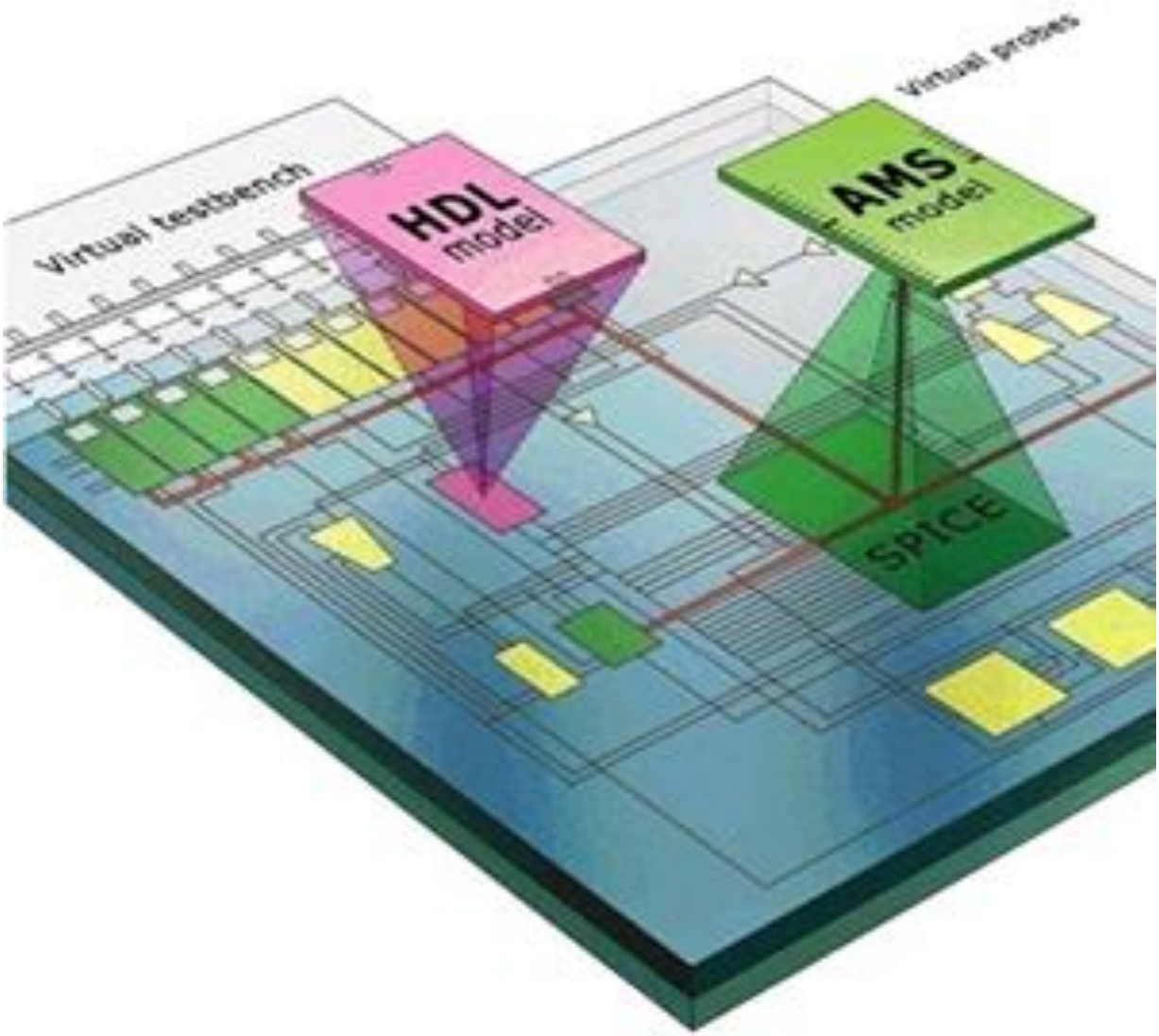
[Planeación y Verificación del Módulo de Comunicaciones UART]

Plan de Verificación y Validación

Versión [2.0] [Final]



Contenido

Integrantes

Aldo Alexandro Vargas Meza----------213495653
Marco Antonio Sandoval Canizales----------2105009955
Ramón Alonso Barba Mercado----------210401232

I. Objetivos

Este Plan de Verificación para el proyecto de verificación y validación del Protocolo de Comunicación UART, soporta los siguientes objetivos:

- Identificar la información de proyecto existente y los componentes de software que deben ser verificados.
- Verificación de los componentes del proyecto por separado, análisis por medio de ModelSim.
- Aleatorización de valores con monitoreo en tiempo real

El objetivo previsto para esta aplicación, es la desarrollar una verificación y comprobación de un protocolo básico de comunicación implementado en Lenguaje Descriptivo de Hardware Systemverilog, transmitiendo datos a través de un puerto serial estándar.

II. Introducción

La comunicación serial consiste en el envío secuencial de un bit a de información entre dos o más dispositivos, con una programación capaz de obtener una lectura libre de errores al manipular y controlar los aspectos de diferentes niveles de abstracción en el código que implementa, obteniendo con precisión en la recepción y transmisión de la información, corroborándolo al final de la misma.

Para diseñar este funcionamiento, fueron diseñado dos módulos que por separados hacen la función de transmisor de datos, y receptor de los mismos. Al ser diseñados se consideraron especificaciones del protocolo para poder implementarlo.

En el protocolo se especifica que u bit de Start es un 0 lógico, y el de Stop es un 1, cada valor enviado y recibido sigue el mismo protocolo orquestado por un nivel más alto de simulación y abstracción, el cual debe ser especificado en la programación interna de ambos módulos, puesto que la correcta comunicación entre ellos es el funcionamiento primario del módulo principal.

Debemos tener en cuenta también, la velocidad a la que se envían lo bits serialmente, para calcular los tiempos y sincronizar ambos módulos que operan por separado. Ambos dispositivos deben estar a la misma velocidad para transmitir y recibir bits, si no lo están, la recepción de datos podría iniciar en un punto en donde el dato a enviar no está preparado en su totalidad, lo cual ocasionaría una lectura incorrecta.

Por medio de este protocolo se pueden enviar cualquier tipo de información: letras, números y caracteres. Se puede enviar el código ASCII de una letra seguida de otra y así sucesivamente para formar palabras y valores numéricos. Una vez el protocolo pase los diferentes niveles de verificación, el módulo tendrá la posibilidad de ser entregado al cliente. El objetivo de la verificación es poder encontrar la mayor cantidad de defectos tempranamente con el objetivo de entregar un software de calidad al cliente, y evitando las cuestiones de rediseño ocasionadas por el descubrimiento de fallas sobre el final del proceso de desarrollo.

Para lograr esto, se requiere tener tanto una visión global del sistema, como visión en cada uno de los subsistemas que lo componen, y también de cada unidad de código fuente que componen los subsistemas. Por ese motivo la verificación se separa en 3 diferentes pruebas:

- Pruebas unitarias: Se verifica cada unidad de código fuente en cada uno de los módulos implementados, para determinar si contiene defectos en su implementación, comprobándolo contra el diseño del mismo y con las generalidades que maneja el proyecto completo.

- Pruebas de integración: Se verifica el correcto funcionamiento de las interacciones entre los módulos conectados, verificando las instancias contenidas en el código HDL, para determinar que no existen defectos en la comunicación de los mismos, y si la hay corregirla.

- Pruebas funcionales: Se genera una serie de impulsos aleatorios de manera que el sistema funcione lo más acercado a la realidad posible, después se verifica que el valor esperado sea correcto y que cumpla con las especificaciones impuestas.

III. Alcance

Para definir el alcance del proceso de verificación y validación es necesario especificar las responsabilidades que deben asumir los integrantes del equipo y aquellos que asumen una función dentro del proceso.

La verificación unitaria será responsabilidad del programador o implementador que le sea asignada, y consistirá en ir revisando de manera cautelosa cada unidad terminada o avanzada, así como crear un programa de verificación **test bench**, diseñado específicamente para la validación de resultados de ese único modulo, y que pueda evaluar el modulo en un conjunto de casos estipulados.

Las pruebas funcionales consistirán en pruebas basadas enteramente en el documento de especificaciones de requerimientos, para de esta manera comprobar Que el sistema cumple con ellas.

IV. Áreas Funcionales

Básicamente, el sistema de comunicación cuenta con los siguientes elementos:

- **Transmisor(Tx):** se encarga de una vez obteniendo los estímulos correctos por parte del módulo Test_bench empezar a manejar el dato a enviar, para poder llevar a cabo la transmisión ordenada después de un bit de inicio y finalizar la misma con un bit de alto, como marca el protocolo.

- **Receptor (Rx):** Se encarga de recibir los bits tanto de inicio y de paro, como los de datos, así como la manipulación de ellos para ser presentados al usuario una vez la transmisión finalice.

- **Test_bench (Tb):** Es el encargado de manejar los tiempos y la cantidad de las pruebas, asi como generar un estímulo aleatorio para poder ser enviado.

- **Monitor (Mt):** Imprime en la pantalla de comandos de la simulación, valores clave para verificar que el funcionamiento está siendo ejecutado correctamente, y en caso de requerir ajustes de código es la manera visual de detección de errores. Al igual que la presentación de resultados.

- **Interfaz(If):** La interfaz no es en sí un módulo, en una función específica perteneciente a Systemverilog, que nos permite asignar valores a una especie de base de conexión desde cualquier punto de la implementación, permitiéndonos conectar los módulos de manera ordenada y controlada, lo cual simplifica el código utilizado, y la comunicación entre ellos.

- **Top (Top):** El más simple de todos. Es el que conecta a todos los demás módulos, en el código están declaradas las instancias entre los módulos y la interfaz, también aquí está declarada la función CLK que oscila entre 0 y 1 cada 10 ns, dando la función de un clock para llevar a cabo las ejecuciones de datos.

La comunicación serial entre el transmisor y receptor requiere de 4 señales, con 2 conexiones entre ellos:

- **TxD**
- **RxD**
- **Ready**
- **Clr_ready**

El primero para la transmisión de datos, el segundo para la recepción de los mismos, y el tercero y cuarto indican cuando un proceso de envío o recepción se completa.

Los datos son transferidos a una velocidad de transmisión predefinida, en bits por segundo y conocida técnicamente como **baud rate**. Los valores más comunes son 4800, 9600, 56000 y 115200. Además del baud rate, también deben ser predefinidos los siguientes valores dentro del módulo UART:

- **Bit de inicio**
- **Bits de datos**
- **Paridad**
- **Bits de alto**

Es posible enviar cualquier dato en forma binaria de manera serial, siempre que exista un protocolo de transmisión y recepción finamente acoplado y funcionando con exactitud en los tiempos de envío y lectura entre ambos dispositivos.

La transmisión está sincronizada a un bit de inicio, al que le procede la transmisión de datos, finalizando la misma con un bit de alto indicando que la transmisión ha sido completada. El funcionamiento de los módulos los obliga a comunicarse cuando empiezan y terminan eventos, por lo que la conexión **ready** indica al módulo

transmisor cuando empezar a enviar los bits que conforman el protocolo, así como ejecutar el reset para la salida del receptor indicando que los datos han sido entregados con éxito.

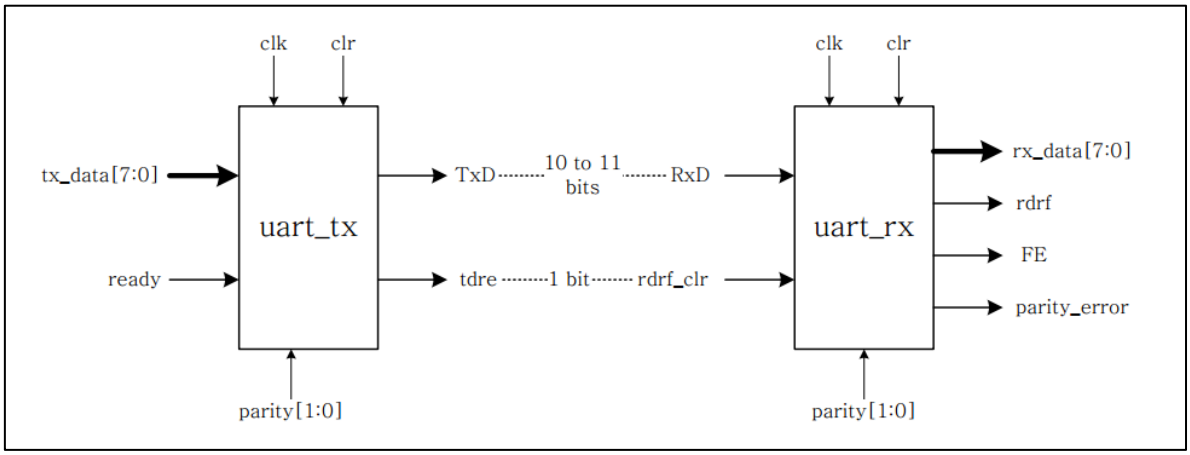


Imagen 01 "Conexiones de los Modulos Transmisor y Receptor"

V. Enfoque de Verificación

Black box:

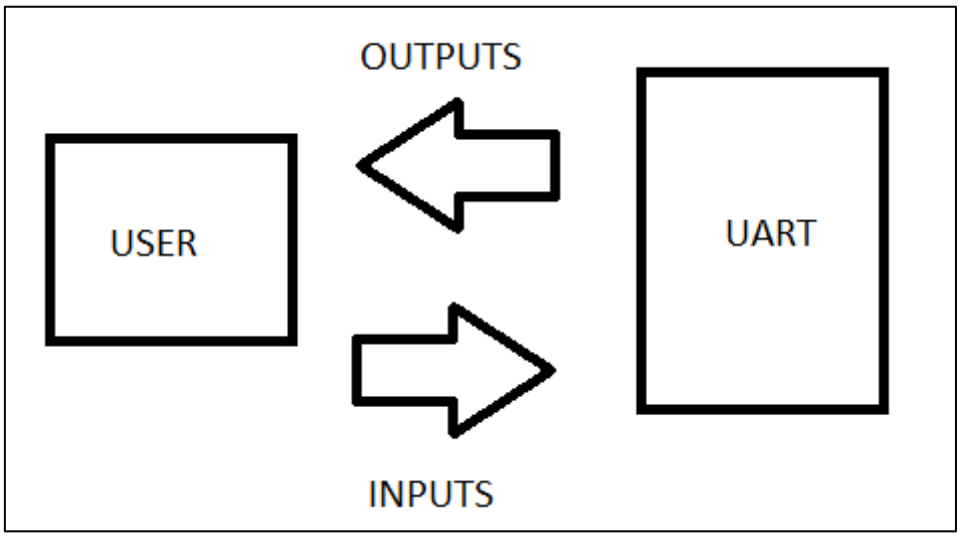


Imagen 02 "Diagrama Caja Negra"

El usuario podrá comprobar que los datos enviados son en realidad los datos que regresan en tiempo real, bit por bit. De manera gráfica también por medio del monitor se presenta los valores en una forma ordenada.

VI. Lenguaje de Programación

La implementación de cada uno de los módulos, así como las tareas y clases, fueron implementadas en Verilog/Systemverilog.

Imagen 03 "Logo SV"

En la industria de los semiconductores y el diseño electronico, este lenguaje es una combinacion entre lenguaje de hardware y lenguaje para verificacion basado en extensiones de Verilog.

El lenguaje ha sido adoptado por IEEE Standard 1800-2005, y ha definido nuevas versiones, la mas actual es 1800-2012.

Existen dos tipos de datos especificacod en SV: static y automatic. Las variables automatic son creadas durante la ejecucion del programa, y las estaticas son creadas al momento del inicio del programa, y mantienen ese mismo valor a menos que se le asigne otro.

El lenguaje maneja tambien nuevos tipos de datos, como lo son legic que es una extension sin las restricciones que tiene reg, es muy utilizado en interfaces. Asi como tambien es capaz de manejar paquetes multidimesionales y diferentes estilos de declaracion de módulos e instancias.

SV tambien contiene un modelo de programacion orientado a objetos, que poveé de una estructura superior y niveles de abstraccion mas altos.

VII. Análisis

Para el éxito de este plan de verificación es necesario comprender que es lo que se busca obtener con las diferentes implementaciones utilizadas.

La transmisión binaria de datos es utilizada en todos los niveles de comunicaciones, y es por excelencia el método más utilizado por cualquier dispositivo electrónico.

La comunicación por palabras, primero es enviada de manera binaria a través de una serie de impulsos llevados a cabo de manera ordenada y codificada, para que una vez recibida, aplicando el

proceso inverso pueda ser desencriptada y de esa manera formar a partir de 1´s y 0´s palabras o letras.

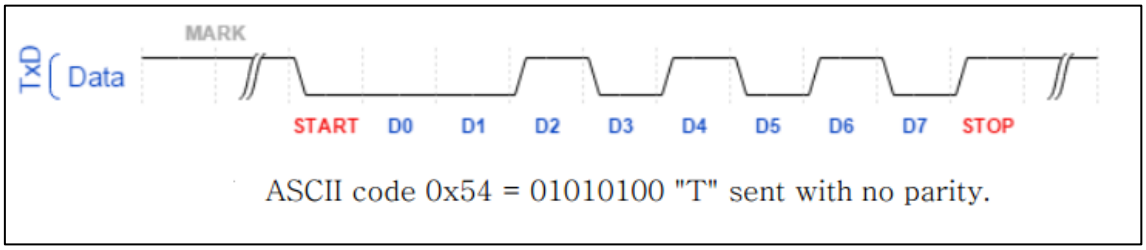


Imagen 04 "Letra T en trnsmision serial binaria"

A partir de eso, establecemos el propósito principal de la UART diseñada, que es enviar un numero binario aleatorio de 8 bits, con la finalidad de recibirlo en tiempo real bit por bit, a través de comunicación de dos módulos. Teniendo también la necesidad de corroborar los resultados, es necesario llevar a cabo la simulación de un programa adaptado para realizar las pruebas, y acotar los resultados.

VIII. Exactitud y Completitud Funcional

Para verificar la funcionalidad, es necesario llevar a cabo una simulación del programa. Para ello utilizamos un Compilador IDE Modelsim, que con su interfaz grafica hace que el análisis funcional pueda ser modelado y se pueda usar para buscar bugs en el código.

Unas de las principales variables a tomar en cuenta son las banderas y las salidas que se comunican solo entre niveles bajos, como lo son las señales que recibe el receptor del transmisor.

Deben de ser cuidadas también las condiciones necesarias para el funcionamiento del módulo, por lo que no será posible orquestar un test con absolutamente todas las variables en aleatorio, de manera que en el diseño del **test_bench** se llevará a cabo un modelo adecuado, partiendo de simulaciones de datos enviadas y registros tomados del tiempo de ejecución del programa, para el envío correcto de un dato aleatorio.

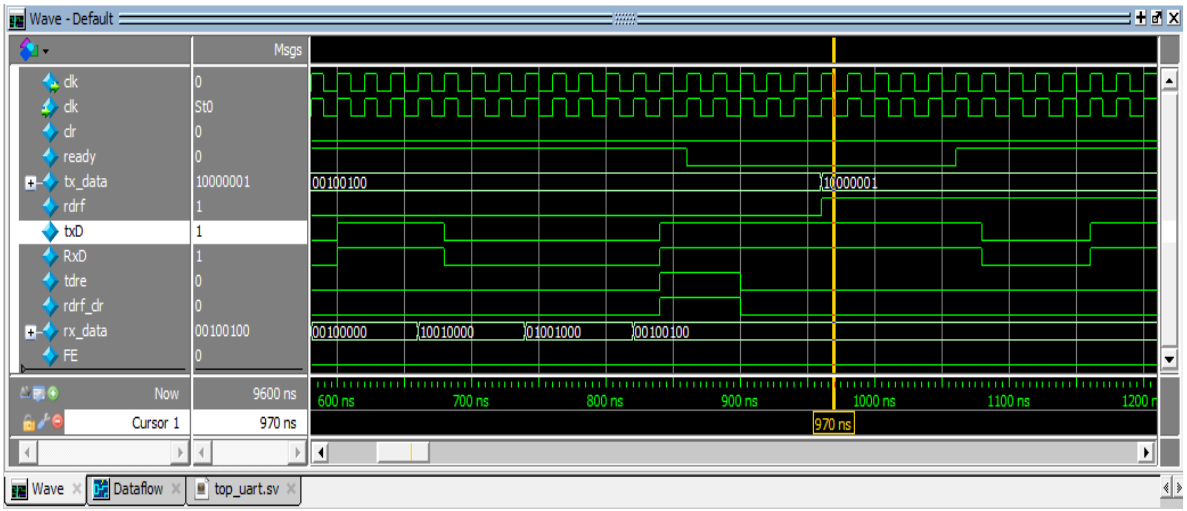


Imagen 05 "Simulacion Tiempo Real transmision"

El test bench diseñado, está preparado para mandar n muestras y está acotada a un valor mayor a 7 y menor a 32000 como valor aleatorio, esto como forma de hacer que los valores tomen combinaciones mas interesantes al momento de ser presentadas.

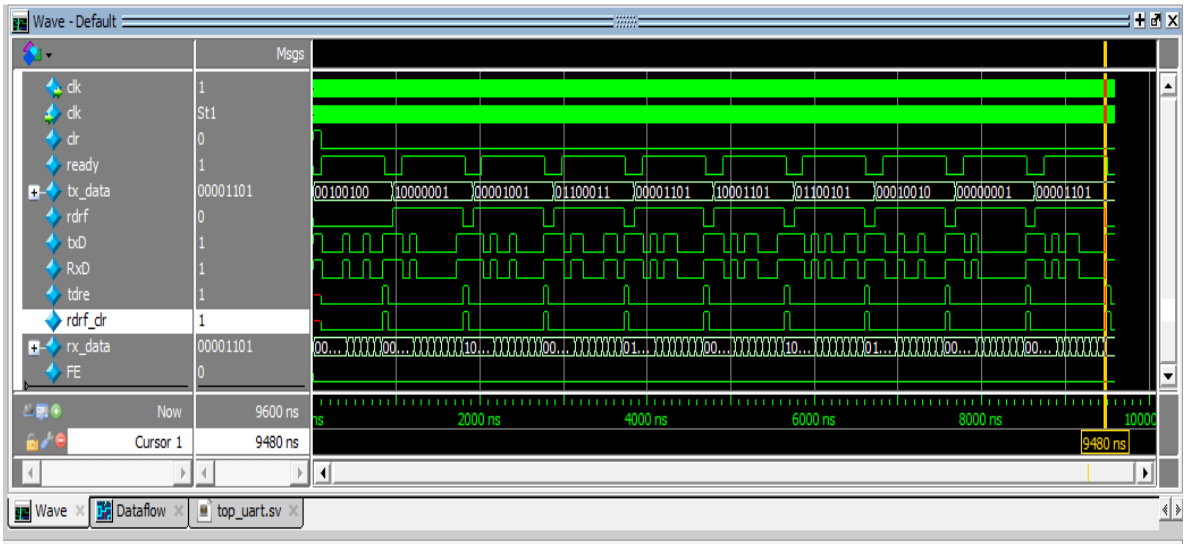


Imagen 06 "10 Transmisiones"

En dichas simulaciones es posible cambiar tanto aspectos graficos, como monitorear el tiempo exacto de la simulacion, para calcular cambios en el codigo. Simultáneamente el **monitor** imprime los valores en la terminal.

IX. Diseño de Verificación

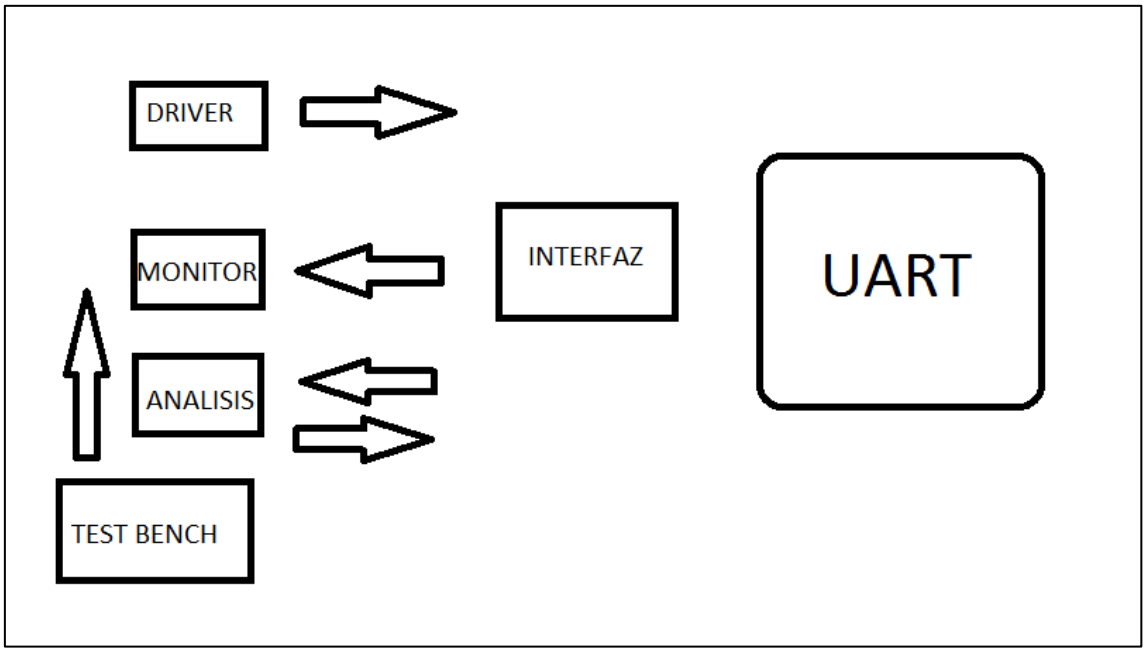


Imagen 07 "Diseño bloques del proyecto"

El componente clave del módulo es la interfaz, puesto que analizando sus propiedades, especialmente la función "modport" nos damos cuenta de lo mucho que puede sintetizar el flujo de información.

Cada módulo se conecta a las variables lógicas de la interfaz, y con ellas trabajan todas sus funciones. Si necesitan un valor, lo toman, si deben escribir un valor, lo actualizan, pero ordenadamente, así las variables fungen como entradas y salidas globales.

```
//-------------------------------------
//---------INTERFAZ--------------------
//-------------------------------------
interface uart_if (input wire clk);
    logic clr;              //Reset
    logic ready;            //Enviar datos
    logic [7:0] tx_data;    //Datos a enviar
    logic txD;              //Salida del Transmisor
    logic tdre;

    logic RxD;              //Aquí se conecta el Transmisor
    logic rdrf_clr;         //Reset envío completo
    logic rdrf;             //Recepción completa
    logic [7:0] rx_data;        //Salida UART
    logic FE;


    modport TX (
        input clk, clr, ready, tx_data,                 //Puertos Transmis
        output txD, tdre);

    modport RX(
        input tdre, txD, clk, clr, RxD, rdrf_clr,       //Puertos de Recep
        output rdrf, rx_data, FE);

    modport TEST(
        input clk,                              //Puertos del Test Bench
        output clr, ready, tx_data);

    modport MONITOR (
        input clk, clr, ready, tx_data, tdre, rx_data, FE);
endinterface: uart_if
```

Imagen 08 "Codigo Interfaz

X. Ejecución

Tras la compilación de los módulos, alojados en el mismo archivo, se precede a crear un nuevo proyecto y se agrega como código principal, para proceder con la simulación.

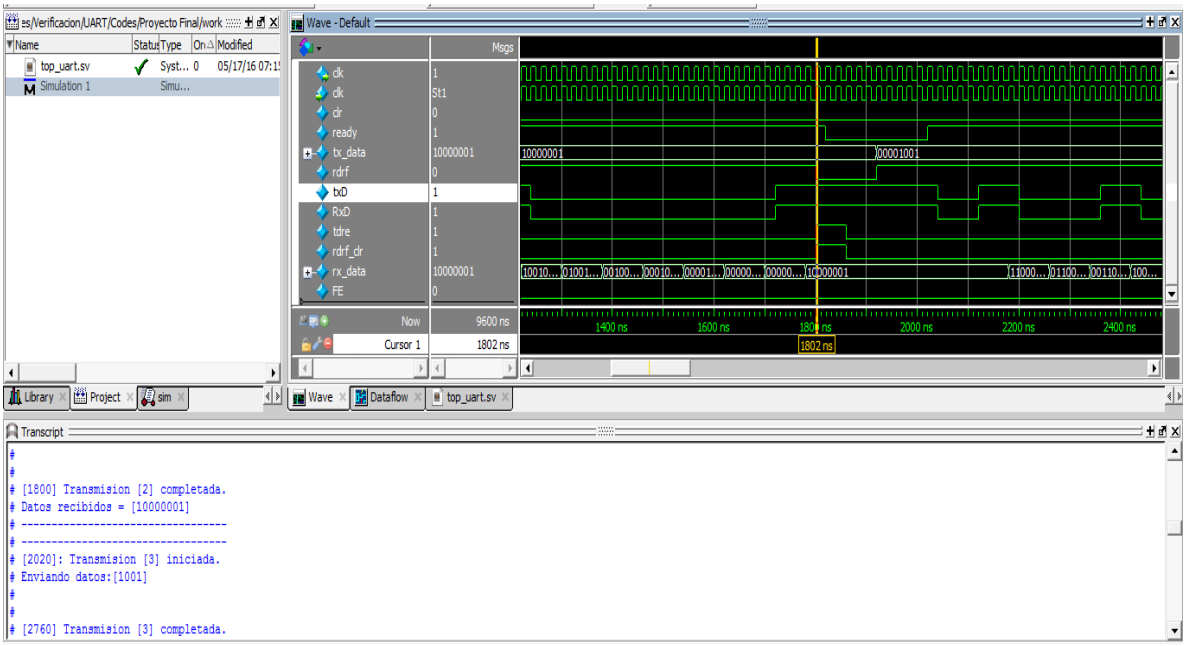


Imagen 09 "Simulacion con Monitoreo manual"

```
# //  ModelSim SE-64 10.1c Jul 28 2012
# //
# //  Copyright 1991-2012 Mentor Graphics Corporation
# //  All Rights Reserved.
# //
# //  THIS WORK CONTAINS TRADE SECRET AND PROPRIETARY INFORMATION
# //  WHICH IS THE PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS
# //  LICENSORS AND IS SUBJECT TO LICENSE TERMS.
# //
# Loading sv_std.std
# Loading work.top_uart(fast)
# Loading work.uart_tx(fast)
# Loading work.uart_rx(fast)
# Loading work.monitor(fast)
# Loading work.test_bench_uart(fast)
# run -all
# [0]: Reset presionado
# ----------------------------------
# [100]: Transmision [1] iniciada.
# Enviando datos:[100100]
#
#
# [840] Transmision [1] completada.
# Datos recibidos = [100100]
# ----------------------------------
# ----------------------------------
# [1060]: Transmision [2] iniciada.
# Enviando datos:[10000001]
#
#
# [1800] Transmision [2] completada.
# Datos recibidos = [10000001]
# ----------------------------------
# ----------------------------------
# [2020]: Transmision [3] iniciada.
# Enviando datos:[1001]
#
#
# [2760] Transmision [3] completada.
# Datos recibidos = [1001]
# ----------------------------------
# ----------------------------------
# [2980]: Transmision [4] iniciada.
# Enviando datos:[1100011]
#
#
# [3720] Transmision [4] completada.
# Datos recibidos = [1100011]
# ----------------------------------
# ----------------------------------
# [3940]: Transmision [5] iniciada.
# Enviando datos:[1101]
#
#
# [4680] Transmision [5] completada.
# Datos recibidos = [1101]
# ----------------------------------
# ----------------------------------
```

Imagen 10 "Simulacion con Monitoreo Automatico"

Se puede corroborar facilmente por cualquiera de las dos simulaciones que el modulo UART funciona correctamente.

XI. Conclusión

Los atributos de la calidad deben ser la corrección, la perfección, la consistencia, la confiabilidad, la utilidad, la eficacia, el apego a los estándares y la eficacia de los costos totales, y de esta manera tener un buen producto como resultado.

Sin duda hace falta a un gran equipo, que no carezca de las habilidades para emprender un proyecto como lo es la verificación y el diseño. La metodología de paralelismo, que le da gran importancia a verificar el funcionamiento del posible diseño, al diseño mismo.

Llevando a cabo la implementación, se presentaron problemas que hicieron cambiar el rumbo de la presentación de datos, utilizando lógica combinatoria y módulos correctamente intercalados en la interfaz para dar con los requerimientos que fueron monitoreo y estímulos.

Por parte del monitoreo, se optó por seguir la lógica compartida de la sincronización que otorgaba el modulo top, pudiendo ejecutar ambos simultáneamente y de manera armónica. Los estímulos por otro lado fueron ejecutados desde el test bench.

Siempre es necesario tomar en cuenta muchas cosas antes del diseño, y tener un plan que agrupe todos los aspectos que deben ser cubiertos, es indispensable.

Aunque el diseño es la parte que encara al usuario, atrás de este se encuentra un programa que reúne todos los objetivos de las actividades de verificación y validación por la cual pasó antes de llegar a ser utilizado. Algunas de estas son valorar y mejorar la calidad de los productos del trabajo generados durante el desarrollo y modificación del software.

XII. Bibliografía

- http://www.testbench.in/TS_23_ONES_COUNTER_EXAMPLE.html
- http://wiki.usgroup.eu/wiki/public/tutorials/svverification
- SystemVerilog.for.Verification.A.Guide.to.Learning.the.Testbench.Language.Features.pdf

XIII. Apéndice

```
//---------------------------------------
//---------TOP UART---------------------
//---------------------------------------
module top_uart(output reg clk);

        uart_if intf(clk);                          //Instancia interfaz
        uart_tx tx1(intf);                          //Instancia transmisor
        uart_rx rx1(intf);                          //Instancia receptor
        monitor mn1(intf);                                  //Instancia de monitor
        test_bench_uart tb1(intf);          //Instancia test bench

        always
        begin
        clk<=1;
        #10;
        clk<=0;  //Generador de reloj
        #10;
        end
endmodule: top_uart

//---------------------------------------
//---------INTERFAZ---------------------
//---------------------------------------
interface uart_if (input wire clk);
        logic clr;                          //Reset
        logic ready;                        //Enviar datos
        logic [7:0] tx_data;        //Datos a enviar
        logic txD;                          //Salida del Transmisor
        logic tdre;

        logic RxD;                          //Aqui se conecta el Transmisor
        logic rdrf_clr;                     //Reset envío completo
        logic rdrf;                         //Recepción completa
        logic [7:0] rx_data;                //Salida UART
        logic FE;


        modport TX (
                input clk, clr, ready, tx_data,                         //Puertos Transmisor
                output txD, tdre);

        modport RX(
                input tdre, txD, clk, clr, RxD, rdrf_clr,           //Puertos de Receptor
                output rdrf, rx_data, FE);

        modport TEST(
                input clk,
        //Puertos del Test Bench
                output clr, ready, tx_data);

        modport MONITOR (
                input clk, clr, ready, tx_data, tdre, rx_data, FE);
endinterface: uart_if

//---------------------------------------
//---------TEST BENCH-------------------
//---------------------------------------
module test_bench_uart(uart_if.TEST intf);
```

```
	initial
	@(posedge intf.clk)
	begin
	intf.clr<=1;
	for (int i = 0; i < 10; i++) begin

		intf.ready<=0;
		intf.tx_data<= $random();
		#100;
		intf.clr<=0;
		intf.ready<=1;
		#760;
		intf.ready<=0;
		#100;
	end

	intf.clr<=1;
	$finish;
	end
endmodule: test_bench_uart

//---------------------------------------
//----------MONITOR--------------------
//---------------------------------------
module monitor (uart_if.MONITOR intf);
	reg [4:0] bandera=0;

	always@(posedge intf.clr or posedge intf.ready or posedge intf.tdre)
	begin
		if(intf.clr)
			$display("[%0t]: Reset presionado",$time);
		else

		if(intf.ready)
		begin
			if(intf.tdre)
			begin
				$display("
");
				$display("[%0t] Transmision [%0d] completada.",$time, bandera);
				$display("Datos recibidos = [%0b]",intf.rx_data);
				$display("-----------------------------------");
				if(intf.FE)
				begin
					$display("[%0t]: |||Error de paridad|||",$time);
				end
			end
			else
			begin
				bandera++;
				$display("-----------------------------------");
				$display("[%0t]: Transmision [%0d] iniciada.",$time, bandera);
				$display("Enviando datos:[%0b]",intf.tx_data);
				$display("
");
			end
		end
	end
endmodule: monitor

//---------------------------------------
```

```
//---------TRANSMISOR-------------------
//--------------------------------------
module uart_tx(uart_if.TX intf);                        //Transmision completa

        reg [2:0] state;                        //Manejador de estados
        reg [7:0] bufftx;                       //Copia del dato a enviar
        reg [15:0] baud_count;                  //Contador Baud Rate
        reg [3:0] bit_count;                    //Bits transmitidos

        parameter bit_time = 3'b010;        //16'h1458;       //Baud Rate, o la tasa de envío
        parameter idle=  3'b000,            //||Estados de la maquina
                          start= 3'b001,            //||Los cuales llevaran procesos
                          delay= 3'b010,            //||En las diferentes etapas
                          shift= 3'b011,
                          stop=  3'b100;

        always @(posedge intf.clk or posedge intf.clr)
        begin
        if(intf.clr)
        begin
                state <= idle;                  //"Espera"
                bufftx <= 0;                    //Copia del Dato a 0
                baud_count <= 0;                //Contador de tasa de envio a 0
                bit_count <= 0;                 //Bits enviados a 0
                intf.txD <= 1;                  //Salida a 1
        end

        else
        begin

                case (state)
                        idle:                                           //Estado principal Espera
                        begin
                                bit_count <= 0;                 //Bits enviados a 0

                                intf.tdre  <= 0;                        //Trans. Completa a 0

                                if (~intf.ready)                //Si "ready" no es presionado
                                begin
                                        state <= idle;          //Cicla el mismo estado en la
maquina
                                end

                                else                                    //Si "ready" es
presionado
                                begin
                                        bufftx <= intf.tx_data; //Copia el dato al buffer
                                        baud_count <= 0;        //Contador de tasa de envío a 0
                                        state <= start;         //Cambia de estado
                                end
                        end

                        start:                                          //Segundo estado
Preparacion
                        begin
                                baud_count <= 0;                //Se mantiene el contador de tasa
en 0
                                intf.txD <= 0;                          //Salida cambia a 1,
indicando el envío
                                intf.tdre <= 0;                         //Envío completo se
mantiene en 0
                                state <= delay;                 //Cambia el estado
```

```
                        end

                        delay:
        ///Tercer estado para enviar el dato

                        begin
        ///en la tasa de envío seleccionada
                                intf.tdre <= 0;
        //Envío completo en 0

                                if (baud_count >= bit_time)                     ///Si es igual o
mayor el contador de tasa
                                begin
        ///al de bits enviados
                                        baud_count <= 0;
        //Contador de tasa a 0

                                        if(bit_count < 8)                       //Si no se han
enviado los 8 bits
                                                state <= shift;
        //Cambia de estado de recorrimiento
                                        else
        //Si se enviaron 8 bits
                                                state <= stop;
        //Cambia estado final
                                end

                                else
        ///Si el contador de tasa es menor o igual al
                                begin
        ///estipulado de Baud Rate
                                        baud_count <= baud_count +1;    //Incrementa    +1    el
contador de tasa
                                        state <= delay;                         //Cicla
el mismo estado
                                end
                        end

                        shift:
        //Cuarto estado Recorrimiento de bits
                        begin

                                intf.tdre <= 0;
        //Transmision completa en 0
                                intf.txD <= bufftx[0];                  //Salida    es    el
LSB de la copia del dato a enviar
                                bufftx[6:0] <= bufftx[7:1];             //Recorre el vector, para
ir sacando los LSB's
                                bit_count <= bit_count +1;              //Aumenta cuando envía
un bit
                                state <= delay;                         //Regresa       al
Tercer estado
                        end

                        stop:
        //Estado final
                        begin
                                intf.tdre <= 1;
        //Transmision completa a 1
                                intf.txD <= 1;                                  //Salida
se queda en 1

                                if(baud_count >= bit_time)              ///Si el contador de tasa,
es mayor
                                begin                                           //o
igual al Baud rate elegido
                                        baud_count <= 0;                //Contador      de
tasa a 0
```

```
                                        state <= idle;                          //Estado a
Primer estado Espera
                                end

                                else                                                    //Si el
contador es menor al parametro
                                begin
                                        baud_count <= baud_count+1;     //Incrementa el contador
                                        state <= stop;                          //Cicla el mismo
estado final
                                end
                        end
                endcase
         end
        end
endmodule

//--------------------------------------
//---------RECEPTOR---------------------
//--------------------------------------

module uart_rx(uart_if.RX intf);                                //Bandera paridad

        reg [2:0] state;                        //Manejador para la FSM
        reg [7:0] rxbuff;                       //Copia del dato
        reg [11:0] baud_count;                  //Contador para tasa de envío
        reg [3:0] bit_count;                    //Contador de Bits recibidos


        parameter mark = 3'b000,                //Valores que tomará la FSM
                start= 3'b001,
                delay= 3'b010,
                shift= 3'b011,
                stop = 3'b100;

        parameter bit_time= 3'b010;             //12'hA28;              //Tasa de envío de cada
bit 9600 baud= 12'hA28
        parameter half_bit_time =3'b001;  //12'h514;  //Mitad de tasa de envío, para sincronización

        assign intf.rx_data = rxbuff;                   //Copia del valor a la salida
        assign intf.RxD = intf.txD;                     //Copia de la interfaz la salida del TX
        assign intf.rdrf_clr = intf.tdre;               //copia el valor de la interfaz


        always @(posedge intf.clk or posedge intf.clr or posedge intf.rdrf_clr)
        begin
        if(intf.clr)                                                    //Reset en 1
        begin
                state <= mark;                          //Selecciona el primer estado
                rxbuff <= 0;                            //Copia en 0
                baud_count <= 0;                        //Contador de tasa en 0
                bit_count <= 0;                         //Bits recibidos en 0
                intf.rdrf <= 0;                                 //Recepcion completa en 0
                intf.FE <= 0;                                   //Error de paridad en 0
        end

        else                                                    //Reset desactivado
        begin
                if(intf.rdrf_clr)                       //||Recepcion completa desactivada a
menos
                        intf.rdrf <= 0;                         //||que el transmisor indique
envio
```

```
        else
                case(state)
                        mark:                                                   //Primer estado,
espera el primer bit
                                begin
                                        bit_count <= 0;                         //Bits enviados a
0
                                        baud_count <= 0;                        //Contador de
tasa de envio a 0
                                if(intf.RxD)                                    //Si la entrada
está en 1
                                        state <= mark;                  //Permanece en el estado
                                else                                            //Si la entrada está en 0
                                        begin
                                        intf.FE <= 0;                   //Error de paridad a 0
                                        state <= start;                 //Cambio de estado
                                end

                        end

                        start:                                                  //Segundo
estado
                                if(baud_count >= half_bit_time)    //Si el contador de tasa llega a
medio bit transmitido
                                begin
                                        baud_count <= 0;                        //Reiniciar el
contador
                        state <= delay;                                 //Cambiar de estado
                                end

                                else                                                    //Si aun
no llega a medio bit
                                begin
                                        baud_count <= baud_count+1;     //Aumentar el contador
+1
                        state <= start;                                 //Permanecer en el estado
                                end

                                delay:                                                  //Tercer
estado
                        if(baud_count >= bit_time)              //Si el contador es igua o mayor al
parametro de BR
                                begin
                                        baud_count <= 0;                        //Contador a 0
                                        if(bit_count < 8)                       //Si no ha enviado los 8
bits
                                                state <= shift;                 //Cambia al 4
estado
                                        else                                            //Si ya
los envió
                                state <= stop;                  //Cambia al ultimo estado
                                end

                                else                                                    //Si el
contador es menor al parametro BR
                                begin
                                        baud_count <= baud_count+1;     //Aumenta el contador
                        state <= delay;                                 //Cicla el mismo estado
                        end


                                shift:
        //Cuarto estado, recorrimiento
                                begin
```

```
                        rxbuff[7] <= intf.RxD;                  //Mete en el MSB el valor
del bit transmitido
                rxbuff[6:0] <= rxbuff[7:1];          //Recorre el vector
                        bit_count <= bit_count + 1; //Aumenta el contador de bits
                        state <= delay;                         //Regresa al tercer estado
                        end

                        stop:
        //Ultimo estado
                        begin
                                intf.rdrf <= 1;
        //Recepcion completa
                if(~intf.RxD)                                   //Si la entrada desde el
transmisor es 0
                begin
                                        intf.FE <= 1;
                                        state<= stop;           //Hay error de
paridad
                                end
                                else                                    //S es 1
                                        begin
                                        intf.FE <= 0;                   //No
hay error
                                        state <= mark;          //Regresa al
primer estado
                        end
                        end
                                endcase
                end
  end
endmodule: uart_rx
```