la ayuda de la aplicación wavedrom, de su formato de desarrollo y de las diferentes herramientas que posee, se procede a desarrollar la transacción para este protocolo. Esta transacción se la desarrolla en formato JSON y se la divide en 2 partes las cuales se las describe a continuación.

Trans. –

En este apartado se describe las características de la transacción de este protocolo, las cuales servirán para la generación del Package VHDL. En el caso de este protocolo se requiere de 1 señal en la transacción la cual se describe a continuación:

- Esta señal tiene por nombre **data** y es de tipo **STD_logic_vector** con una longitud de **16** bits y empieza a transmitir los bits desde el más significativo **MSB.**

Signal. –

En esta parte se describe el movimiento de los pines o cambio de estados de cada uno de los puertos que tiene el circuito digital a diseñar. Las señales descritas en este apartado son las que se utilizan para la elaboración tanto del módulo Driver como del Monitor, Para el protocolo SPI se requiere de 2 señales de salida las cuales se les explica a continuación:

- La primera es la señal de reloj con la cual se sincroniza este protocolo con los demás dispositivos y tiene por nombre **sclk**, en su apartado **wave** inicia con un estado en nivel lógico alto **h**, para luego iniciar con los ciclos de reloj **n**, es de tipo **STD_logic**, con una longitud de 1 bit y una duración de 10 ciclos de reloj por cada estado.
- La segunda es la señal encargada de transmitir los datos desde el dispositivo maestro al esclavo y tiene por nombre **mosi**, en su apartado **wave** inicia con un estado en nivel lógico alto **h**, para luego iniciar con la transmisión de datos utilizando el = aquí se envían 16 datos y finalmente la señal se coloca a un estado en nivel lógico alto **h**, es de tipo **STD_logic**, con una longitud de 1 bit y una duración de 10 ciclos de reloj por cada estado.
- La tercera es la señal que determina que el dispositivo está enviando los datos y tiene por nombre **ss**, en su apartado **wave** inicia con un estado en nivel lógico alto **h**, para luego colocarse a nivel lógico bajo **l** durante todo el tiempo que dure la transmisión de datos y finalmente la señal se coloca a un estado en nivel lógico alto **h**, es de tipo **STD_logic**, con una longitud de 1 bit y una duración de 10 ciclos de reloj por cada estado.

```
{
 "trans":
  [
    {"name": "data", "type":"vector", "len": 16, "start": "MSB"}
  ],

 "signal":
  [
      {"name": "sclk", "wave": "hnnnnnnnnnnnnnnnn", "type":"logic","len": 1,
      "ciclos": 10},

      {"name": "mosi", "wave": "h================h", "type":"logic","len": 1,
      "ciclos": 10},

      {"name": "ss", "wave": "hllllllllllllllllh", "type":"logic","len": 1,
      "ciclos": 10}
  ]
}
```

La transacción mostrada anteriormente se la puede cargar en la aplicación wavedrom la cual genera una forma de onda como la que aparece en la figura 6 – 3. Además, dicha transacción al cargarla en el algoritmo de generación también genera un archivo SVG en el cual se puede observar la forma de onda que produce tal transacción, esto se realizó gracias a las librerías con las que cuenta Python y además se lo hizo como forma de verificación visual de las formas de onda que se están generando.

En la figura 6 – 3 se puede apreciar de forma gráfica la transacción diseñada, se observa como la señal **sclk** genera la señal de reloj con la cual el dispositivo se sincronizan con otros módulos, la señal **mosi** es la señal que envía los datos y la señal **ss**, es la señal que determina que el dispositivo está enviando los datos hacia otro dispositivo.

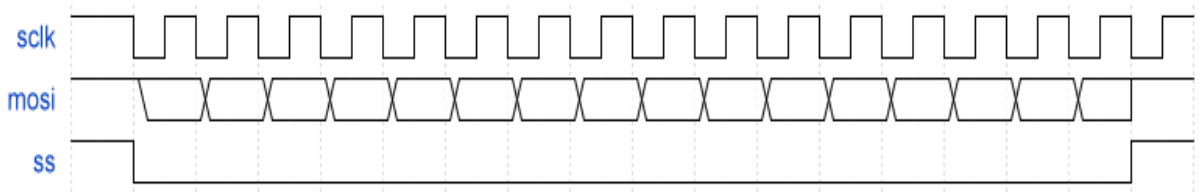


Figura 6-3. Grafica SVG del protocolo SPI

6.1.4 Protocolo ficticio

En el capítulo 3 ya se describió el funcionamiento de este protocolo junto con la forma de onda y los distintos cambios de estados que este protocolo genera al ponerse en funcionamiento. Por lo tanto, ahora mediante la ayuda de la aplicación wavedrom, de su formato de desarrollo y de las diferentes herramientas que posee, se procede a desarrollar la transacción para este protocolo. Esta transacción se la desarrolla en formato JSON y se la divide en 2 partes las cuales se las describe a continuación.

Trans. –

En este apartado se describe las características de la transacción de este protocolo, las cuales servirán para la generación del Package VHDL. En el caso de este protocolo se requiere de 1 señal en la transacción la cual se describe a continuación:

- Esta señal tiene por nombre **data** y es de tipo **STD_logic_vector** con una longitud de **16** bits y empieza a transmitir los bits desde el más significativo **MSB.** Este protocolo al ser ficticio solo se le da una señal ya partir de esta transacción generara las demás señales de salida que son necesarias para el correcto funcionamiento de este protocolo.

Signal. –

En esta parte se describe el movimiento de los pines o cambio de estados de cada uno de los puertos que tiene el circuito digital a diseñar. Las señales descritas en este apartado son las que se utilizan para la elaboración tanto del módulo Driver como del Monitor, Para el este protocolo ficticio se requiere de 4 señales de salida las cuales se les explica a continuación:

- La primera es la señal encargada de transmitir los datos desde el dispositivo hacia otro modulo y tiene por nombre **data**, en su apartado **wave** inicia con dos estados en alta impedancia **z**, para luego empezar la transmisión de datos con el símbolo =, esta transmisión se la alarga utilizando el símbolo **punto** con lo cual él envió de cada paquete de datos tarda 4 espacio de estados y finalmente al terminar de enviar los datos la señal vuelve a colocarse en alta impedancia utilizando la letra **z**. es de tipo **STD_logic_vector**, con una longitud de 4 bits lo que significa que se transmitirán paquetes de 4 bits y una duración de 10 ciclos de reloj por cada estado.
- La segunda es la señal encargada de definir el inicio de la transmisión de datos y tiene por nombre **starp**, en su apartado **wave** inicia con un estado en nivel lógico bajo **l**, para luego colocarse en nivel lógico alto **h** durante un espacio de estado y finalmente se coloca en nivel lógico bajo **l** durante el resto de la transmisión. es de tipo **STD_logic**, con una longitud de 1 bit y una duración de 10 ciclos de reloj por cada estado.
- La tercera es la señal encargada de definir el final de la transmisión de datos y tiene por nombre **endp**, en su apartado **wave** inicia con un estado en nivel lógico bajo **l** y este continua así hasta que se terminan de enviar todos los datos, entonces en ese momento esta señal se coloca en un nivel lógico alto **h** durante un espacio de estado y finalmente se coloca en nivel lógico bajo **l,** es de tipo **STD_logic**, con una longitud de 1 bit y una duración de 10 ciclos de reloj por cada estado.
- La cuarta es la señal que determina que el dispositivo está enviando los datos o que está en funcionamiento y tiene por nombre **ena**, en su apartado **wave** inicia con un estado en nivel lógico alto **h**, para luego colocarse a nivel lógico bajo **l** durante todo el tiempo que dure la transmisión de datos y finalmente la señal se coloca a un estado en nivel lógico alto **h**, es de tipo **STD_logic**, con una longitud de 1 bit y una duración de 10 ciclos de reloj por cada estado.

```
{
 "trans":
 [
   {"name": "data", "type":"vector","len": 16, "start": "MSB"}
 ],

 "signal":
 [
      {"name":"data","wave":"zz=...=...=...=...zz", "type":"vector","len": 4,
      "ciclos": 10},

      {"name":"startp","wave":"lhlllllllllllllllll","type":"logic","len": 1,
      "ciclos": 10},

      {"name": "endp","wave":"lllllllllllllllllhl", "type":"logic","len": 1,
      "ciclos": 10},

      {"name": "ena", "wave":"hlllllllllllllllllh", "type":"logic","len": 1,
      "ciclos": 10}
 ]
}
```

La transacción mostrada anteriormente se la puede cargar en la aplicación wavedrom la cual genera una forma de onda como la que aparece en la figura 6 – 4. Además, dicha transacción al cargarla en el algoritmo de generación también genera un archivo SVG en el cual se puede observar la forma de onda que produce tal transacción, esto se realizó gracias a las librerías con las que cuenta Python y además se lo hizo como forma de verificación visual de las formas de onda que se están generando.

En la figura 6 – 4 se puede apreciar de forma gráfica la transacción diseñada, se observa como la señal **startp** y la señal **endp** están la mayoría del tiempo a nivel lógico bajo y solo se ponen a nivel lógico alto en el caso de la señal **startp** cuando inicia la transmisión de datos y la señal **endp** cuando finaliza la transmisión de datos, la señal **data** se observa que normalmente pasa en estado de alta impedancia y solo cambia para enviar los datos y la señal **ena**, es la señal que determina que el dispositivo está enviando los datos hacia otro dispositivo, se observa que esta inicialmente a nivel lógico alto y cambia a nivel lógico bajo durante el tiempo que tarda la transmisión de datos.

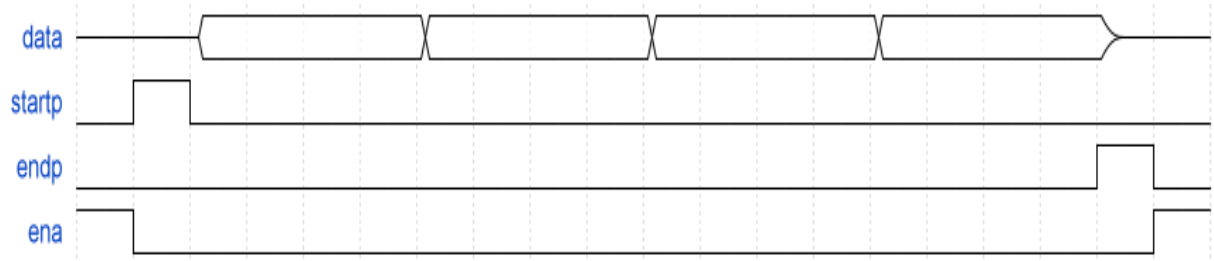


Figura 6-4. Grafica SVG del protocolo ficticio

6.2 Resultados Obtenidos

Una vez que se han descrito los diferentes pruebas que se van a realizar y se diseñan sus respectivas transacciones, se procede a cargarlas una a una en el algoritmo de generación de módulos VHDL, el cual genera los diferentes módulos para cada protocolo, ahora es momento de diseñar los testbench para cada módulo y simularlos para verificar su correcto funcionamiento.

Se realiza los testbench tanto para el Driver como para el Monitor y se los integra en un solo fichero y así obtener la arquitectura de verificación de los TLM que se muestra en la figura 6 – 5. Con este modelo se ingresa una transacción al Driver y al final de toda esta metodología de verificación el Monitor nos entregara como resultado una transacción, la cual comprobaremos si es la misma que la ingresada al Driver.

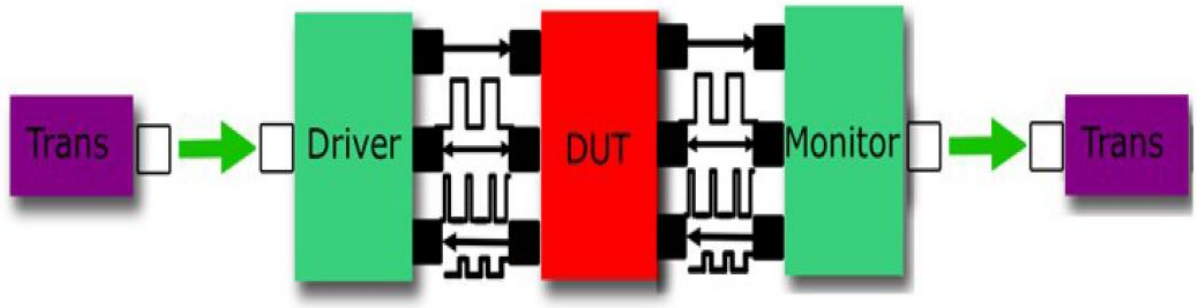


Figura 6-5. Testbench modelado a nivel de transacciones [15]

6.2.1 UART

En esta simulación se verifica el correcto funcionamiento del protocolo UART, para lo cual se crean los respectivos módulos testbench para el Driver como para el Monitor. En estos módulos se ingresan los estímulos o datos que se quieren enviar, para lo cual se ingrese el código que se muestra a continuación en este apartado, el que se consta los datos que se desean transmitir.

```
stim_proc: process
   begin
      -- hold reset state for 100 ns.
               input_tran.valid <='0';
               wait for 100 ns;
               input_tran.data <= "00110011";
               input_tran.valid <='1';
               wait for clk_period;
               input_tran.valid <='0';
               wait for clk_period*10;
               wait;
   end process;
```

Estos estímulos se los ingresa a manera de proceso, en donde a la señal **input_tran.valid** se le asigna un valor lógico bajo '0', ahora se espera 100 nano segundos y se procede a cambiar el estado de esta señal **input_tran.valid** a un valor lógico alto '1', en esta parte también se le asigna el valor de los datos a enviar a la señal **input_tran.data** que en este caso se enviara los datos **"00110011"**, ahora se espera un ciclo de reloj que es equivalente a 10 nano segundos y se procede a cambiar el estado de esta señal **input_tran.valid** a un valor lógico bajo '0'. Se espera 10 ciclos de reloj y el proceso de estimulación se termina.

6.2.1.1 Package VHDL

Este es uno de los módulos que se generan y es en el que se describe la transacción que se desea enviar, este varía de acuerdo con la cantidad de datos que se desea enviar y a otras funcionalidades como son: si la transacción necesita identificar una dirección de dispositivo o una dirección de registro. Las cuales se encapsularán en un paquete VHDL como el que se muestra a continuación.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package protocol_common is

        type protocol_type is
                record
                        data : STD_LOGIC_VECTOR (7 downto 0);
                        valid : STD_LOGIC;
                end record;

end protocol_common;

package body protocol_common is

end protocol_common;
```

En este paquete se describe primeramente las librerías necesarias para la elaboración de este, en este caso solo se utiliza la librería principal la cual es la **IEEE** y hace referencia al estándar **IEEE.STD_LOGIC_1164**. A continuación de esto se empieza a describir el paquete al cual se lo denomina con el nombre de **protocol_common** y dentro de este se declara un tipo de protocolo el cual se denominará **protocol_type** el que contendrá un récord en el que se describen todas las señales que este paquete tendrá. En este caso el paquete contendrá 2 señales las cuales se des describe continuación:

- **Data.** – esta señal es un vector cuya longitud es de 8 bits y es en donde el estímulo asignara los datos que se desean enviar.
- **Valid.** - esta señal es de tipo lógico por lo tanto solo será de 1 bit de longitud y es utilizada por el estímulo del testbench para iniciar con la transición de datos.

6.2.1.2 Driver

Este módulo es el encargado en convertir la transacción enviada en movimiento de pines y así generar las formas de onda para la transmisión del paquete de datos. En este módulo se describe al dispositivo físico y los puertos que este tendrá, los cuales pueden ser tanto pines de entrada como de salida, posteriormente se describe el movimiento que estos pines tendrán con el propósito de realizar la función de la transacción y enviar el paquete de datos. A continuación, se describe una parte del código del Driver generado en concreto se describe la entidad o caja negra.

Lo primero que se describe En este módulo son las librerías utilizadas, se utiliza la librería principal la cual es la **IEEE** y hace referencia al estándar **IEEE.STD_LOGIC_1164** en esta parte también se debe indicar que se utilizara un paquete de datos al cual se lo denomina como **work.protocol_common**, ahora se procede a describir la entidad de este módulo en donde se indica todos los puertos que este tendrá ya sean de entrada o de salida, también pueden ser bidireccionales, a continuación se describe los puertos necesarios para el funcionamiento de este protocolo.

- **Input_tran. –** Este es un puerto de entrada de tipo **protocol_type**, lo que quiere decir que por este puerto ingresaran o estarán los datos y las señales descritos en el paquete de datos anteriormente mencionado.
- **Clk. –** Esta es una señal de entrada de tipo lógico, es una señal de reloj utilizada por el software para el funcionamiento de los módulos que se desarrollan.
- **Data. –** Esta es una señal de salida de tipo lógico, Esta es la única señal de salida ya que este protocolo solo necesita de una línea para transmitir los datos.

Después de definir la entidad se procede a diseñar la arquitectura de este módulo, en la cual se describe el movimiento de pines y cambios de señal que cada una de las líneas tiene que realizar a efecto de cumplir con la función de la transacción y enviar el paquete de datos deseado.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use work.protocol_common.ALL;

entity Driver is
        Port (
                input_tran : in protocol_type;
                clk : in  STD_LOGIC;
                data : out STD_LOGIC);
end Driver;
```

Ahora se procede a relizar la simulacion del modulo Driver y se ontiene como resultado las formas de onda mostradas en la figura 6 – 6. A continuacion se procede a describir cada una de estas señales y los datos que estan llevan.

Input_tran. – cómo se indicó esta señal contiene los datos del paquete VHDL. Por lo tanto, tiene 2 líneas una denominada input_tran.data la que contiene los datos a transmitir que en este caso son **00110011** y la línea input_tran.valid la que indica el momento en el cual se debe iniciar con la transmisión de datos.

Clk. – Es una señal de reloj interna que utiliza el software para sincronizar las líneas y determinar el tiempo de transmisión de cada estado lógico.

Data. – Es la única señal de salida, debido a que este protocolo envía sus datos por una sola línea a continuación se describe las diferentes etapas de esta línea.

- **Start. –** es el bit de inicio de este protocolo, el cual tiene que estar en valor lógico bajo y se comprueba que efectivamente lo está.
- **Datos. –** En esta etapa se envía los datos empezando por el menos significativo quede debe ser **11001100**, efectivamente se envía de esta manera.
- **BP. –** Aquí este módulo calcula el bit de paridad de los datos enviados, se utiliza el bit de paridad Par en donde la cantidad de estados en alto debe ser par y como aquí se envían 4 datos en alto el BP debe ser 0 el cual efectivamente lo es.
- **Stop. –** es el bit de finalización de este protocolo, el cual tiene que estar en valor lógico alto y se comprueba que efectivamente lo está.

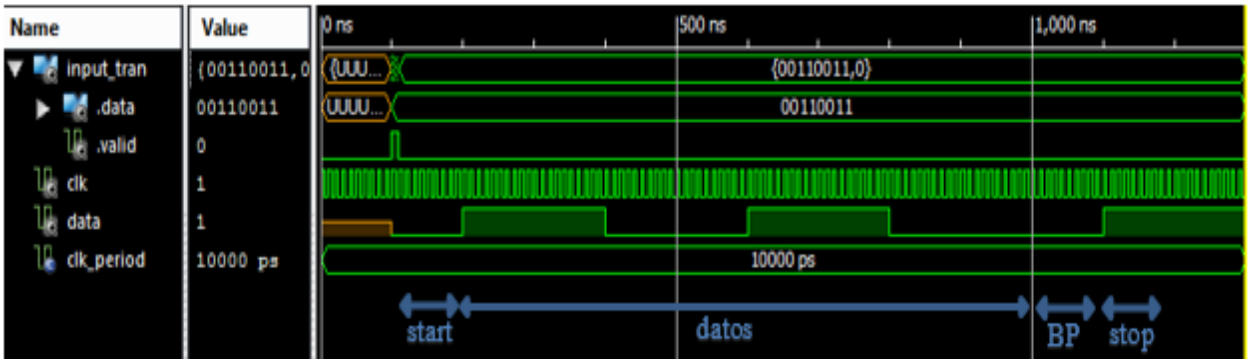


Figura 6-6. Simulación Driver del protocolo UART

6.2.1.3 Monitor

Este módulo es el encargado en convertir el movimiento de pines a transacciones y así poder verificar la correcta transmisión del paquete de datos. En este módulo se describe al dispositivo físico y los puertos que este tendrá, los cuales pueden ser tanto pines de entrada como de salida o pines bidireccionales, posteriormente se describe la forma en la que este módulo va tomando los cambios de estado de cada una de las líneas y va obtenido los datos enviados en la transacción. A continuación, se describe una parte del código del Monitor generado en concreto se describe la entidad o caja negra.

Lo primero que se describe En este módulo son las librerías utilizadas, se utiliza la librería principal la cual es la **IEEE** y hace referencia al estándar **IEEE.STD_LOGIC_1164** en esta parte también se debe indicar que se utilizara un paquete de datos al cual se lo denomina como **work.protocol_common**, ahora se procede a describir la entidad de este módulo en donde se indica todos los puertos que este tendrá ya sean de entrada o de salida, también pueden ser bidireccionales, a continuación se describe los puertos necesarios para el funcionamiento de este protocolo.

- **Output_tran. –** Este es un puerto bidireccional, pero se lo utiliza como puerto de salida que es de tipo **protocol_type**, lo que quiere decir que en este puerto se guardaran los datos obtenidos del movimiento de pines de las señales
- **Clk. –** Esta es una señal de entrada de tipo lógico, es una señal interna de reloj utilizada por el software para el funcionamiento de los módulos que se desarrollan.
- **Data. –** Esta es una señal de entrada de tipo lógico ya que este protocolo envía sus datos por una sola línea esta será la única señal analizada para verificar el correcto funcionamiento de este protocolo.

Después de definir la entidad se procede a diseñar la arquitectura de este módulo, en la cual se describe la forma en la que este módulo va tomando los cambios de estado de cada una de las líneas y va obtenido los datos enviados en la transacción y así poder verificar el correcto funcionamiento del protocolo de comunicación analizado.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use work.protocol_common.ALL;

entity Monitor is
        Port (
                output_tran : inout protocol_type;
                clk : in  STD_LOGIC;
                data : in STD_LOGIC);
end Monitor;
```

Ahora se procede a relizar la simulacion del modulo Monitor y se ontiene como resultado las formas de onda mostradas en la figura 6 – 7. A continuacion se procede a describir cada una de estas señales y los datos que estan llevan.

Clk. – Es una señal de reloj interna que utiliza el software para sincronizar las líneas y determinar el tiempo de transmisión de cada estado lógico.

Data. – Es la única señal de salida, debido a que este protocolo envía sus datos por una sola línea a continuación se describe las diferentes etapas de esta línea.

- **Start. –** es el bit de inicio de este protocolo, el cual tiene que estar en valor lógico bajo y se comprueba que efectivamente lo está.
- **Datos. –** En esta etapa se envía los datos empezando por el menos significativo quede debe ser **11001100**, efectivamente se envía de esta manera.
- **BP.** – Aquí este módulo calcula el bit de paridad de los datos enviados, se utiliza el bit de paridad Par en donde la cantidad de estados en alto debe ser par y como aquí se envían 4 datos en alto el BP debe ser 0 el cual efectivamente lo es.
- **Stop. –** es el bit de finalización de este protocolo, el cual tiene que estar en valor lógico alto y se comprueba que efectivamente lo está.

Input_tran. – cómo se indicó esta señal contiene los datos del paquete VHDL. Por lo tanto, tiene 2 líneas una denominada input_tran.data la que contiene los datos a transmitir que en este caso son **00110011** y la línea input_tran.valid la que indica el momento en el cual se debe iniciar con la transmisión de datos.

Output_tran. – En esta señal se van guardando los datos que se van obteniendo del movimiento de pines generado por la transmisión de datos. Al ser de tipo **protocol_type** contiene 2 líneas una denominada output_tran.data la que contiene los datos recibidos en la transmisión que en este caso son **00110011** los cuales concuerdan con los datos enviados por la línea input_tran.data y la línea Output_tran. Valid la que indica el momento en el cual finaliza la transmisión de datos.

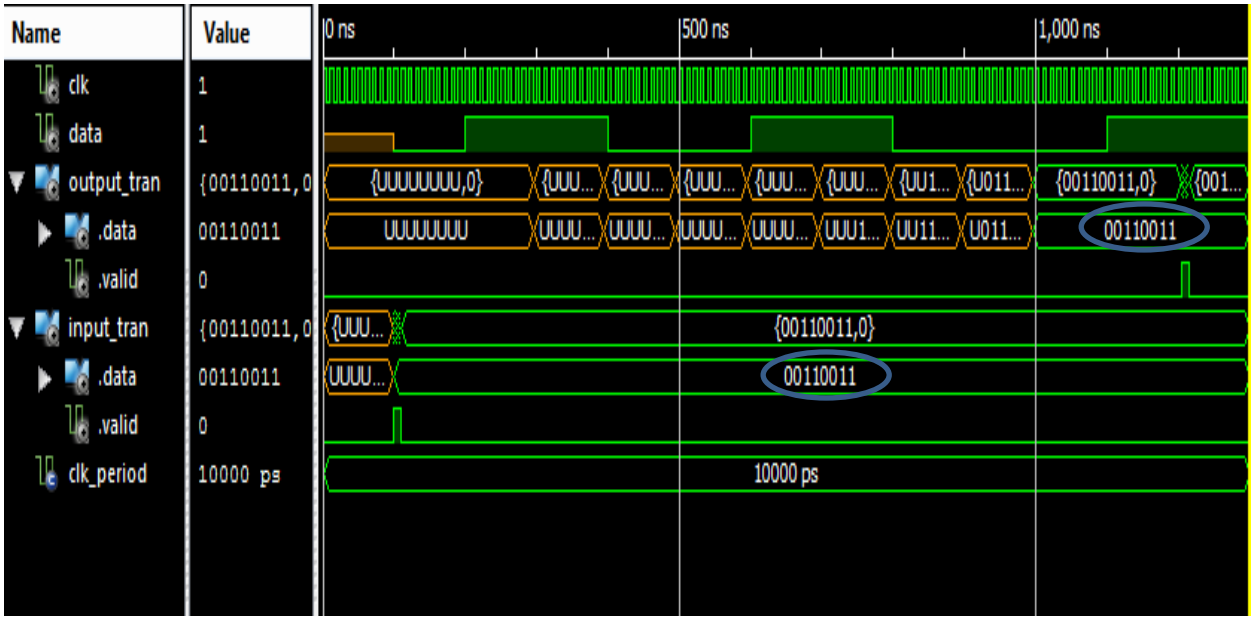


Figura 6-7. Simulación Monitor del protocolo UART

6.2.2 I2C

En esta simulación se verifica el correcto funcionamiento del protocolo UART, para lo cual se crean los respectivos módulos testbench para el Driver como para el Monitor. En estos módulos se ingresan los estímulos o datos que se quieren enviar, para lo cual se ingrese el código que se muestra a continuación en este apartado, el que se consta los datos que se desean transmitir.

```
stim_proc: process
   begin
      -- hold reset state for 100 ns.
               input_tran.valid <='0';
               wait for 100 ns;
               input_tran.slave_address <= "1100011";
               input_tran.reg_address <= "10101010";
               input_tran.data <= "00110011";
               input_tran.valid <='1';
               wait for clk_period;
               input_tran.valid <='0';
               wait for clk_period*10;
      wait;
   end process;
```

Estos estímulos se los ingresa a manera de proceso, en donde a la señal **input_tran.valid** se le asigna un valor lógico bajo '0', ahora se espera 100 nano segundos y se procede a cambiar el estado de esta señal **input_tran.valid** a un valor lógico alto '1', en esta parte también se le asigna el valor de los datos y las direcciones a las cuales se quiere enviar los datos. La primera señal es la dirección del esclavo a la que se denomina **input_tran.slave_address** y se le asigna el valor de **1100011**, la segunda señal es la dirección de registro a la que se denomina **input_tran.reg_address** y se le asigna el valor de **10101010**, la tercera señal son los datos para enviar a la que se denomina **input_tran.data** y se le asigna el valor de **00110011.** ahora se espera un ciclo de reloj que es equivalente a 10 nano segundos y se procede a cambiar el estado de esta señal **input_tran.valid** a un valor lógico bajo '0'. Se espera 10 ciclos de reloj y el proceso de estimulación se termina.

6.2.2.1 Package VHDL

Este es uno de los módulos que se genera y es en el que se describe la transacción que se desea enviar, este varía de acuerdo con la cantidad de datos que se desea enviar y a otras funcionalidades como son: si la transacción necesita identificar una dirección de dispositivo o una dirección de registro. Las cuales se encapsularán en un paquete VHDL como el que se muestra a continuación.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package protocol_common is

        type protocol_type is
                record
                        slave_address : STD_LOGIC_VECTOR (6 downto 0);
                        reg_address : STD_LOGIC_VECTOR (7 downto 0);
                        data : STD_LOGIC_VECTOR (7 downto 0);
                        valid : STD_LOGIC;
                end record;

end protocol_common;

package body protocol_common is

end protocol_common;
```

En este paquete se describe primeramente las librerías necesarias para la elaboración de este, en este caso solo se utiliza la librería principal la cual es la **IEEE** y hace referencia al estándar **IEEE.STD_LOGIC_1164**. A continuación de esto se empieza a describir el paquete al cual se lo denomina con el nombre de **protocol_common** y dentro de este se declara un tipo de protocolo el cual se denominará **protocol_type** el que contendrá un récord en el que se describen todas las señales que este paquete tendrá. En este caso el paquete contendrá 4 señales las cuales se des describe continuación:

Slave_address. –

Esta señal es un vector cuya longitud es de 7 bits y es en donde el estímulo asignara la dirección del dispositivo esclavo a donde se desean enviar los datos.

Reg_address. -

Esta señal es un vector cuya longitud es de 8 bits y es en donde el estímulo asignara la dirección de registro del dispositivo esclavo en donde se desean enviar los datos.

Data. –

Esta señal es un vector cuya longitud es de 8 bits y es en donde el estímulo asignara los datos que se desean enviar.

Valid. -

Esta señal es de tipo lógico por lo tanto solo será de 1 bit de longitud y es utilizada por el estímulo del testbench para iniciar con la transición de datos.

6.2.2.2 Driver

Este módulo es el encargado en convertir la transacción enviada en movimiento de pines y así generar las formas de onda para la transmisión del paquete de datos. En este módulo se describe al dispositivo físico y los puertos que este tendrá, los cuales pueden ser tanto pines de entrada como de salida, posteriormente se describe el movimiento que estos pines tendrán con el propósito de realizar la función de la transacción y enviar el paquete de datos. A continuación, se describe una parte del código del Driver generado en concreto se describe la entidad o caja negra.

Lo primero que se describe En este módulo son las librerías utilizadas, se utiliza la librería principal la cual es la **IEEE** y hace referencia al estándar **IEEE.STD_LOGIC_1164** en esta parte también se debe indicar que se utilizara un paquete de datos al cual se lo denomina como **work.protocol_common**, ahora se procede a describir la entidad de este módulo en donde se indica todos los puertos que este tendrá ya sean de entrada o de salida, también pueden ser bidireccionales, a continuación se describe los puertos necesarios para el funcionamiento de este protocolo.

- **Input_tran. –** Este es un puerto de entrada de tipo **protocol_type**, lo que quiere decir que por este puerto ingresaran o estarán los datos y las señales descritos en el paquete de datos anteriormente mencionado. Por lo tanto, en esta línea contendrá la dirección del dispositivo esclavo, la dirección de registro, el dato a enviar y la señal de Valid para iniciar la transmisión.
- **Clk. –** Esta es una señal de entrada de tipo lógico, es una señal de reloj utilizada por el software para el funcionamiento de los módulos que se desarrollan.
- **scl. –** Esta es una señal de salida de tipo lógico, Esta es una de las señales de salida que este protocolo tiene y es la señal de reloj con la cual se sincroniza con los demás dispositivos de la red y así poder enviar los datos.
- **sda. –** Esta es una señal de salida de tipo lógico, Esta es una de las señales de salida que este protocolo tiene y es por donde se van a enviar tanto las direcciones como los datos .

Después de definir la entidad se procede a diseñar la arquitectura de este módulo, en la cual se describe el movimiento de pines y cambios de señal que cada una de las líneas tiene que realizar a efecto de cumplir con la función de la transacción y enviar el paquete de datos deseado.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use work.protocol_common.ALL;

entity Driver is
        Port (
                input_tran : in protocol_type;
                clk : in  STD_LOGIC;
                scl : out STD_LOGIC;
                sda : out STD_LOGIC);
end Driver;
```

Ahora se procede a relizar la simulacion del modulo Driver y se ontiene como resultado las formas de onda mostradas en la figura 6 – 8. A continuacion se procede a describir cada una de estas señales y los datos que estan llevan.

Input_tran. – cómo se indicó esta señal contiene los datos del paquete VHDL. Por lo tanto, tiene 4 líneas una denominada input_tran.slave_address la que contiene la dirección de dispositivo a donde se envía el dato la cual es **11000011,** la otra se denominada input_tran.reg_address que contiene la dirección de registro que es **10101010**, la tercera se denominada input_tran.data la que contiene los datos a transmitir que en este caso son **00110011** y la línea input_tran.valid la que indica el momento en el cual se debe iniciar con la transmisión de datos.

Clk. – Es una señal de reloj interna que utiliza el software para sincronizar las líneas y determinar el tiempo de transmisión de cada estado lógico.

scl. – Esta es la señal de reloj propia del protocolo de comunicación y la cual es utilizada para la sincronización con los dispositivos a los cuales se transmiten los datos.

sda. – Esta es la línea por la cual se transmiten tanto las direcciones como los datos, a considerar que los cambios de estados se realizan solo cuando la línea scl se encuentra en nivel lógico bajo, con excepción del inicio y fin de la transmisión en donde el cambio de estado se realiza cuando la línea scl se encuentra en alto, en esta línea existen diferentes etapas las cuales están enumeradas en la figura 6 – 8 y a continuación se procede a detallarlas:

- **1. –** es el bit de inicio de este protocolo, el cual tiene que estar en valor lógico bajo y se comprueba que efectivamente lo está.
- **2. –** En esta parte se envía la dirección del dispositivo esclavo al cual se le va a enviar los datos empezando por el bit más significativo, en donde se puede observar que la dirección del esclavo es **1100011**, por lo tanto, se envía la dirección correcta.
- **3.** – Este bit determina si la acción que se quiere realizar es de lectura o escritura, en este caso al estar la señal a nivel lógico bajo quiere decir que es de tipo escritura.
- **4.** – Este bit determina si el paquete de datos ha sido recibido correctamente y se lo denomina bit de reconocimiento ACK, si se encuentra a nivel lógico bajo quiere decir que la información se ha recibido correctamente. En este caso se recibe un valor lógico bajo.
- **5. –** En esta parte se envía la dirección del registro del dispositivo al cual se le va a enviar los datos empezando por el bit más significativo, en donde se puede observar que la dirección de registro es **10101010**, por lo tanto, se envía la dirección de registro correcta.
- **6.** – Este bit determina si el paquete de datos ha sido recibido correctamente y se lo denomina bit de reconocimiento ACK, si se encuentra a nivel lógico bajo quiere decir que la información se ha recibido correctamente. En este caso se recibe un valor lógico bajo.

- **7. –** En esta parte se envía la dirección del registro del dispositivo al cual se le va a enviar los datos empezando por el bit más significativo, en donde se puede observar que la dirección de registro es **10101010**, por lo tanto, se envía la dirección de registro correcta.
- **8.** – Este bit determina si el paquete de datos ha sido recibido correctamente y se lo denomina bit de reconocimiento ACK, si se encuentra a nivel lógico bajo quiere decir que la información se ha recibido correctamente. En este caso se recibe un valor lógico bajo.
- **9. –** es el bit de finalización de este protocolo, el cual tiene que estar en valor lógico bajo y se comprueba que efectivamente lo está.

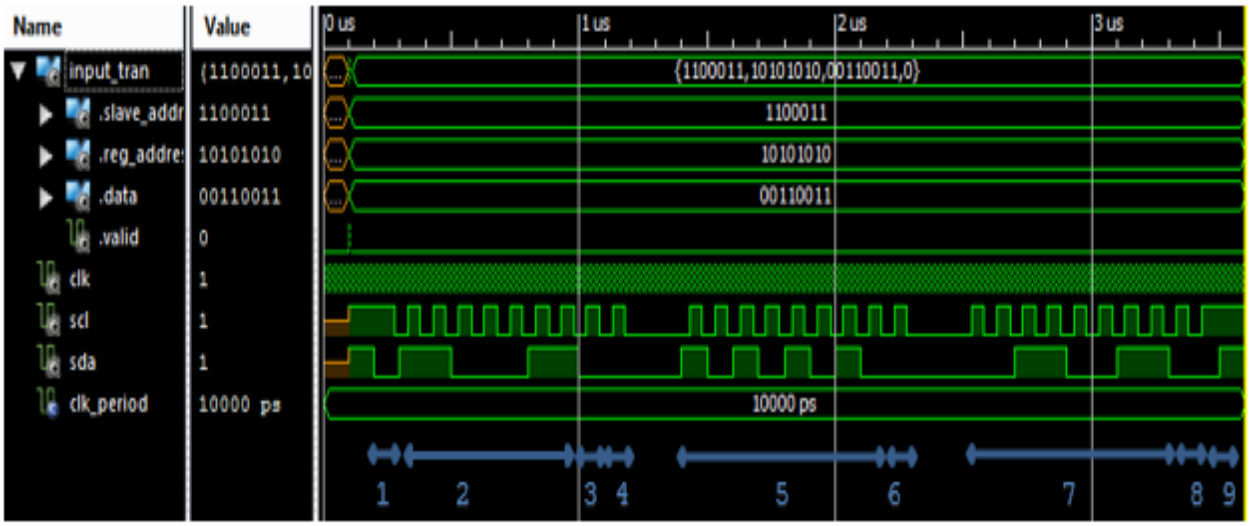


Figura 6-8. Simulación Driver del protocolo I2C

Además de las partes que tiene este protocolo y las cuales ya se describieron anteriormente, se tiene que tomar en cuanta algunas consideraciones como son el cambio de estados de la línea sda. Normalmente los cambios ocurren cuando la línea scl se encuentra en nivel lógico bajo sin embargo existe una excepción que es al inicio y al final de la transmisión de los datos ya que estos puntos los cambios de estados de la línea sda ocurren cuando la línea scl está a nivel lógico alto. Como se puede observar en la imagen 6-9 los cambios que ocurren en el inicio de la transmisión.

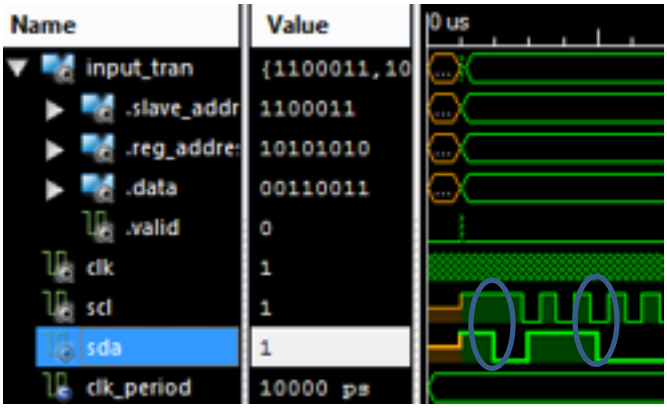


Figura 6-9. Secuencia de inicio del protocolo I2C

En la figura 6 – 10 de igual manera se indica como son los cambios de estado en las líneas sda al final de la transmisión y se puede observar que de igual manera normalmente los cambio se hacen cuando la señal scl está en bajo con excepción al final de la transmisión donde el cambio de estado de la señal sda se produce cuando la señal scl está en alto.

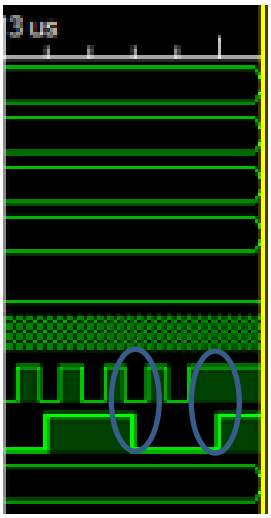


Figura 6-10. Secuencia de parada del protocolo I2C

6.2.2.3 Monitor

Este módulo es el encargado en convertir el movimiento de pines a transacciones y así poder verificar la correcta transmisión del paquete de datos. En este módulo se describe al dispositivo físico y los puertos que este tendrá, los cuales pueden ser tanto pines de entrada como de salida o pines bidireccionales, posteriormente se describe la forma en la que este módulo va tomando los cambios de estado de cada una de las líneas y va obtenido los datos enviados en la transacción. A continuación, se describe una parte del código del Monitor generado en concreto se describe la entidad o caja negra.

Lo primero que se describe En este módulo son las librerías utilizadas, se utiliza la librería principal la cual es la **IEEE** y hace referencia al estándar **IEEE.STD_LOGIC_1164** en esta parte también se debe indicar que se utilizara un paquete de datos al cual se lo denomina como **work.protocol_common**, ahora se procede a describir la entidad de este módulo en donde se indica todos los puertos que este tendrá ya sean de entrada o de salida, también pueden ser bidireccionales, a continuación se describe los puertos necesarios para el funcionamiento de este protocolo.

- **Output_tran. –** Este es un puerto bidireccional, pero se lo utiliza como puerto de salida que es de tipo **protocol_type**, lo que quiere decir que en este puerto se guardaran los datos obtenidos del movimiento de pines de las señales simuladas.
- **Clk. –** Esta es una señal de entrada de tipo lógico, es una señal interna de reloj utilizada por el software para el funcionamiento de los módulos que se desarrollan.
- **scl. –** Esta es una señal de entrada de tipo lógico, la cual es la señal de reloj utilizada por este protocolo para sincronizar los dispositivos con los cuales se quiere realizar la transmisión de los datos. Esta señal es la que se verificara los ciclos de reloj que esta produce.
- **sda. –** Esta es una señal de entrada de tipo lógico, esta es la línea por la cual se envía tanto las direcciones como los datos, está la señal que se tomara en cuenta para la recepción y posterior verificación de envío adecuado de datos.

Después de definir la entidad se procede a diseñar la arquitectura de este módulo, en la cual se describe la forma en la que este módulo va tomando los cambios de estado de cada una de las líneas y va obtenido las direcciones y los datos enviados en la transacción y así poder verificar el correcto funcionamiento del protocolo de comunicación analizado.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use work.protocol_common.ALL;

entity Monitor is
        Port (
                output_tran : inout protocol_type;
                clk : in  STD_LOGIC;
                scl : in STD_LOGIC;
                sda : in STD_LOGIC);
end Monitor;
```

Ahora se procede a relizar la simulacion del modulo Monitor y se obtiene como resultado las formas de onda mostradas en la figura 6 – 11. A continuacion se procede a describir cada una de estas señales y los datos que estan llevan.

Clk. – Es una señal de reloj interna que utiliza el software para sincronizar las líneas y determinar el tiempo de transmisión de cada estado lógico.

scl. – Ahora esta señal se la toma como una señal de entrada en el módulo Monitor y se va obteniendo los datos que esta va generando con sus cambios de estados y así determinar si esta señal cuenta con algún dato o dirección de transmisión.

sda. – de igual manera esta señal ahora se la toma como una entrada al módulo Monitor y se la tomando los datos que los cambios de estados que esta señal produce para así determinar tanto las direcciones como los datos de transmisión que está enviando el Driver.

- **1. –** es el bit de inicio de este protocolo, el cual tiene que estar en valor lógico bajo y se comprueba que efectivamente lo está.
- **2. –** En esta parte se envía la dirección del dispositivo esclavo al cual se le va a enviar los datos empezando por el bit más significativo, en donde se puede observar que la dirección del esclavo es **1100011**, por lo tanto, se envía la dirección correcta.
- **3.** – Este bit determina si la acción que se quiere realizar es de lectura o escritura, en este caso al estar la señal a nivel lógico bajo quiere decir que es de tipo escritura.
- **4.** – Este bit determina si el paquete de datos ha sido recibido correctamente y se lo denomina bit de reconocimiento ACK, si se encuentra a nivel lógico bajo quiere decir que la información se ha recibido correctamente. En este caso se recibe un valor lógico bajo.
- **5. –** En esta parte se envía la dirección del registro del dispositivo al cual se le va a enviar los datos empezando por el bit más significativo, en donde se puede observar que la dirección de registro es **10101010**, por lo tanto, se envía la dirección de registro correcta.
- **6.** – Este bit determina si el paquete de datos ha sido recibido correctamente y se lo denomina bit de reconocimiento ACK, si se encuentra a nivel lógico bajo quiere decir que la información se ha recibido correctamente. En este caso se recibe un valor lógico bajo.
- **7. –** En esta parte se envía la dirección del registro del dispositivo al cual se le va a enviar los datos empezando por el bit más significativo, en donde se puede observar que la dirección de registro es **10101010**, por lo tanto, se envía la dirección de registro correcta.
- **8.** – Este bit determina si el paquete de datos ha sido recibido correctamente y se lo denomina bit de reconocimiento ACK, si se encuentra a nivel lógico bajo quiere decir que la información se ha recibido correctamente. En este caso se recibe un valor lógico bajo.
- **9. –** es el bit de finalización de este protocolo, el cual tiene que estar en valor lógico bajo y se comprueba que efectivamente lo está.

Input_tran. – cómo se indicó esta señal contiene los datos del paquete VHDL. Por lo tanto, tiene 4 líneas, una denominada input_tran.slave_address la que contiene la dirección de dispositivo a donde se envía el dato la cual es **11000011,** la otra se denominada input_tran.reg_address que contiene la dirección de registro que es **10101010,** la otra se denominada input_tran.data la que contiene los datos a transmitir que en este caso son **00110011** y la línea input_tran.valid la que indica el momento en el cual se debe iniciar con la transmisión de datos.

Output_tran. – En esta señal se van guardando los datos que se van obteniendo del movimiento de pines generado por la transmisión de datos. Al ser de tipo **protocol_type** contiene 4 líneas, la primera es Output_tran.Slave_address en donde se guarda la dirección del dispositivo recibida en la transmisión que en este caso es **1100011**, la segunda es Output_tran.Reg_address en donde se guarda la dirección de registro recibida en la transmisión que en este caso es **10101010** y la tercera es output_tran.data la que contiene los datos recibidos en la transmisión que en este caso son **00110011**. Se puede verificar que evidentemente los datos que se obtuvieron en el Monitor son los mismos datos de la transacción que ingresa al Driver y finalmente la línea Output_tran.Valid la que indica el momento en el cual finaliza la transmisión de datos.

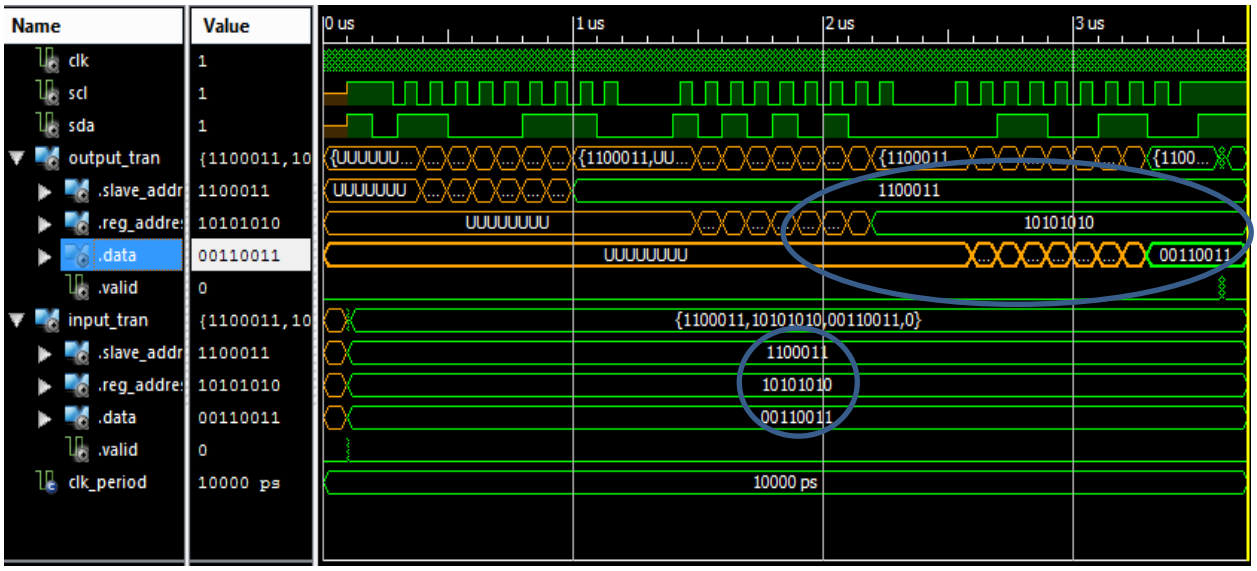


Figura 6-11. Simulación Monitor del protocolo I2C

Los datos obtenidos por el Monitor también se los puede comparar en el apartado name y value de la figura 6 – 11. Y verificar que evidentemente se obtiene los datos de la transacción que se ingresan al módulo Driver. Además, en la figura 6 – 12 se observan unas notas en las que se indica el momento en el que el Monitor recibe una señal de lectura o escritura de la transmisión y también se indica el momento en el que se recibe el bit de reconocimiento de cada transmisión tanto de las direcciones como del dato enviado.

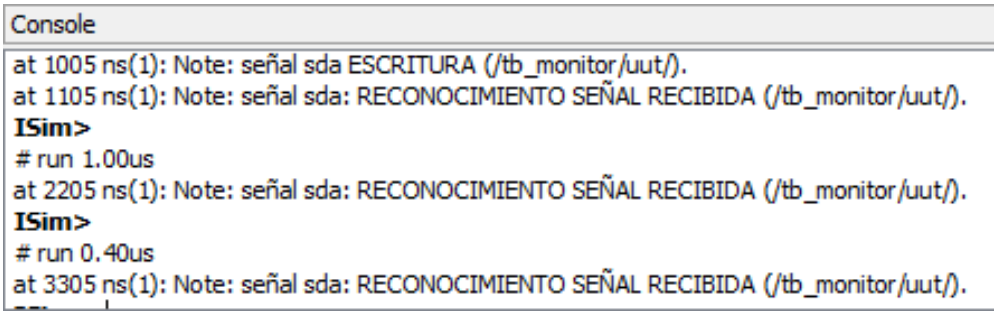


Figura 6-12. Notificación Consola del protocolo I2C

6.2.3 SPI

En esta simulación se verifica el correcto funcionamiento del protocolo UART, para lo cual se crean los respectivos módulos testbench para el Driver como para el Monitor. En estos módulos se ingresan los estímulos o datos que se quieren enviar, para lo cual se ingrese el código que se muestra a continuación en este apartado, el que se consta los datos que se desean transmitir.

```
stim_proc: process
   begin
      -- hold reset state for 100 ns.
               input_tran.valid <='0';
      wait for 100 ns;
               input_tran.data <= "0011001100110011";
               input_tran.valid <='1';
      wait for clk_period;
               input_tran.valid <='0';
               wait for clk_period*10;
      wait;
   end process;
```

Estos estímulos se los ingresa a manera de proceso, en donde a la señal **input_tran.valid** se le asigna un valor lógico bajo '0', ahora se espera 100 nano segundos y se procede a cambiar el estado de esta señal **input_tran.valid** a un valor lógico alto '1', en esta parte también se le asigna el valor de los datos a enviar a la señal **input_tran.**data la cual será un vector de 16 bits y en este caso se enviara los datos **"0011001100110011"**, ahora se espera un ciclo de reloj que es equivalente a 10 nano segundos y se procede a cambiar el estado de esta señal **input_tran.valid** a un valor lógico bajo '0'. Se espera 10 ciclos de reloj y el proceso de estimulación se termina.

6.2.3.1 Package VHDL

Este es uno de los módulos que se genera y es en el que se describe la transacción que se desea enviar, este varía de acuerdo con la cantidad de datos que se desea enviar y a otras funcionalidades como son: si la transacción necesita identificar una dirección de dispositivo o una dirección de registro. Las cuales se encapsularán en un paquete VHDL como el que se muestra a continuación.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package protocol_common is

        type protocol_type is
                record
                        data : STD_LOGIC_VECTOR (15 downto 0);
                        valid : STD_LOGIC;
                end record;

end protocol_common;

package body protocol_common is

end protocol_common;
```

En este paquete se describe primeramente las librerías necesarias para la elaboración de este, en este caso solo se utiliza la librería principal la cual es la **IEEE** y hace referencia al estándar **IEEE.STD_LOGIC_1164**. A continuación de esto se empieza a describir el paquete al cual se lo denomina con el nombre de **protocol_common** y dentro de este se declara un tipo de protocolo el cual se denominará **protocol_type** el que contendrá un récord en el que se describen todas las señales que este paquete tendrá. En este caso el paquete contendrá 2 señales las cuales se des describe continuación:

Data. –

Esta señal es un vector cuya longitud es de 16 bits y es en donde el estímulo asignara los datos que se desean enviar en la transmisión.

Valid. -

Esta señal es de tipo lógico por lo tanto solo será de 1 bit de longitud y es utilizada por el estímulo del testbench para iniciar con la transición de datos.

6.2.3.2 Driver

Este módulo es el encargado en convertir la transacción enviada en movimiento de pines y así generar las formas de onda para la transmisión del paquete de datos. En este módulo se describe al dispositivo físico y los puertos que este tendrá, los cuales pueden ser tanto pines de entrada como de salida, posteriormente se describe el movimiento que estos pines tendrán con el propósito de realizar la función de la transacción y enviar el paquete de datos. A continuación, se describe una parte del código del Driver generado en concreto se describe la entidad o caja negra.

Lo primero que se describe En este módulo son las librerías utilizadas, se utiliza la librería principal la cual es la **IEEE** y hace referencia al estándar **IEEE.STD_LOGIC_1164** en esta parte también se debe indicar que se utilizara un paquete de datos al cual se lo denomina como **work.protocol_common**, ahora se procede a describir la entidad de este módulo en donde se indica todos los puertos que este tendrá ya sean de entrada o de salida, también pueden ser bidireccionales, a continuación se describe los puertos necesarios para el funcionamiento de este protocolo.

- **Input_tran. –** Este es un puerto de entrada de tipo **protocol_type**, lo que quiere decir que por este puerto ingresaran o estarán los datos y las señales descritos en el paquete de datos anteriormente mencionado. Por lo tanto, en esta línea contendrá la dirección del dispositivo esclavo, la dirección de registro, el dato a enviar y la señal de Valid para iniciar la transmisión.
- **Clk. –** Esta es una señal de entrada de tipo lógico, es una señal de reloj utilizada por el software para el funcionamiento de los módulos que se desarrollan.
- **sclk. –** Esta es una señal de salida de tipo lógico, Esta es una de las señales de salida que este protocolo tiene y es la señal de reloj con la cual se sincroniza con los demás dispositivos de la red y así poder enviar los datos.
- **mosi. –** Esta es una señal de salida de tipo lógico, Esta es una de las señales de salida que este protocolo tiene y es por donde se van a enviar los datos que se transmiten por este protocolo, esta es la señal que envía los datos desde el dispositivo maestro al esclavo.
- **ss. –** Esta es una señal de salida de tipo lógico, Esta es una de las señales de salida y es la utilizada para la sincronización con el esclavo y se activa en el momento en el que los datos se están transmitiendo hacia algún dispositivo.

Después de definir la entidad se procede a diseñar la arquitectura de este módulo, en la cual se describe el movimiento de pines y cambios de señal que cada una de las líneas tiene que realizar a efecto de cumplir con la función de la transacción y enviar el paquete de datos deseado.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use work.protocol_common.ALL;

entity Driver is
        Port (
                input_tran : in protocol_type;
                clk : in  STD_LOGIC;
                sclk : out STD_LOGIC;
                mosi : out STD_LOGIC;
                ss : out STD_LOGIC);
end Driver;
```

Ahora se procede a relizar la simulacion del modulo Driver y se ontiene como resultado las formas de onda mostradas en la figura 6 – 13. A continuacion se procede a describir cada una de estas señales y los datos que estan llevan.

- **Input_tran.** – cómo se indicó esta señal contiene los datos del paquete VHDL. Por lo tanto, tiene 2 líneas una denominada input_tran.data la que contiene los datos a transmitir que en este caso son **0011001100110011** y la línea input_tran.valid la que indica el momento en el cual se debe iniciar con la transmisión de datos.
- **Clk.** – Es una señal de reloj interna que utiliza el software para sincronizar las líneas y determinar el tiempo de transmisión de cada estado lógico.
- **sclk.** – Esta es la señal de reloj propia del protocolo de comunicación y la cual es utilizada para la sincronización con los dispositivos a los cuales se transmiten los datos.
- **mosi.** – esta es la línea por la cual el dispositivo maestro envía los bits del paquete de datos hacia un dispositivo esclavo loa envía bit a bit y sincronizados por la señal sclk que este protocolo posee para el envío de datos.
- **ss.** – Esta línea es una señal que se activa en el momento que se está transmitiendo un paquete de datos hacia algún dispositivo y permanece activa durante todo el periodo que dura la transmisión de dichos datos.

Cuando se activa este protocolo la señal sclk empieza a generar los pulsos de reloj y paralelamente si el dispositivo desea enviar un dato la señal ss se activa, entonces empieza la señal mosi empieza con la transmisión del paquete de datos una vez que se han transmitido todos los datos del paquete la señal ss se desactiva y así termina el proceso de transmisión de datos en este protocolo. En la figura 6 – 13 se puede apreciar de manera gráfica el funcionamiento de este protocolo.

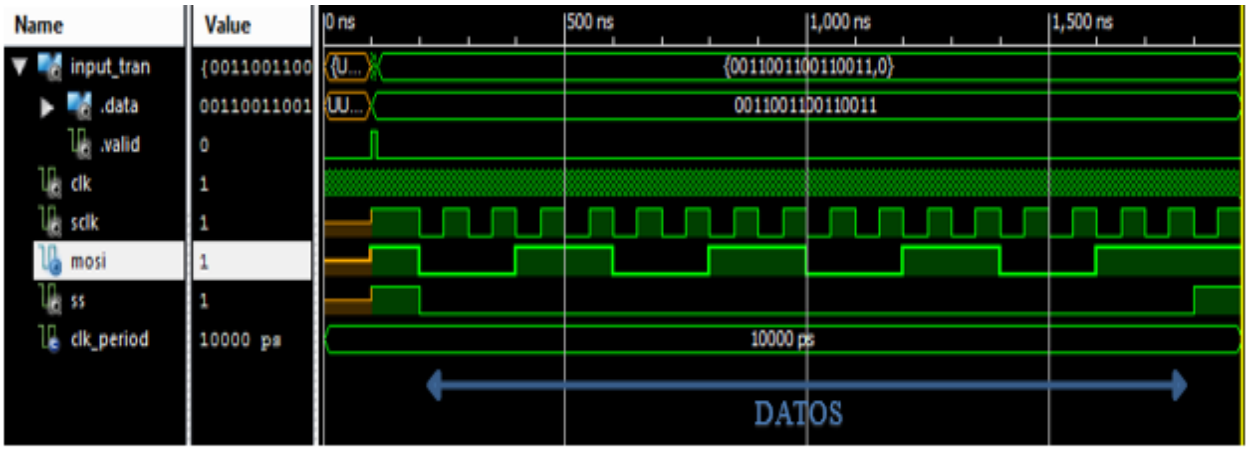


Figura 6-13. Simulación Driver del protocolo SPI

6.2.3.3 Monitor

Este módulo es el encargado en convertir el movimiento de pines a transacciones y así poder verificar la correcta transmisión del paquete de datos. En este módulo se describe al dispositivo físico y los puertos que este tendrá, los cuales pueden ser tanto pines de entrada como de salida o pines bidireccionales, posteriormente se describe la forma en la que este módulo va tomando los cambios de estado de cada una de las líneas y va obtenido los datos enviados en la transacción. A continuación, se describe una parte del código del Monitor generado en concreto se describe la entidad o caja negra.

Lo primero que se describe En este módulo son las librerías utilizadas, se utiliza la librería principal la cual es la **IEEE** y hace referencia al estándar **IEEE.STD_LOGIC_1164** en esta parte también se debe indicar que se utilizara un paquete de datos al cual se lo denomina como **work.protocol_common**, ahora se procede a describir la entidad de este módulo en donde se indica todos los puertos que este tendrá ya sean de entrada o de salida, también pueden ser bidireccionales, a continuación se describe los puertos necesarios para el funcionamiento de este protocolo.

- **Output_tran. –** Este es un puerto bidireccional, pero se lo utiliza como puerto de salida que es de tipo **protocol_type**, lo que quiere decir que en este puerto se guardaran los datos obtenidos del movimiento de pines de las señales simuladas.
- **Clk. –** Esta es una señal de entrada de tipo lógico, es una señal interna de reloj utilizada por el software para el funcionamiento de los módulos que se desarrollan.
- **sclk. –** Esta es una señal de entrada de tipo lógico, la cual es la señal de reloj utilizada por este protocolo para sincronizar los dispositivos con los cuales se quiere realizar la transmisión de los datos. Esta señal es en la que se verificara los ciclos de reloj que esta produce.
- **mosi. –** Esta es una señal de entrada de tipo lógico, esta es la señal por donde se transmite el paquete de datos y es la señal que se tomara en cuenta para que el módulo del Monitor reciba estos cambios de niveles lógicos y los convierta en los datos transmitidos por el Driver y así poder verificar el correcto funcionamiento de este protocolo.
- **ss. –** Esta es una señal de entrada de tipo lógico, esta señal se activa todo el momento en el que se está realizando la transmisión de datos, por lo cual es fundamental al momento de la verificación ya que será la que nos indique cuando se está enviando datos y ese momento el Monitor guardará estos datos en su pin de salida.

Después de definir la entidad se procede a diseñar la arquitectura de este módulo, en la cual se describe la forma en la que este módulo va tomando los cambios de estado de cada una de las líneas y va obtenido las direcciones y los datos enviados en la transacción y así poder verificar el correcto funcionamiento del protocolo de comunicación analizado.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use work.protocol_common.ALL;

entity Monitor is
        Port (
                output_tran : inout protocol_type;
                clk : in  STD_LOGIC;
                sclk : in STD_LOGIC;
                mosi : in STD_LOGIC;
                ss : in STD_LOGIC);
end Monitor;
```

Ahora se procede a relizar la simulacion del modulo Monitor y se obtiene como resultado las formas de onda mostradas en la figura 6 – 14. A continuacion se procede a describir cada una de estas señales y los datos que estan llevan.

- **Clk. –** Es una señal de reloj interna que utiliza el software para sincronizar las líneas y determinar el tiempo de transmisión de cada estado lógico.
- **sclk. –** Ahora esta señal se la toma como una señal de entrada en el módulo Monitor y se va obteniendo los datos que esta va generando con sus cambios de estados y así determinar si esta señal cuenta con algún dato o dirección de transmisión.
- **mosi. –** de igual manera esta señal ahora se la toma como una entrada al módulo Monitor y va tomando los datos que los cambios de estados que esta señal produce para así determinar los datos de transmisión que está enviando el Driver.
- **ss. –** Esta señal ahora se la toma como entrada del módulo Monitor y gracias a este pin se puede determinar el momento en el cual el dispositivo está enviando datos hacia un esclavo y así obtener los datos de la transmisión y guardarlos en la señal del Monitor.

Input_tran. – cómo se indicó esta señal contiene los datos del paquete VHDL. Por lo tanto, tiene 2 líneas, una denominada input_tran.data la que contiene los datos a transmitir que en este caso son **0011001100110011** y la línea input_tran.valid la que indica el momento en el cual se debe iniciar con la transmisión de datos. Como se observa en la figura 6 – 14.

Output_tran. – En esta señal se van guardando los datos que se van obteniendo del movimiento de pines generado por la transmisión de datos. Al ser de tipo **protocol_type** contiene 2 líneas, la primera es output_tran.data la que contiene los datos recibidos en la transmisión que en este caso son **0011001100110011**. Se puede verificar que evidentemente los datos que se obtuvieron en el Monitor son los mismos datos de la transacción que ingresa al Driver y finalmente la línea Output_tran.Valid la que indica el momento en el cual finaliza la transmisión de datos. En la figura 6 – 14 se puede apreciar de forma gráfica como los datos que envía el Driver son los mismos que los datos que recibe el Monitor por lo tanto la metodología de verificación aplicada funciona correctamente al igual que el protocolo de comunicación.

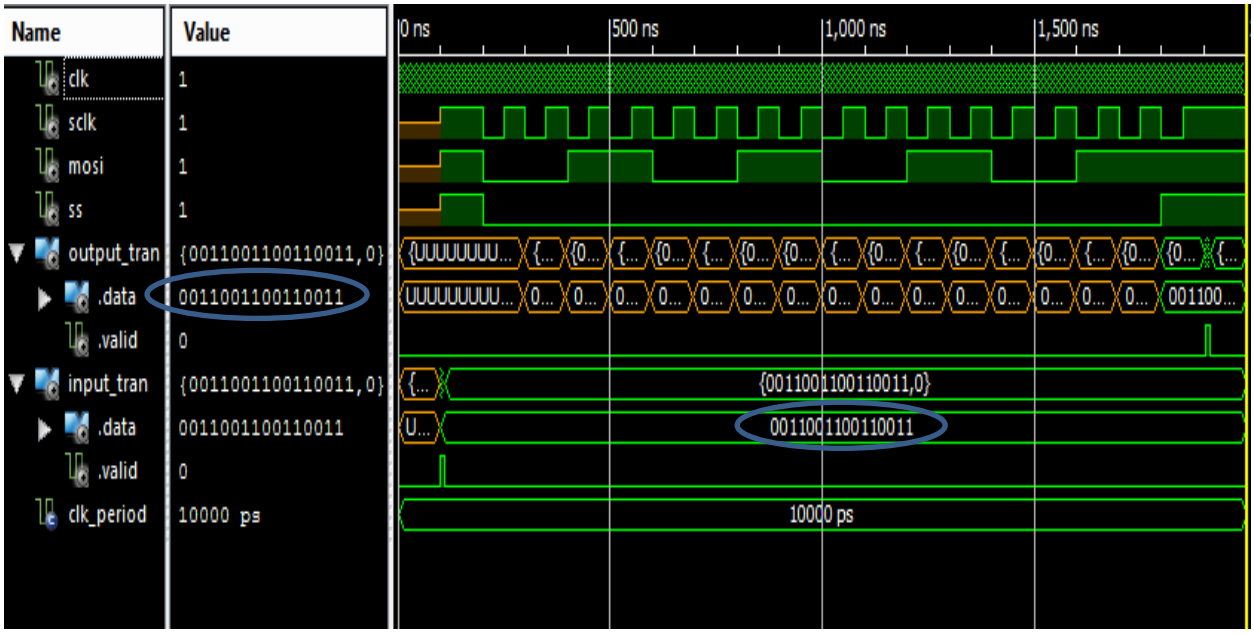


Figura 6-14. Simulación Monitor del protocolo SPI

6.2.4 Protocolo ficticio

En esta simulación se verifica el correcto funcionamiento del protocolo UART, para lo cual se crean los respectivos módulos testbench para el Driver como para el Monitor. En estos módulos se ingresan los estímulos o datos que se quieren enviar, para lo cual se ingrese el código que se muestra a continuación en este apartado, el que se consta los datos que se desean transmitir.

```
stim_proc: process
   begin
      -- hold reset state for 100 ns.
               input_tran.valid <='0';
      wait for 100 ns;
               input_tran.data <= x"abcd";
               input_tran.valid <='1';
      wait for clk_period;
               input_tran.valid <='0';
               wait for clk_period*10;
      wait;
   end process;
```

Estos estímulos se los ingresa a manera de proceso, en donde a la señal **input_tran.valid** se le asigna un valor lógico bajo '0', ahora se espera 100 nano segundos y se procede a cambiar el estado de esta señal **input_tran.valid** a un valor lógico alto '1', en esta parte también se le asigna el valor de los datos a enviar a la señal **input_tran.**data la cual será un vector de 16 bits y en este caso se enviara los datos **"1010101111001101"** en formato binario o **"abcd"** en formato hexadecimal , ahora se espera un ciclo de reloj que es equivalente a 10 nano segundos y se procede a cambiar el estado de esta señal **input_tran.valid** a un valor lógico bajo '0'. Se espera 10 ciclos de reloj y el proceso de estimulación se termina.

6.2.4.1 Package VHDL

Este es uno de los módulos que se genera y es en el que se describe la transacción que se desea enviar, este varía de acuerdo con la cantidad de datos que se desea enviar y a otras funcionalidades como son: si la transacción necesita identificar una dirección de dispositivo o una dirección de registro. Las cuales se encapsularán en un paquete VHDL como el que se muestra a continuación.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package protocol_common is

        type protocol_type is
                record
                        data : STD_LOGIC_VECTOR (15 downto 0);
                        valid : STD_LOGIC;
                end record;

end protocol_common;

package body protocol_common is

end protocol_common;
```

En este paquete se describe primeramente las librerías necesarias para la elaboración de este, en este caso solo se utiliza la librería principal la cual es la **IEEE** y hace referencia al estándar **IEEE.STD_LOGIC_1164**. A continuación de esto se empieza a describir el paquete al cual se lo denomina con el nombre de **protocol_common** y dentro de este se declara un tipo de protocolo el cual se denominará **protocol_type** el que contendrá un récord en el que se describen todas las señales que este paquete tendrá. En este caso el paquete contendrá 2 señales las cuales se des describe continuación:

Data. –

Esta señal es un vector cuya longitud es de 16 bits y es en donde el estímulo asignara los datos que se desean enviar en la transmisión, en este caso se realizaran 2 pruebas una enviando datos binarios y otra enviando datos en formato hexadecimal.

Valid. -

Esta señal es de tipo lógico por lo tanto solo será de 1 bit de longitud y es utilizada por el estímulo del testbench para iniciar con la transición de datos.

6.2.4.2 Driver

Este módulo es el encargado en convertir la transacción enviada en movimiento de pines y así generar las formas de onda para la transmisión del paquete de datos. En este módulo se describe al dispositivo físico y los puertos que este tendrá, los cuales pueden ser tanto pines de entrada como de salida, posteriormente se describe el movimiento que estos pines tendrán con el propósito de realizar la función de la transacción y enviar el paquete de datos. A continuación, se describe una parte del código del Driver generado en concreto se describe la entidad o caja negra.

Lo primero que se describe En este módulo son las librerías utilizadas, se utiliza la librería principal la cual es la **IEEE** y hace referencia al estándar **IEEE.STD_LOGIC_1164** en esta parte también se debe indicar que se utilizara un paquete de datos al cual se lo denomina como **work.protocol_common**, ahora se procede a describir la entidad de este módulo en donde se indica todos los puertos que este tendrá ya sean de entrada o de salida, también pueden ser bidireccionales, a continuación se describe los puertos necesarios para el funcionamiento de este protocolo.

- **Input_tran. –** Este es un puerto de entrada de tipo **protocol_type**, lo que quiere decir que por este puerto ingresaran o estarán los datos y las señales descritos en el paquete de datos anteriormente mencionado. Por lo tanto, en esta línea contendrá el dato a enviar y la señal de Valid para iniciar la transmisión.
- **Clk. –** Esta es una señal de entrada de tipo lógico, es una señal de reloj utilizada por el software para el funcionamiento de los módulos que se desarrollan y así sincronizar a las señales que se necesitan para el funcionamiento del protocolo.
- **data. –** Esta es una señal de salida de tipo vector de 4 bits de longitud, Esta es una de las señales de salida que este protocolo tiene y es por donde se van a enviar los datos que se transmiten por este protocolo, esta es la señal que envía los datos desde el dispositivo maestro al esclavo.
- **startp. –** Esta es una señal de salida de tipo lógico, Esta es una de las señales de salida que este protocolo tiene y es la encargada de marcar el inicio para que empiece la transmisión del paquete de datos hacia algún dispositivo.
- **endp. –** Esta es una señal de salida de tipo lógico, Esta es una de las señales de salida que este protocolo tiene y es la encargada de marcar el final para que termine la transmisión del paquete de datos que contiene la transacción.
- **ena. –** Esta es una señal de salida de tipo lógico, Esta es una de las señales de salida que este protocolo tiene y es la encargada de marcar tanto el inicio como el final de la transacción esta señal se activa durante todo el tiempo que dura la transmisión de datos.

Después de definir la entidad se procede a diseñar la arquitectura de este módulo, en la cual se describe el movimiento de pines y cambios de señal que cada una de las líneas tiene que realizar a efecto de cumplir con la función de la transacción y enviar el paquete de datos deseado.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use work.protocol_common.ALL;

entity Driver is
        Port (
                input_tran : IN protocol_type;
                clk : in  STD_LOGIC;
                data : OUT STD_LOGIC_VECTOR (3 downto 0);
                startp : OUT STD_LOGIC;
                endp : OUT STD_LOGIC;
                ena : OUT STD_LOGIC);
end Driver;
```

Ahora se procede a relizar la simulacion del modulo Driver y se ontiene como resultado las formas de onda mostradas en la figura 6 – 15 en formato binario de los datos enviados y en la figura 6 – 16 en formato hexadecimal de los datos enviados. A continuacion se procede a describir cada una de estas señales y los datos que estas llevan.

Input_tran. – cómo se indicó esta señal contiene los datos del paquete VHDL. Por lo tanto, tiene 2 líneas una denominada input_tran.data la que contiene los datos a transmitir en formato binario que en este caso son **1010101111001101** y la línea input_tran.valid la que indica el momento en el cual se debe iniciar con la transmisión de datos.

Por otro la se realizó otra prueba del mismo protocolo enviando datos hexadecimales, en donde igual tiene 2 líneas una denominada input_tran.data la que contiene los datos a transmitir en formato hexadecimal que en este caso son **abcd** y la línea input_tran.valid la que indica el momento en el cual se debe iniciar con la transmisión de datos.

- **Clk. –** Es una señal de reloj interna que utiliza el software para sincronizar las líneas y determinar el tiempo de transmisión de cada estado lógico.
- **data. –** Esta línea es la encargada de transmitir los datos como se puede apreciar inicialmente está en alta impedancia para luego empezar con la transmisión de datos en paquetes de 4 bits, una vez terminado de enviar todos los paquetes la señal procede a ponerse en alta impedancia.
- **startp. –** Esta línea es la encargada de indicar el momento En el cual se inicia la transmisión de datos, como se puede apreciar en las figuras esta normalmente se encuentra en estado lógico bajo, pero existe un momento en el cual esta cambia a nivel alto por un espacio de estado este es el detonante para el inicio de la transmisión de datos.
- **endp. –** Esta línea es la encargada de indicar el momento En el cual se finaliza la transmisión de datos, como se puede apreciar en las figuras esta normalmente se encuentra en estado lógico bajo, pero existe un momento en el cual esta cambia a nivel alto por un espacio de estado este es el momento en el cual se termina la transmisión de datos.
- **ena. –** Esta línea es la encargada de determinar el momento en el cual el protocolo está en funcionamiento, por lo tanto, esta se activa desde en instante que la señal startp se activa hasta el instante que la señal endp también se activa.

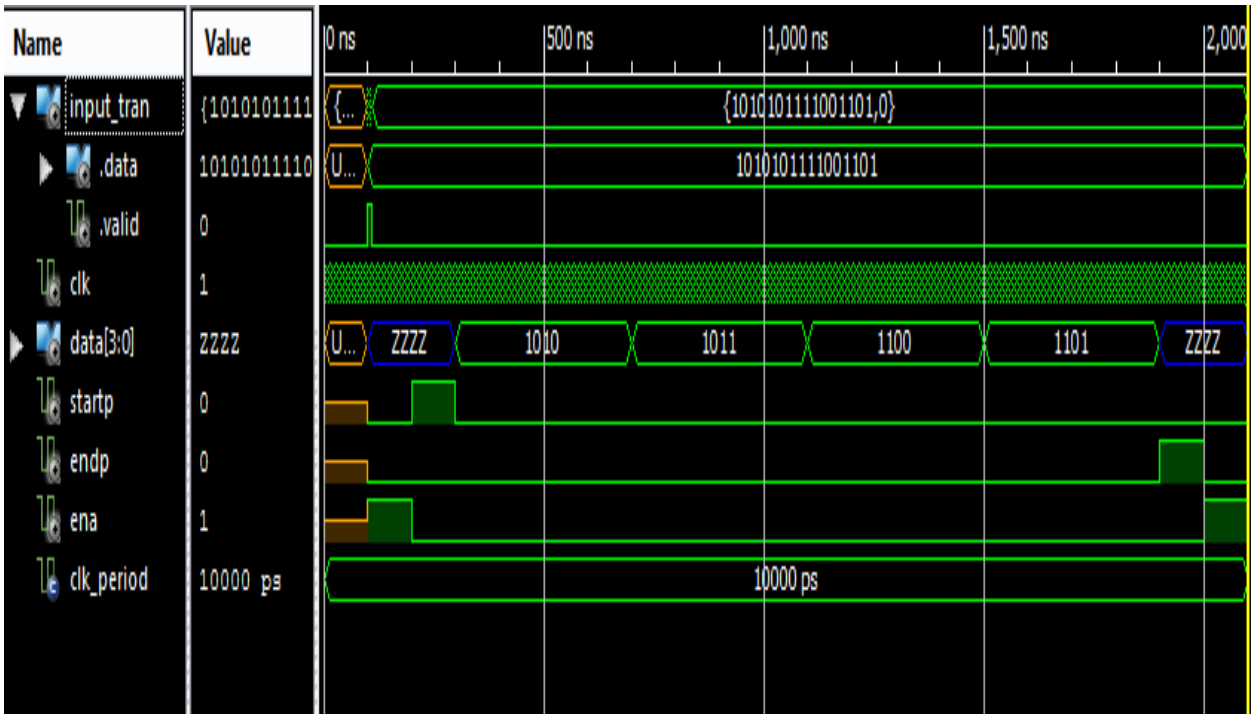


Figura 6-15. Simulación Driver Protocolo ficticio – formato binario

En la figura 6 – 15 se presenta la simulación del envío de un dato en formato binario se puede apreciar todos los cambios de estado que tiene las señales y el envío de los datos en paquetes de 4 bits. Mientras tanto en la figura 6 – 16 se presenta la simulación del envío de un dato en formato hexadecimal en donde se puede aprecia como se envían los datos en paquetes, en esta ocasión se envía 1 letra por cada paquete esto se debe a que cada letra contiene 4 bits además de igual manera se observa los cambios de estado que tienen las demás señales.

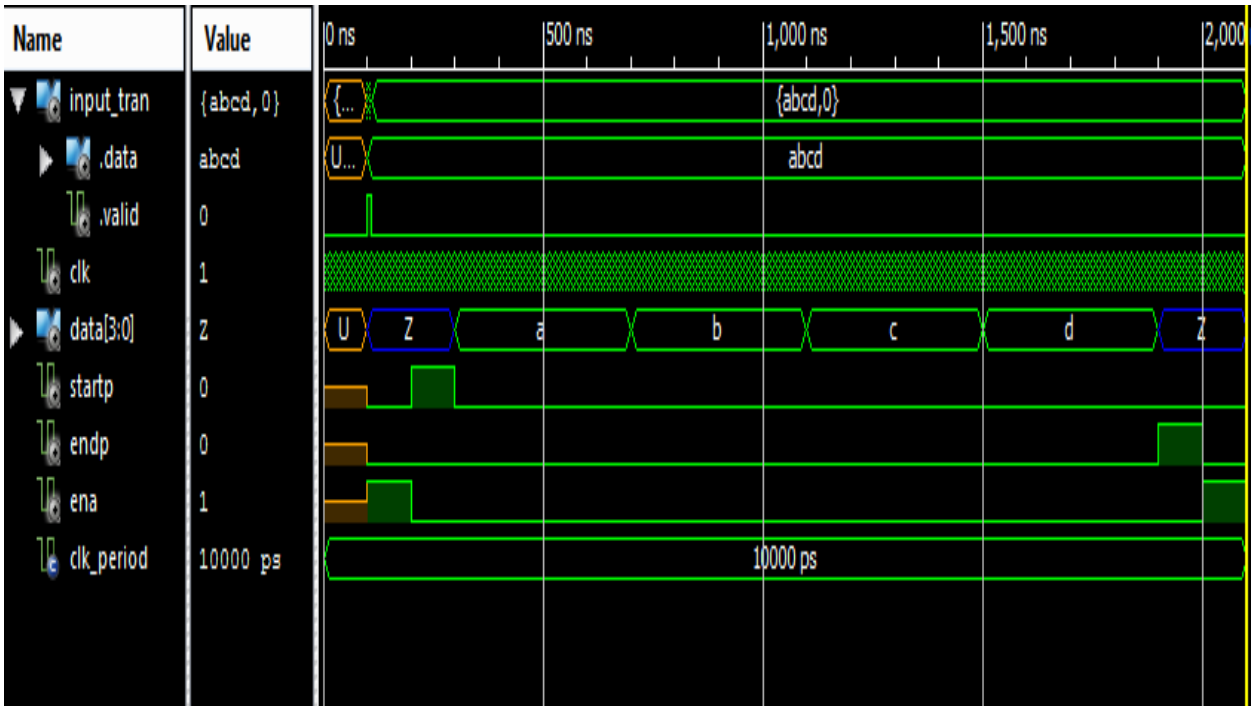


Figura 6-16. Simulación Driver Protocolo ficticio – formato hexadecimal

6.2.4.3 Monitor

Este módulo es el encargado en convertir el movimiento de pines a transacciones y así poder verificar la correcta transmisión del paquete de datos. En este módulo se describe al dispositivo físico y los puertos que este tendrá, los cuales pueden ser tanto pines de entrada como de salida o pines bidireccionales, posteriormente se describe la forma en la que este módulo va tomando los cambios de estado de cada una de las líneas y va obtenido los datos enviados en la transacción. A continuación, se describe una parte del código del Monitor generado en concreto se describe la entidad o caja negra.

Lo primero que se describe En este módulo son las librerías utilizadas, se utiliza la librería principal la cual es la **IEEE** y hace referencia al estándar **IEEE.STD_LOGIC_1164** en esta parte también se debe indicar que se utilizara un paquete de datos al cual se lo denomina como **work.protocol_common**, ahora se procede a describir la entidad de este módulo en donde se indica todos los puertos que este tendrá ya sean de entrada o de salida, también pueden ser bidireccionales, a continuación se describe los puertos necesarios para el funcionamiento de este protocolo.

- **Output_tran. –** Este es un puerto bidireccional, pero se lo utiliza como puerto de salida que es de tipo **protocol_type**, lo que quiere decir que en este puerto se guardaran los datos obtenidos del movimiento de pines de las señales simuladas.
- **Clk. –** Esta es una señal de entrada de tipo lógico, es una señal interna de reloj utilizada por el software para el funcionamiento de los módulos que se desarrollan.
- **data. –** Esta es una señal de entrada de tipo vector de 4 bits de longitud, esta es la señal por donde se transmite el paquete de datos y es la señal que se tomara en cuenta para que el módulo del Monitor reciba estos cambios de niveles lógicos y los convierta en los datos transmitidos por el Driver y así poder verificar el correcto funcionamiento de este protocolo.
- **startp. –** Esta es una señal de entrada de tipo lógico, la cual marca el inicio de la transmisión de datos
- **endp. –** Esta es una señal de entrada de tipo lógico, la cual indica el fin de la transmisión del paquete de datos.
- **ena. –** Esta es una señal de entrada de tipo lógico, esta es la línea la cual determina el momento en el cual el protocolo está en funcionamiento y viene delimitada por el cambio de estado de startp y en cambio de estado de endp.

Después de definir la entidad se procede a diseñar la arquitectura de este módulo, en la cual se describe la forma en la que este módulo va tomando los cambios de estado de cada una de las líneas y va obtenido las direcciones y los datos enviados en la transacción y así poder verificar el correcto funcionamiento del protocolo de comunicación analizado.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use work.protocol_common.ALL;

entity Monitor is
        Port (
                output_tran : INOUT protocol_type;
                clk : IN  STD_LOGIC;
                data : IN STD_LOGIC_VECTOR (3 downto 0);
                startp : IN STD_LOGIC;
                endp : IN STD_LOGIC;
                ena : IN STD_LOGIC);
end Monitor;
```

Ahora se procede a relizar la simulacion del modulo Monitor y se obtiene como resultado las formas de onda mostradas en la figura 6 – 17 del envio de un dato en formato binario mientras en la figura 6 – 18 se muestra el envio de un dato en formato hesadecimal. A continuacion se procede a describir cada una de estas señales y los datos que estan llevan.

- **Clk. –** Es una señal de reloj interna que utiliza el software para sincronizar las líneas y determinar el tiempo de transmisión de cada estado lógico.
- **data. –** de igual manera esta señal ahora se la toma como una entrada al módulo Monitor, y este módulo va tomando los cambios de estado y el envío de los paquetes de datos que esta señal tiene para compararlos con los datos ingresados al Driver y verificar si el protocolo esta funcionado de manera adecuada.
- **startp. –** Ahora esta señal se la toma como una señal de entrada en el módulo Monitor y se observa que es la señal que al cambiar de estado inicia la transmisión de datos por lo cual es importante para la verificación de la transacción.
- **endp. –** Esta señal ahora es una entrada al módulo Monitor, en la figura se observa que esta señal al cambiar de estado produce la finalización de la transmisión de datos.
- **ena. –** de igual manera esta señal que en el Driver era una señal de salida aquí en el Monitor se integra como entrada, en la figura se observa como esta señal se activa en el momento que está funcionando el protocolo y está delimitada por el cambio de estado tanto de la señal startp como de la señal endp.

Input_tran. – cómo se indicó esta señal contiene los datos del paquete VHDL. Por lo tanto, tiene 2 líneas, una denominada input_tran.data la que contiene los datos a transmitir que en este caso son **1010101111001101** en formato binario o **abcd** en formato hexadecimal y la línea input_tran.valid la que indica el momento en el cual se debe iniciar con la transmisión de datos.

Output_tran. – En esta señal se van guardando los datos que se van obteniendo del movimiento de pines generado por la transmisión de datos. Al ser de tipo **protocol_type** contiene 2 líneas, la primera es output_tran.data la que contiene los datos recibidos en la transmisión que en este caso son **1010101111001101** en formato binario o **abcd** en formato hexadecimal. Se puede verificar que evidentemente los datos que se obtuvieron en el Monitor son los mismos datos de la transacción que ingresa al Driver y finalmente la línea Output_tran.Valid la que indica el momento en el cual finaliza la transmisión de datos.

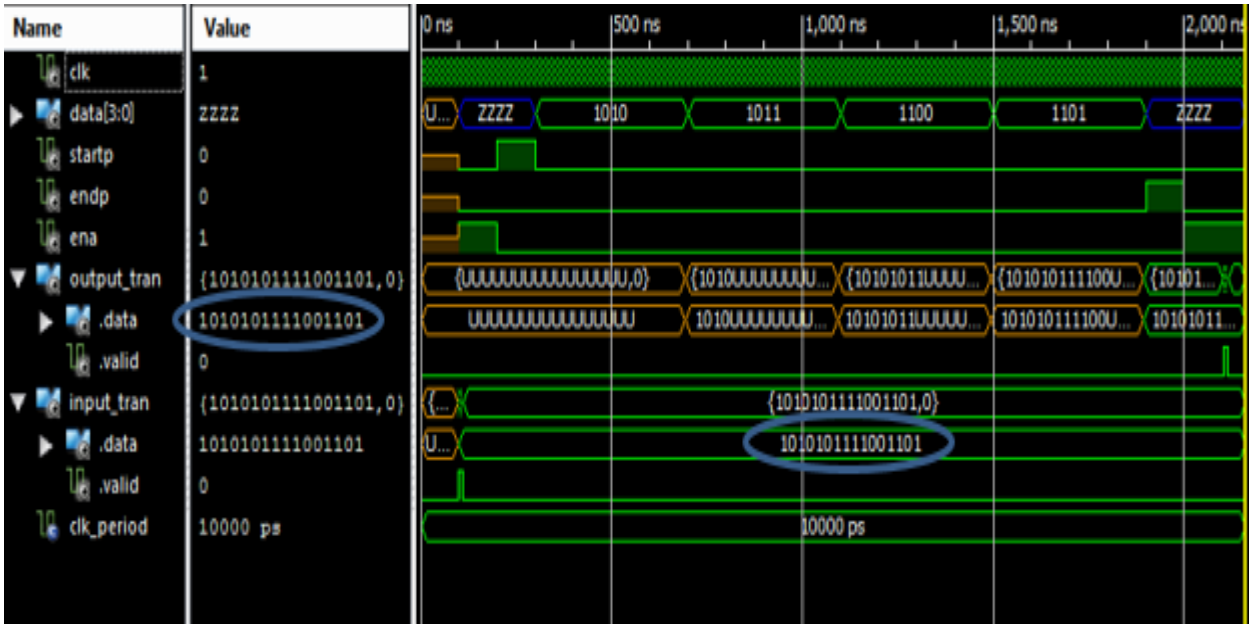


Figura 6-17. Simulación Monitor Protocolo ficticio – formato binario

En la figura 6 – 17 se presenta la simulación del envío y la recepción de un dato en formato binario se puede apreciar todos los cambios de estado que tiene las señales y el envío de los datos en paquetes de 4 bits así también como su recepción en paquetes de 4 bits. Mientras tanto en la figura 6 – 18 se presenta la simulación del envío y recepción de un dato en formato hexadecimal en donde se puede aprecia como se envían y se reciben los datos en paquetes, en esta ocasión se envía y se recibe 1 letra por cada paquete esto se debe a que cada letra contiene 4 bits además de igual manera se observa los cambios de estado que tienen las demás señales. Como resultado se concluye que gracias a la metodología de verificación utilizada se pudo constatar el correcto funcionamiento de este protocolo ficticio de comunicación.

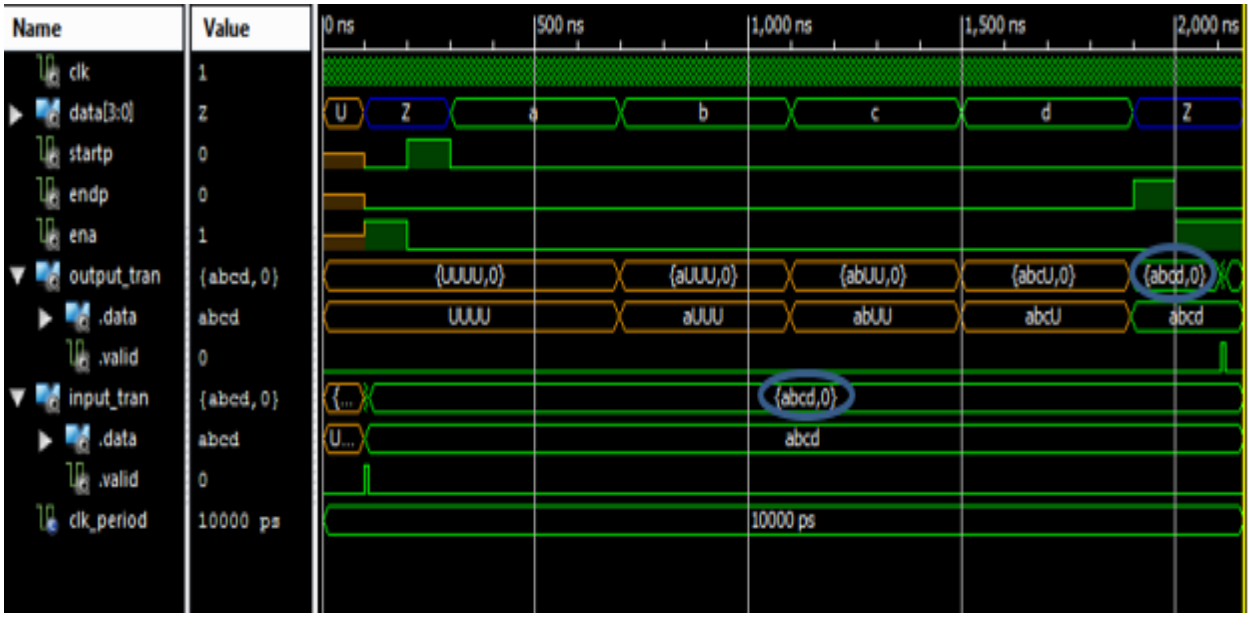


Figura 6-18. Simulación Monitor Protocolo ficticio – formato hexadecimal

7 CONCLUSIONES Y TRABAJOS FUTUROS

Cualquiera que deje de aprender es viejo, ya sea a los veinte u ochenta. Cualquier persona que sigue aprendiendo se mantiene joven

- Henry Ford-

EN este capítulo se presentan las conclusiones que se obtuvieron con el desarrollo de este proyecto y además se describen las posibles líneas futuras que este proyecto puede habilitar, así también como posibles trabajos de escalamiento con el propósito de mejorar la funcionalidad de este algoritmo que se desarrolló en este trabajo.

7.1 Conclusiones

Se ha implementado un algoritmo capaz de generar automáticamente los módulos Driver y Monitor en una sola ejecución del algoritmo mencionado y con la utilización de una sola transacción en la que se encapsulan todos los datos necesarios para el diseño de un circuito digital, la generación de estos módulos permite aparte del diseñar un circuito digital la verificación de su correcto funcionamiento mediante la capacidad de verificación funcional de un circuito digital.

El diseño y construcción de las transacciones para cada circuito digital se desarrolló de manera muy efectiva ya que aparte de contener los cambios de estado de cada una de las señales del circuito se logró introducir y encapsular en la misma transacción otros datos y funcionalidades muy importantes para la elaboración de los protocolos de comunicación, como son la selección de lectura o escritura, cálculo del bit de paridad.

Al ser el algoritmo desarrollado una herramienta capaz de crear no solo los Drivers sino también los Monitores de los circuitos digitales, se convierte en un algoritmo muy potente en el ámbito de verificación de circuitos, ya que en la actualidad la verificación suele consumir el doble de tiempo que se consume en el diseño. Y al generar de manera automática los Drivers y los Monitores se reduce el tiempo empleado en la etapa de diseño y verificación. Pudiendo así diseñar y verificar más circuitos en menos tiempo.

Las pruebas se realizaron a diferentes protocolos de comunicación, porque los protocolos tienen diferentes cambios de estado y con funcionalidades muy interesantes como selección de bit de paridad, selección de lectura o escritura, son una muestra significativa de protocolos de diversos tipos, en si son protocolos con varios cambios de estado es sus formas de onda. Es por eso por lo que se realizan las pruebas con estos circuitos para probar el nivel de efectividad que tiene el algoritmo desarrollado, el cual generó de manera satisfactoria los módulos VHDL para estos protocolos.

Para realizar la verificación del funcionamiento de los módulos generados se utiliza la arquitectura de un testbench TLM (modelado a nivel de transacciones), para ello se crean los respectivos testbench para cada módulo y se los simula mediante la arquitectura anteriormente mencionada, obteniendo como resultado que en todas pruebas realizadas se verificó que la información que se recibe como respuesta del Monitor es la misma información que ingresa el Driver para ser transmitida.

Posee una limitación la cual es que este algoritmo no puede crear en los módulos diseñados accesos a memoria de los dispositivos

7.2 Líneas futuras

Una de las líneas futuras más viables a corto plazo es la integración de otros lenguajes HDL ya que el algoritmo genera los módulos de los circuitos digitales en lenguaje VHDL, sería muy interesante que generara los módulos en formato Verilog y así incluir otro lenguaje HDL en el funcionamiento del algoritmo, para que los usuarios puedan diseñar y facilitar la tarea de verificación gracias a la herramienta y en los 2 lenguajes más utilizados en la descripción de hardware.

Wavedrom tiene muchas funcionalidades adicionales, accesibles a través de caracteres especiales en el JSON como la (x,=,h,l,p,etc) y códigos con los que se pueden diseñar las formas de onda de un sinfín de circuitos, sería algo muy novedoso que integraran más de estas funciones en el algoritmo que se desarrolló, ya que actualmente solo están integradas muy pocas de esas funciones. Si se integran más de estas funciones el algoritmo será capaz de reconocer y diseñar los módulos VHDL de las formas de onda de protocolos mucho más avanzados y complejos.

Aprovechando que se utiliza Python en el desarrollo del algoritmo serie muy interesante que en un futuro se utilizaran las librerías que este lenguaje nos brinda para enlazar Python con VHDL mediante subrutinas y simular los módulos VHDL en el mismo algoritmo y ya no se utilizaría un software adicional para realizar la simulación. Además, se podrían verificar el comportamiento de las formas de onda comparando estas graficas con las gráficas SVG creadas inicialmente.

Hacer al algoritmo más genérico al momento de generar los módulos, para esto se implementarán rutinas de los 2 lenguajes que se pretende utilizar VHDL y Verilog. También se puede implementar funciones para que el Driver reaccione a una señal que viene del mismo modulo haciendo que este cuente con pines bidireccionales.

REFERENCIAS

[1] IEEE std. 1800-2017, IEEE Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language, 2017

[2] J. Bergeron, E. Cerny, A. Hunter, A. Nightingale, ¨Verification Methodology Manual for SystemVerilog¨ , Ed. Springer, 2006

[3] Synopsis VC Verification IP. Synopsys Inc. https://www.synopsys.com/verification/verification-ip.html

[4] Mentor Verification IP, Mentor Inc. https://www.mentor.com/products/fv/verification-ip

[5] UVVM Github, https://github.com/UVVM/UVVM

[6] M. Sedghi, A. Shahabi, Z. Navabi, "TLM Studio for Transaction Level Simulation and Synthesis", University of Tehran, 2009.

[7] H. Guzmán, Capacidades de Verificación en Circuitos Digitales (Apuntes del Máster en Ingeniería electrónica, Robótica y automática)

[8] Xilinx XST User Guide

[9] Roth Jr, Charles H., Lizy Kurian, John; Digital Systems Design UsingVHDL, ed Thomson, USA 2008

[10] Pardo Carpio, Fernando; VHDL, Lenguaje para modelado y descripción de circuitos; Universidad de Valencia, 1997

[11] Diseño de sistemas digitales con VHDL. Santiago Fernández Gómez, Enrique Soto Campos, Serafín Pérez López, Santiago Fernández Gómez, Enrique Soto Campos, Serafín Pérez López. Ed. Paraninfo, 2006.

[12] Síntesis de circuitos digitales: un enfoque algorítmico. Jean Paul Deschamps, Ed. International Thomson Publishing, 2002

[13] RTL hardware design using VHDL. Coding for efficiency, portability, and scalability. Pong P. Chu. Ed. Wiley-Interscience, 2006

[14] VHDL-2008 Just the New Stuff. Peter Ashenden, Jim Lewis. 1º Edition. Ed. Morgan Kaufmann, 2008

[15] Graciani María, Generación automática de drivers de protocolo en VHDL para verificación, Universidad de Sevilla, 2018.

[16] M.A. Domínguez. Aportación al análisis del nivel de enlace en protocolos de comunicación para buses de campo normalizados. Tesis Doctoral, Departamento de tecnología Electrónica, Universidad de Vigo (España), septiembre de 2000.

[17] Salvador Cristina, IMPLEMENTACIÓN DE ANALIZADORES DE PROTOCOLOS DE COMUNICACIONES SPI, I2C. Universidad Carlos III de Madrid, Madrid, 2014.

GLOSARIO

ISO: International Organization for Standardization

IEEE: Institute of Electrical and Electronics Engineers

HDL: Hardware description language

VHDL: VHSIC-HDL, Very High-Speed Integrated Circuit Hardware Description Language

Verilog: Lenguaje de descripción hardware

VMM: Verification Methodology Manual)

OVM: Open Verification Methodology

UVM: Universal Verification Methodology

VIP: Verification IP

UVVM: Universal VHDL Verification Methodology

DUT: Dispositive under test

UART: Universal Asynchronous Receiver-Transmitter

I2C: Inter-Integrated Circuit

SPI: Serial Peripheral Interface

JSON: JavaScript Object Notation

SVG: Scalable Vector Graphics

TLM: Transaction-Level Modeling

STD: standard

INTEL: Integrated Electronics Corporation

ANEXOS

ANEXO A (Transacciones)

Protocolo UART

```
{
 "trans":
      [
        {"name": "data", "type":"vector","len": 8, "start": "LSB"}
      ],

 "signal":
      [
        {"name": "data", "wave": "l========xh", "type":"logic","len": 1,
        "ciclos": 10}
      ]
}
```

Protocolo I2C

```
{
 "trans":
  [
    {"name": "slave_address", "type":"vector", "len": 7, "start": "MSB"},
    {"name": "reg_address", "type":"vector", "len": 8, "start": "MSB"},
    {"name": "data", "type":"vector", "len": 8, "start": "MSB"}
  ],
 "signal":
  [
      {"name": "scl", "wave": "hhnnnnnnnnnllnnnnnnnnnllnnnnnnnnnnh",
      "type":"logic","len": 1, "ciclos": 10, "phase": 0.2},

      {"name": "sda", "wave": "hl=======34ll========4ll========4lh",
      "type":"logic","len": 1, "ciclos": 10, "R/W": "W","ACK/NACK":
      ["ACK","ACK","ACK"]}
  ]
}
```

Protocolo SPI

```
{
 "trans":
  [
    {"name": "data", "type":"vector", "len": 16, "start": "MSB"}
  ],

 "signal":
  [
      {"name": "sclk", "wave": "hnnnnnnnnnnnnnnnn", "type":"logic","len": 1,
      "ciclos": 10},

      {"name": "mosi", "wave": "h================h", "type":"logic","len": 1,
      "ciclos": 10},

      {"name": "ss", "wave": "hlllllllllllllllh", "type":"logic","len": 1,
      "ciclos": 10}
  ]
}
```

Protocolo Ficticio

```
{
 "trans":
 [
   {"name": "data", "type":"vector","len": 16, "start": "MSB"}
 ],

 "signal":
 [
      {"name":"data","wave":"zz=...=...=...=...zz", "type":"vector","len": 4,
      "ciclos": 10},

      {"name":"startp","wave":"lhlllllllllllllllll","type":"logic","len": 1,
      "ciclos": 10},

      {"name": "endp","wave":"llllllllllllllllhl", "type":"logic","len": 1,
      "ciclos": 10},

      {"name": "ena", "wave":"hllllllllllllllllh", "type":"logic","len": 1,
      "ciclos": 10}
 ]
}
```

ANEXO B (Algoritmo)

```
import json
import wavedrom
import sys

def crear_package():
    file = open ('protocol_common.vhdl','w')
    file.close()
def crear_Driver():
    file = open ('Driver.vhdl','w')
    file.close()
def crear_Monitor():
    file = open ('Monitor.vhdl','w')
    file.close()

f = open("trans.json", "r")
content = f.read()
jsondecoded = json.loads(content)
print(jsondecoded)
a=json.dumps(jsondecoded,indent=4)
print(a)
svg = wavedrom.render(a)
svg.saveas("demo1.svg")

if 'trans' in jsondecoded:
    crear_package()
    file = open('protocol_common.vhdl', 'a')
    file.write('library IEEE;\n')
    file.write('use IEEE.STD_LOGIC_1164.all;\n\n')
    file.write('package protocol_common is\n\n')
    file.write('\ttype protocol_type is\n')
    file.write('\t\trecord\n')

    for entity in jsondecoded['trans']:
        if entity['type'] == 'vector':
            file.write('\t\t\t'+entity["name"]+' : STD_LOGIC_VECTOR
            ('+str(entity["len"] - 1)+' downto 0);\n')
        if entity['type'] == 'logic':
            file.write('\t\t\t'+entity["name"]+' : STD_LOGIC;\n')
    file.write('\t\t\t'+'valid : STD_LOGIC;\n')
    file.write('\t\tend record;\n\n')
    file.write('end protocol_common;\n\n')
    file.write('package body protocol_common is\n\n')
    file.write('end protocol_common;\n')
    file.close()
else:
      sys.exit('Error: No se ha encontrado una transaccion por favor ingrese
      una transaccon valida')


if 'signal' in jsondecoded:
    crear_Driver()
    fin=len(jsondecoded['signal'])

    file = open('Driver.vhdl', 'a')
    file.write('library IEEE;\n')
    file.write('use IEEE.STD_LOGIC_1164.all;\n')
    file.write('use work.protocol_common.ALL;\n\n')
    file.write('entity Driver is\n')
    file.write('\tPort (\n')
    file.write('\t\tinput_tran : IN protocol_type;\n')
```

```
file.write('\t\tclk : in  STD_LOGIC;\n')
for entity in jsondecoded['signal']:
    pos = jsondecoded["signal"].index(entity)

    if entity['type'] == 'vector':
        file.write('\t\t'+entity["name"]+' : OUT STD_LOGIC_VECTOR
        ('+str(entity["len"] - 1)+' downto 0)')
    elif entity['type'] == 'logic':
        file.write('\t\t'+entity["name"]+' : OUT STD_LOGIC')
    if(pos < (fin-1)):
        file.write(';\n')
    else:
        file.write(');\n')
file.write('end Driver;\n')
file.write('\narchitecture Behavioral of Driver is\n')
file.write('\nbegin\n')

for entity in jsondecoded['signal']:

    file.write('\nproc_'+entity["name"]+' : process\n')
    file.write('\tbegin\n')
    file.write("\t\tif (input_tran.valid= '1') then\n")

    entitywave = entity['wave']
    num = len(entity['wave'])
    letter = entitywave[0]
    #print(letter)
    if entity['type'] == 'vector':
        if letter == 'z':
            file.write('\t\t\t'+entity["name"]+' ('+str(entity["len"] -
            1)+' downto 0) <= "')
             for i in range(entity['len']):
                 file.write('Z')
             file.write('";\n')
    elif entity['type'] == 'logic':
        if letter == 'h':
             file.write('\t\t\t' + entity["name"] + " <= '1';\n")
        elif letter == 'l':
             file.write('\t\t\t' + entity["name"] + " <= '0';\n")
    if 'phase' in entity:
        if entity["phase"] != 0:
             nph=entity["phase"]*entity["ciclos"]
             file.write('\t\t\tfor i in 0 to ' + str(entity["ciclos"] - 1
            - round(nph)) + ' loop\n')
        elif entity["phase"] == 0:
            file.write('\t\t\tfor i in 0 to ' + str(entity["ciclos"] - 1)
            + ' loop\n')
    else:
        file.write('\t\t\tfor i in 0 to ' + str(entity["ciclos"] - 1) + '
        loop\n')
    file.write('\t\t\t\twait until rising_edge(clk);\n')
    file.write('\t\t\tend loop;\n')

    c = 0
    d = 0
    con = 0
    aux = 0
    ba=0
    for j in range(1, num):
        entitywave = entity['wave']
        letter2 = entitywave[j]
        # print(letter2)
```

```
letter1 = entitywave[j - 1]
# pos = jsondecoded["signal"].index(entity)
if entity['type'] == 'logic':
    if letter2 == 'h':
        file.write('\t\t\t' + entity["name"] + " <= '1';\n")
           file.write('\t\t\tfor i in 0 to ' +
           str(entity["ciclos"] - 1) + ' loop\n')
        file.write('\t\t\t\twait until rising_edge(clk);\n')
        file.write('\t\t\tend loop;\n')
    elif letter2 == 'l':
        file.write('\t\t\t' + entity["name"] + " <= '0';\n")
           file.write('\t\t\tfor i in 0 to ' +
           str(entity["ciclos"] - 1) + ' loop\n')
        file.write('\t\t\t\twait until rising_edge(clk);\n')
        file.write('\t\t\tend loop;\n')
    elif letter2 == 'p':
        file.write('\t\t\t' + entity["name"] + " <= '1';\n")
           file.write('\t\t\tfor i in 0 to ' +
           str(int((entity["ciclos"] / 2))- 1) + ' loop\n')
        file.write('\t\t\t\twait until rising_edge(clk);\n')
        file.write('\t\t\tend loop;\n')
        file.write('\t\t\t' + entity["name"] + " <= '0';\n")
           file.write('\t\t\tfor i in 0 to ' +
           str(int((entity["ciclos"] / 2)) - 1) + ' loop\n')
        file.write('\t\t\t\twait until rising_edge(clk);\n')
        file.write('\t\t\tend loop;\n')
    elif letter2 == 'n':
        file.write('\t\t\t' + entity["name"] + " <= '0';\n")
           file.write('\t\t\tfor i in 0 to ' +
           str(int((entity["ciclos"] / 2)) - 1) + ' loop\n')
        file.write('\t\t\t\twait until rising_edge(clk);\n')
        file.write('\t\t\tend loop;\n')
        file.write('\t\t\t' + entity["name"] + " <= '1';\n")
           file.write('\t\t\tfor i in 0 to ' +
           str(int((entity["ciclos"] / 2)) - 1) + ' loop\n')
        file.write('\t\t\t\twait until rising_edge(clk);\n')
        file.write('\t\t\tend loop;\n')
    elif letter2 == '=':

        if (letter2 == letter1):
            if jsondecoded["trans"][c]["start"] == 'LSB':
                lsb = lsb + 1
                # print(lsb)
                  file.write('\t\t\t' + entity["name"] + ' <=
                  input_tran.' + jsondecoded["trans"][c]["name"] +
                  '(' + str(lsb) + ');\n')
                if lsb == msb:
                    c = c + 1
            elif jsondecoded["trans"][c]["start"] == 'MSB':
                msb = msb - 1
                # print(msb)
                  file.write('\t\t\t' + entity["name"] + ' <=
                  input_tran.' + jsondecoded["trans"][c]["name"] +
                  '(' + str(msb) + ');\n')
                if msb == lsb:
                    c = c + 1
        else:
            if jsondecoded["trans"][c]["start"] == 'LSB':
                lsb = 0
                msb = jsondecoded["trans"][c]["len"] - 1
```

```
                    file.write('\t\t\t' + entity["name"] + ' <=
                    input_tran.' + jsondecoded["trans"][c]["name"] +
                    '(' + str(lsb) + ');\n')
                elif jsondecoded["trans"][c]["start"] == 'MSB':
                    lsb = 0
                    msb = jsondecoded["trans"][c]["len"] - 1
                    file.write('\t\t\t' + entity["name"] + ' <=
                    input_tran.' + jsondecoded["trans"][c]["name"] +
                    '(' + str(msb) + ');\n')
        file.write('\t\t\tfor i in 0 to ' + str(entity["ciclos"] -
        1) + ' loop\n')
            file.write('\t\t\t\twait until rising_edge(clk);\n')
            file.write('\t\t\tend loop;\n')
    elif letter2 == 'x':
            file.write('\t\t\t' + entity["name"] + ' <= ')
            num_bp = jsondecoded["trans"][c-1]["len"]
            for i in range(num_bp):
                file.write('input_tran.' + jsondecoded["trans"][c-
                1]["name"] + '(' + str(i) + ')')
                if i < (num_bp - 1):
                    file.write(' XOR ')
                else:
                    file.write(';\n')
                file.write('\t\t\tfor i in 0 to ' +
                str(entity["ciclos"] - 1) + ' loop\n')
            file.write('\t\t\t\twait until rising_edge(clk);\n')
            file.write('\t\t\tend loop;\n')
    elif letter2 == '3':
            if 'R/W' in entity:
                if entity["R/W"] == 'R':
                        file.write('\t\t\t' + entity["name"] + " <=
                        '1';\n")

                        file.write('\t\t\tfor i in 0 to ' +
                        str(entity["ciclos"] - 1) + ' loop\n')

                        file.write('\t\t\t\twait until
                        rising_edge(clk);\n')
                    file.write('\t\t\tend loop;\n')
                elif entity["R/W"] == 'W':
                        file.write('\t\t\t' + entity["name"] + " <=
                        '0';\n")

                        file.write('\t\t\tfor i in 0 to ' +
                        str(entity["ciclos"] - 1) + ' loop\n')

                        file.write('\t\t\t\twait until
                        rising_edge(clk);\n')
                    file.write('\t\t\tend loop;\n')
            else:
                print("DRIVER: INGRESE UN MODO LECTURA O ESCRITURA")
    elif letter2 == '4':
            if 'ACK/NACK' in entity:
                if entity["ACK/NACK"][ba] == 'ACK':
                        file.write('\t\t\t' + entity["name"] + " <=
                        '0';\n")

                        file.write('\t\t\tfor i in 0 to ' +
                        str(entity["ciclos"] - 1) + ' loop\n')

                        file.write('\t\t\t\twait until
                        rising_edge(clk);\n')
```

```
                    file.write('\t\t\tend loop;\n')
                    ba=ba+1
                elif entity["ACK/NACK"][ba] == 'NACK':
                     file.write('\t\t\t' + entity["name"] + " <= '-
                     ';\n")

                     file.write('\t\t\tfor i in 0 to ' +
                     str(entity["ciclos"] *50) + ' loop\n')

                     file.write('\t\t\t\twait until
                     rising_edge(clk);\n')
                    file.write('\t\t\tend loop;\n')
                    ba=ba+1
                    break
            else:
                     print("DRIVER: INGRESE UN BIT DE ACENTAMIENTO
                     ACK O NACK")
    elif entity['type'] == 'vector':
        if letter2 == '=':
            if con == 0:
                cn = 0
                for entidad in entity['wave']:
                    if entidad.isalpha ():
                       cn =cn +1
                l_vec = num - cn
                #print(l_vec)
                msb_v = jsondecoded["trans"][d]["len"] - 1
                lsb_v = 0
                con = con + 1
            else:
                con = con + 1
            if jsondecoded["trans"][d]["start"] == 'MSB':
               file.write('\t\t\t' + entity["name"] + ' (' +
               str(entity["len"] - 1) + ' downto 0) <= input_tran.'
               +jsondecoded["trans"][d]["name"] + ' (' + str(msb_v) +
               ' downto ' + str(msb_v - entity["len"] + 1)+');\n')
                msb_v = msb_v - (entity['len'])
            elif jsondecoded["trans"][d]["start"] == 'LSB':
               file.write('\t\t\t' + entity["name"] + ' (' +
               str(entity["len"] - 1) + ' downto 0) <= input_tran.'
               +jsondecoded["trans"][d]["name"] + ' (' + str(lsb_v +
               entity["len"] - 1) + ' downto ' + str(lsb_v)+');\n')
                lsb_v = lsb_v + (entity['len'])
        elif letter2 == '.':
            if con == l_vec - 1:
                aux = 1
                d = d + 1
                con = 0
                pos = jsondecoded["signal"].index(entity)
                #print(pos)
            else:
                con = con + 1
               file.write('\t\t\tfor i in 0 to ' +
               str(entity["ciclos"] - 1) + ' loop\n')
        file.write('\t\t\t\twait until rising_edge(clk);\n')
        file.write('\t\t\tend loop;\n')
    if aux == 1:
        aux = 0
        file.write('\t\t\t' + jsondecoded['signal'][pos]['name'] + '
       (' + str(jsondecoded['signal'][pos]['len'] - 1) + ' downto 0)
       <= "')
        for i in range(jsondecoded['signal'][pos]['len']):
```

```
                file.write('Z')
            file.write('";\n')
    file.write("\t\tend if;\n")
    file.write("\twait on input_tran.valid;\n")
    file.write("end process proc_" + entity["name"] +";\n\n")
file.write("end Behavioral;\n")
file.close()


crear_Monitor()
file = open('Monitor.vhdl', 'a')
file.write('library IEEE;\n')
file.write('use IEEE.STD_LOGIC_1164.all;\n')
file.write('use work.protocol_common.ALL;\n\n')
file.write('entity Monitor is\n')
file.write('\tPort (\n')
file.write('\t\toutput_tran : INOUT protocol_type;\n')
file.write('\t\tclk : IN  STD_LOGIC;\n')
for entity in jsondecoded['signal']:
    pos = jsondecoded["signal"].index(entity)

    if entity['type'] == 'vector':
        file.write('\t\t'+entity["name"]+' : IN STD_LOGIC_VECTOR
        ('+str(entity["len"] - 1)+' downto 0)')
    elif entity['type'] == 'logic':
        file.write('\t\t'+entity["name"]+' : IN STD_LOGIC')
    if(pos < (fin-1)):
        file.write(';\n')
    else:
        file.write(');\n')
file.write('end Monitor;\n')
file.write('\narchitecture Behavioral of Monitor is\n')
file.write('\nbegin\n')

for entity in jsondecoded['signal']:
    entitywave = entity['wave']
    num = len(entity['wave'])
    letter = entitywave[0]
    #print(num)
    file.write('\nproc_' + entity["name"] + ' : process\n')
    if '=' in entitywave:
        if entity['type'] == 'vector':
            file.write('\tvariable
            data_aux:std_logic_vector('+str(entity["len"] - 1)+' downto
            0);\n')
        elif entity['type'] == 'logic':
            file.write('\tvariable data_aux:std_logic;\n')
    file.write('\tbegin\n')

    if entity['type'] == 'vector':
        if letter == 'z':
            file.write("\t\tif (" + entity['name'] + ' = "')
            for i in range(entity['len']):
                file.write('Z')
            file.write('") then\n')
    elif entity['type'] == 'logic':
        if letter == 'h':
            file.write("\t\tif (" + entity['name'] + " = '1') then\n")
        elif letter == 'l':
            file.write("\t\tif (" + entity['name'] + " = '0') then\n")
    if 'phase' in entity:
        if entity["phase"] != 0:
```

```
            nph=entity["phase"]*entity["ciclos"]
            file.write('\t\t\tfor i in 0 to ' + str(entity["ciclos"] - 1 -
            round(nph)) + ' loop\n')
        elif entity["phase"] == 0:
            file.write('\t\t\tfor i in 0 to ' + str(entity["ciclos"] - 1)
            + ' loop\n')
    else:
        file.write('\t\t\tfor i in 0 to ' + str(entity["ciclos"] - 1) + '
        loop\n')
    if entity['type'] == 'vector':
        if letter == 'z':
            file.write('\t\t\t\tassert(' + entity["name"] + '="')
            for i in range(entity['len']):
                file.write('Z')
            file.write('") report "UPS, señal ' + entity["name"] +' No se
            puso en alta impedancia" severity note;\n')
            file.write('\t\t\t\twait until falling_edge(clk);\n')
            file.write('\t\t\tend loop;\n')
    elif entity['type'] == 'logic':
        if letter == 'h':
            file.write('\t\t\t\tassert(' + entity["name"] + " = '1')
            report" + ' "señal ' + entity["name"] + ' incorrecta" severity
            note;\n')
            file.write('\t\t\t\twait until falling_edge(clk);\n')
            file.write('\t\t\tend loop;\n')
        elif letter == 'l':
            file.write('\t\t\t\tassert(' + entity["name"] + " = '0')
            report" + ' "señal ' + entity["name"] + ' incorrecta" severity
            note;\n')
            file.write('\t\t\t\twait until falling_edge(clk);\n')
            file.write('\t\t\tend loop;\n')
    c = 0
    d = 0
    con = 0
    ba=0
    for j in range(1, num):
        entitywave = entity['wave']
        letter2 = entitywave[j]
        if j < num-1:
            letter3 = entitywave[j + 1]
            #print(letter3)
        letter1 = entitywave[j - 1]
        # pos = jsondecoded["signal"].index(entity)
        if entity['type'] == 'logic':
            if letter2 == 'h':
              file.write('\t\t\tfor i in 0 to ' + str(entity["ciclos"] -
              1) + ' loop\n')

              file.write('\t\t\t\tassert('+entity["name"]+" = '1')
              report"+' "señal '+entity["name"]+' incorrecta" severity
              note;\n')
                file.write('\t\t\t\twait until falling_edge(clk);\n')
                file.write('\t\t\tend loop;\n')
            elif letter2 == 'l':
              file.write('\t\t\tfor i in 0 to ' + str(entity["ciclos"] -
              1) + ' loop\n')
              file.write('\t\t\t\tassert(' + entity["name"] + " = '0')
              report" + ' "señal ' + entity["name"] + ' incorrecta"
              severity note;\n')
                file.write('\t\t\t\twait until falling_edge(clk);\n')
                file.write('\t\t\tend loop;\n')
            elif letter2 == 'p':
```

```
                file.write('\t\t\tfor i in 0 to ' +
                str(int(entity["ciclos"]/2) - 1) + ' loop\n')
                file.write('\t\t\t\tassert(' + entity["name"] + " = '1')
                report" + ' "señal ' + entity["name"] + ' incorrecta"
                severity note;\n')
                  file.write('\t\t\t\twait until falling_edge(clk);\n')
                  file.write('\t\t\tend loop;\n')
                file.write('\t\t\tfor i in 0 to ' + str(int(entity["ciclos"]
                / 2) - 1) + ' loop\n')
                file.write('\t\t\t\tassert(' + entity["name"] + " = '0')
                report" + ' "señal ' + entity["name"] + ' incorrecta"
                severity note;\n')
                  file.write('\t\t\t\twait until falling_edge(clk);\n')
                  file.write('\t\t\tend loop;\n')
            elif letter2 == 'n':
                file.write('\t\t\tfor i in 0 to ' +
                str(int(entity["ciclos"]/2) - 1) + ' loop\n')
                file.write('\t\t\t\tassert(' + entity["name"] + " = '0')
                report" + ' "señal ' + entity["name"] + ' incorrecta"
                severity note;\n')
                  file.write('\t\t\t\twait until falling_edge(clk);\n')
                  file.write('\t\t\tend loop;\n')
                file.write('\t\t\tfor i in 0 to ' + str(int(entity["ciclos"]
                / 2) - 1) + ' loop\n')
                file.write('\t\t\t\tassert(' + entity["name"] + " = '1')
                report" + ' "señal ' + entity["name"] + ' incorrecta"
                severity note;\n')
                  file.write('\t\t\t\twait until falling_edge(clk);\n')
                  file.write('\t\t\tend loop;\n')
            elif letter2 == '=':
                if (letter2 == letter1):
                    if jsondecoded["trans"][c]["start"] == 'LSB':
                        lsb = lsb + 1
                        # print(lsb)

                          file.write('\t\t\tdata_aux:=' + entity["name"] +
                          ';\n')

                          file.write('\t\t\tfor i in 0 to ' +
                          str(entity["ciclos"] - 2) + ' loop\n')

                          file.write('\t\t\t\tassert (' + entity["name"] +
                          '=data_aux) report "ERROR EN LA TRANSMISION DE
                          DATOS" severity error ;\n')


                          file.write('\t\t\t\tdata_aux:=' + entity["name"]
                          + ';\n')
                        file.write('\t\t\tend loop;\n')

                          file.write('\t\t\toutput_tran.' +
                          jsondecoded["trans"][c]["name"] + '(' + str(lsb)
                          + ')<=data_aux;\n')

                          file.write('\t\t\twait until
                          falling_edge(clk);\n')
                        if lsb == msb:
                            c = c + 1
                    elif jsondecoded["trans"][c]["start"] == 'MSB':
                        msb = msb - 1
                        # print(msb)
```

```
                file.write('\t\t\tdata_aux:=' + entity["name"] +
                ';\n')

                file.write('\t\t\tfor i in 0 to ' +
                str(entity["ciclos"] - 2) + ' loop\n')

                file.write('\t\t\t\tassert (' + entity["name"] +
                '=data_aux) report "ERROR EN LA TRANSMISION DE
                DATOS" severity error ;\n')

                file.write('\t\t\t\twait until
                falling_edge(clk);\n')

                file.write('\t\t\t\tdata_aux:=' + entity["name"]
                + ';\n')
            file.write('\t\t\tend loop;\n')

                file.write('\t\t\toutput_tran.' +
                jsondecoded["trans"][c]["name"] + '(' + str(msb)
                + ')<=data_aux;\n')

                file.write('\t\t\twait until
                falling_edge(clk);\n')
            if msb == lsb:
                c = c + 1
    else:
        if jsondecoded["trans"][c]["start"] == 'LSB':
            lsb = 0
            msb = jsondecoded["trans"][c]["len"] - 1

                file.write('\t\t\tdata_aux:=' + entity["name"] +
                ';\n')

                file.write('\t\t\tfor i in 0 to ' +
                str(entity["ciclos"] - 2) + ' loop\n')

                file.write('\t\t\t\tassert
                ('+entity["name"]+'=data_aux) report "ERROR EN
                LA TRANSMISION DE DATOS" severity error ;\n')

                file.write('\t\t\t\twait until
                falling_edge(clk);\n')

                file.write('\t\t\t\tdata_aux:=' + entity["name"]
                + ';\n')
            file.write('\t\t\tend loop;\n')

                file.write('\t\t\toutput_tran.'+jsondecoded["tra
                ns"][c]["name"]+'('+str(lsb)+')<=data_aux;\n')

                file.write('\t\t\twait until
                falling_edge(clk);\n')
        elif jsondecoded["trans"][c]["start"] == 'MSB':
            lsb = 0
            msb = jsondecoded["trans"][c]["len"] - 1

                file.write('\t\t\tdata_aux:=' + entity["name"] +
                ';\n')

                file.write('\t\t\tfor i in 0 to ' +
                str(entity["ciclos"] - 2) + ' loop\n')
```

```
                    file.write('\t\t\t\tassert (' + entity["name"] +
                    '=data_aux) report "ERROR EN LA TRANSMISION DE
                    DATOS" severity error ;\n')

                    file.write('\t\t\t\twait until
                    falling_edge(clk);\n')

                    file.write('\t\t\t\tdata_aux:=' + entity["name"]
                    + ';\n')
                file.write('\t\t\tend loop;\n')

                    file.write('\t\t\toutput_tran.' +
                    jsondecoded["trans"][c]["name"] + '(' + str(msb)
                    + ')<=data_aux;\n')

                    file.write('\t\t\twait until
                    falling_edge(clk);\n')
elif letter2 == 'x':
  file.write('\t\t\tfor i in 0 to ' + str(entity["ciclos"] -
  1) + ' loop\n')
    file.write('\t\t\t\tdata_aux:=')
    num_bp = jsondecoded["trans"][c - 1]["len"]
    for i in range(num_bp):
        file.write('output_tran.' + jsondecoded["trans"][c -
        1]["name"] + '(' + str(i) + ')')
        if i < (num_bp - 1):
            file.write(' XOR ')
        else:
            file.write(';\n')
        file.write('\t\t\t\tassert (' + entity["name"] +
        '=data_aux) report "BIT DE PARIDAD ERRONEO" severity
        error ;\n')
    file.write('\t\t\t\twait until falling_edge(clk);\n')
    file.write('\t\t\tend loop;\n')
elif letter2 == '3':
    if 'R/W' in entity:
        if entity["R/W"] == 'R':

                    file.write('\t\t\tassert(' + entity["name"] + "
                    = '0') report" + ' "señal ' + entity["name"] + '
                    LECTURA" severity note;\n')

                    file.write('\t\t\twait until
                    falling_edge(clk);\n')

                    file.write('\t\t\tfor i in 0 to ' +
                    str(entity["ciclos"] - 2) + ' loop\n')

                    file.write('\t\t\t\tassert(' + entity["name"] +
                    " = '1') report" + ' "señal ' + entity["name"] +
                    ' incorrecta" severity note;\n')

                    file.write('\t\t\t\twait until
                    falling_edge(clk);\n')
                file.write('\t\t\tend loop;\n')
            elif entity["R/W"] == 'W':

                    file.write('\t\t\tassert(' + entity["name"] + "
                    = '1') report" + ' "señal ' + entity["name"] + '
                    ESCRITURA" severity note;\n')
```

```
                    file.write('\t\t\twait until
                    falling_edge(clk);\n')

                    file.write('\t\t\tfor i in 0 to ' +
                    str(entity["ciclos"] - 2) + ' loop\n')

                    file.write('\t\t\t\tassert(' + entity["name"] +
                    " = '0') report" + ' "señal ' + entity["name"] +
                    ' incorrecta" severity note;\n')

                    file.write('\t\t\t\twait until
                    falling_edge(clk);\n')
                  file.write('\t\t\tend loop;\n')
            else:
                print("MONITOR: INGRESE UN MODO LECTURA O ESCRITURA")
        elif letter2 == '4':
            if 'ACK/NACK' in entity:
                if entity["ACK/NACK"][ba] == 'ACK':

                    file.write('\t\t\tassert(' + entity["name"] + "
                    = '1') report" + ' "señal ' + entity["name"] +
                    ': ACENTAMIENTO SE'AL RECIBIDA" severity
                    note;\n')
                    file.write('\t\t\twait until
                    falling_edge(clk);\n')
                    file.write('\t\t\tfor i in 0 to ' +
                    str(entity["ciclos"] - 2) + ' loop\n')
                    file.write('\t\t\t\tassert(' + entity["name"] +
                    " = '0') report" + ' "señal ' + entity["name"] +
                    ' incorrecta" severity note;\n')
                    file.write('\t\t\t\twait until
                    falling_edge(clk);\n')
                  file.write('\t\t\tend loop;\n')
                  ba=ba+1
                elif entity["ACK/NACK"][ba] == 'NACK':
                    file.write('\t\t\tassert(' + entity["name"] + "
                    = '1') report" + ' "señal ' + entity["name"] +
                    ': NO ACENTAMIENTO SE'AL NO RECIBIDA" severity
                    note;\n')

                    file.write('\t\t\twait until
                    falling_edge(clk);\n')

                    file.write('\t\t\tfor i in 0 to ' +
                    str((entity["ciclos"] * 2)-2) + ' loop\n')

                    file.write('\t\t\t\tassert(' + entity["name"] +
                    " = '-') report" + ' "señal ' + entity["name"] +
                    'incorrecta" severity note;\n')

                    file.write('\t\t\t\twait until
                    falling_edge(clk);\n')
                  file.write('\t\t\tend loop;\n')
                  ba=ba+1
                  break
            else:
               print("MONITOR: INGRESE UN BIT DE ACENTAMIENTO ACK O
               NACK")
    elif entity['type'] == 'vector':
        if letter2 == 'z':
```

```
        file.write('\t\t\tfor i in 0 to ' + str(entity["ciclos"] -
        1) + ' loop\n')
            file.write('\t\t\t\tassert(' + entity["name"] + '="')
            for i in range(entity['len']):
                file.write('Z')
               file.write('") report "UPS, señal ' + entity["name"] +
               ' No se puso en alta impedancia" severity note;\n')
            file.write('\t\t\t\twait until falling_edge(clk);\n')
            file.write('\t\t\tend loop;\n')
    elif letter2 == '=':
        if con == 0:
            cn = 0
            for entidad in entity['wave']:
                if entidad.isalpha ():
                    cn =cn +1
            l_vec = num - cn
            #print(l_vec)
            msb_v = jsondecoded["trans"][d]["len"] - 1
            lsb_v = 0
            con = con + 1
        else:
            con = con + 1
        file.write('\t\t\tdata_aux:=' + entity["name"] + ';\n')
           file.write('\t\t\tfor i in 0 to ' +
           str(entity["ciclos"] - 1) + ' loop\n')
           file.write('\t\t\t\tassert (' + entity["name"] +
           '=data_aux) report "ERROR EN LA TRANSMISION DE DATOS"
           severity error ;\n')
        file.write('\t\t\t\twait until falling_edge(clk);\n')
        file.write('\t\t\t\tdata_aux:=' + entity["name"] + ';\n')
        file.write('\t\t\tend loop;\n')
    elif letter2 == '.':
        if letter3 != '.':
           file.write('\t\t\tdata_aux:=' + entity["name"] +
           ';\n')
            file.write('\t\t\tfor i in 0 to ' +
            str(entity["ciclos"] - 2) + ' loop\n')
            file.write('\t\t\t\tassert (' + entity["name"] +
            '=data_aux) report "ERROR EN LA TRANSMISION DE DATOS"
            severity error ;\n')
            file.write('\t\t\t\twait until falling_edge(clk);\n')

           file.write('\t\t\t\tdata_aux:=' + entity["name"] +
           ';\n')
            file.write('\t\t\tend loop;\n')
            if jsondecoded["trans"][d]["start"] == 'MSB':

                  file.write('\t\t\toutput_tran.' +
                  jsondecoded["trans"][d]["name"] + '(' +
                  str(msb_v) + ' downto ' + str(msb_v -
                  entity["len"] + 1) + ')<=data_aux;\n')
                  file.write('\t\t\twait until
                  falling_edge(clk);\n')
                  msb_v = msb_v - (entity['len'])
            elif jsondecoded["trans"][d]["start"] == 'LSB':

                  file.write('\t\t\toutput_tran.' +
                  jsondecoded["trans"][d]["name"] + '(' +
                  str(lsb_v + entity["len"] - 1) + ' downto ' +
                  str(lsb_v) + ')<=data_aux;\n')
```

```
                                file.write('\t\t\twait until
                                falling_edge(clk);\n')
                            lsb_v = lsb_v + (entity['len'])
                    else:
                        file.write('\t\t\tdata_aux:=' + entity["name"] +
                        ';\n')
                        file.write('\t\t\tfor i in 0 to ' +
                        str(entity["ciclos"] - 1) + ' loop\n')
                        file.write('\t\t\t\tassert (' + entity["name"] +
                        '=data_aux) report "ERROR EN LA TRANSMISION DE DATOS"
                        severity error ;\n')
                        file.write('\t\t\t\twait until falling_edge(clk);\n')

                        file.write('\t\t\t\tdata_aux:=' + entity["name"] +
                        ';\n')
                        file.write('\t\t\tend loop;\n')
                    if con == l_vec - 1:
                        d = d + 1
                        con = 0
                    else:
                        con = con + 1
            #print(d)
            #print(con)
            file.write("\t\t\toutput_tran.valid<='1';\n")
            file.write('\t\t\twait until falling_edge(clk);\n')
            file.write("\t\t\toutput_tran.valid<='0';\n")
            file.write('\t\telse\n')
            file.write("\t\t\toutput_tran.valid<='0';\n")
            file.write('\t\t\twait until falling_edge(clk);\n')
            file.write('\t\tend if;\n')
            file.write("end process proc_" + entity["name"] + ";\n\n")
        file.write("end Behavioral;\n")
        file.close()
else:
    sys.exit('Error: No se ha encontrado una señal por favor ingrese una
señal valida')
a = jsondecoded['signal'][-1]

print(a)
```