**Plan de verificación: “Controlador de memoria SDRAM”**

**Autores:**

Daniel Martínez Alvarado

Ronny Zárate Ferreto

**12/December/2021**

**Revisiones**

| **Número de Revisión** | **Fecha de revisión** | **Descripción** | **Autor** |
| --- | --- | --- | --- |
| 0.1 | 12/Nov/2021 | Primera iteración | R.Zárate / D. Martínez |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[**Estrategia de verificación:**](#_qlcr2zj4njnb) **4**

[**Niveles de verificación**](#_hs9ye2at9j6m) **4**

[**DUV a verificar**](#_a72i05cz4mpd) **4**

[**Ambiente verificación:**](#_fs8t9dih8bkt) **6**

[Metodología](#_hznwwc3lhnyw) 6

[DUV: Controlador de memoria SDRAM](#_yw7z8rfpivmf) 10

[**Alcance del plan de verificación**](#_nbyum08dnpv1) **11**

[**A nivel de arquitectura**](#_itsbcn3v84vh) **11**

[**Requerimientos de la verificación.**](#_dwg42f6c0488) **11**

[**Herramientas requeridas.**](#_ppop726c3xeb) **12**

[**Riesgos y dependencias.**](#_t6wdgnuikcqb) **13**

[**Requerimientos de recursos.**](#_2v8lh4ufhd8w) **13**

[**Calendarización.**](#_m28x29j8v58p) **13**

[**Arquitectura:**](#_rv67pcbqwzkc) **13**

[**13. Configuraciones**](#_c0g639i5125u) **19**

[**14. Casos de uso:**](#_j8qrxhxprz3j) **19**

[**Nivel de aplicación.**](#_c46ogff79dr2) **19**

[**Nivel de interfaz.**](#_7viirv5vj42v) **19**

[**Protocolos y estándares.**](#_9fb69tnw9amm) **19**

[**Rendimiento.**](#_lcrnjfo5irm0) **19**

[**15. Interrupciones:**](#_gx19grypewps) **20**

[**16. Manejo de Errores:**](#_ftgtl3q32rj0) **20**

[**17. Funciones**](#_g3h961lbds0j) **20**

# Estrategia de verificación:

A nivel de estrategia de verificación se procede a incluir un módulo externo al controlador de memoria, basado en JTAG, “Boundary Scan” y por último un MBIST para la comprobación de la memoria.

Al utilizar JTAG, se dispone de un controlador TAP, el cual permite acceder y ejecutar las diferentes instrucciones que permiten propiamente acceder a los pines del controlador y así insertar comandos que permitan que permitan probar el mismo, esta prueba se realiza específicamente utilizando el “boundary scan”.

Por último, se implementa un módulo el MBIST, el cual realiza diferentes pruebas a nivel de los FIFOS asincrónicos utilizados en el controlador de memoria, el mismo tiene instrucciones de para probar todos los bits en 0 o 1, así como el llamado “checkboard”

# Niveles de verificación

## DUV a verificar

El DUV a verificar consiste en dos módulos o capas: (1) el adaptador de memoria o el controlador de memoria en sí, y (2) el adaptador de Wishbone.

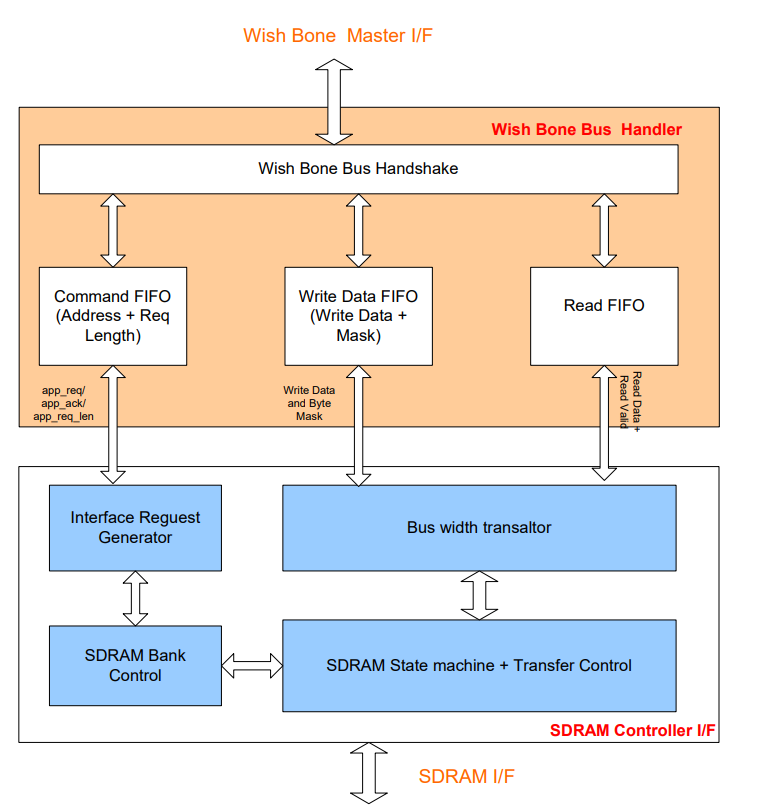
****

Figura 1 - DUV a verificar.

* 1. **Descripción de la jerarquía de verificación**

Para efectos de verificación, se seguirá una verificación de unidad, dado que el DUV a verificar es una unidad que generalmente se localiza en algún subsistema y se comunica a nivel de bus. Por ende, se debe seguir la verificación del módulo de SDRAM como una unidad. Asimismo, no es posible continuar a más bajo nivel por falta de especificaciones.

A nivel de Jerarquía de verificación, como se observa en la figura 2, se denota que el módulo JTAG es el controlador principal, así como que controla el “Boundary Scan”, y por último dentro del controlador de memoria existe el módulo MBIST.

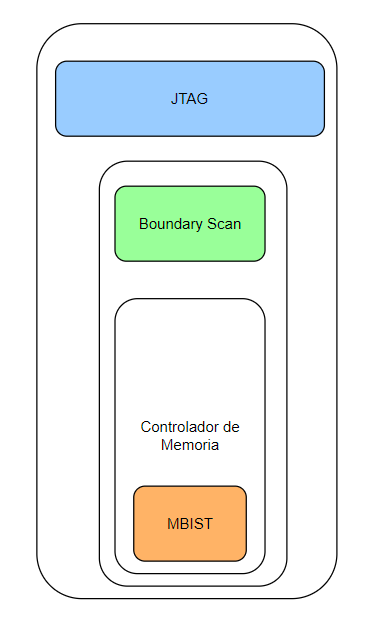


Figura 2. Jerarquía de Verificación

# Ambiente verificación:

Para generar un ambiente de verificación, primero se genera un controlador JTAG basado en IEEE 1149.1, donde el mismo puede ser probado por medio de herramientas de Synopsis tales como; “Design Compiler”, “Verdi” y “VCS”. Esto utilizando los ordenadores de CentOS proveídos por el Instituto Tecnológico de Costa Rica.

## Metodología

Para el ambiente de verificación, se utilizará un módulo JTAG o “**Joint Test Action Group”,** el mismo consiste en :

* **Controlador TAP**: Este módulo dirige las acciones necesarias para ejecutar en el registro de instrucciones, esto por medio de datos serializados en los pines:
  + **TDI** (Test Data Input)
  + **TDO** (Test Data Output)
  + **TCK** (Test Clock)
  + **TMS** (Test Mode)
  + **TRST**( Test Reset) \*Opcional

El controlador, es una máquina de estados, la cual dependiendo de por ejemplo el pin TMS, se pasa de un modo a otro. Las instrucciones mandatorias son:

* **EXTEST,** Se procede a generar una prueba por medio del “Boundary Scan” insertado en el DUV
* **SAMPLE**, Permite tomar pruebas o bien copias de las salidas y entradas del DUV.
* **PRELOAD**, Permite ingresar un patrón de prueba sin interferir con el DUV.

Por otro lado, también existen instrucciones opcionales:

* **IDCODE**, Salida de un código de identificación almacenado en un registro de datos de identificación
* **CLAMP** y **HIGHZ**, Este comando ayuda a crear un insolación entre el módulo JTAG y el DUV, ya que coloca los pines en modo de alta impedancia.
* **IC\_RESET,** Permite realizar diferentes reinicios al JTAG o DUV
* **RUNBIST,** Correr la comprobación de algún módulo BIST

La máquina de estados de este módulo se comporta como la figura 3, donde se observa que el mismo consisten 16 estados diferentes, en donde toda la instrucción se recibe en el TDI, y esto por medio de corrimientos ingresa al registro de instrucciones. Luego la instrucción es decodificada, para poder ser ejecutada. De igual manera en por medio del TDI, se puede agregar el patrón de prueba. Cuando se ejecuta la prueba los resultados son guardados en el registro de datos y expuestos en el pin TDO.

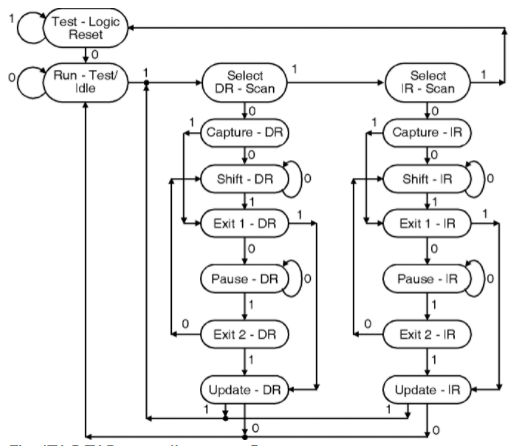


Figura 3. Máquina de estados JTAG

Por otro lado, en el módulo de MBIST, este va dirigido a los FIFOS asincrónicos, los cuales tienen la tarea de manejar una cola de lecturas o escrituras cuando se realiza en “burst”, lo cual permite que en solo una instrucción enviar una cantidad específica de lecturas. Para eso el MBIST consiste en:

* **Controlador BIST**, este módulo es el que mueve los estados dentro de cada prueba, por ejemplo, si se está haciendo una lectura y ya terminó, es necesario activar la comparación.
* **Módulo de escritura**, este módulo se encarga de colocar los parámetros básicos para genera una escritura en el FIFO asincrónico
* **Módulo de Lectura**, este módulo se encarga de colocar los parámetros básicos para genera una lectura en el FIFO asincrónico
* **Comparador**, se encarga de determinar si lo que se está leyendo en el FIFO concuerda con el patrón de la prueba.
* **Data**, Guarda la dirección y el patrón cuando el comparador marca un fallo.

El MBIST también cuenta con instrucciones de para que el mismo se configurado estas son:

Para modificar el modo:

* **IDLE**, se desactiva el modo bist, por lo que el FIFO asincrónico puede funcionar normalmente
* **AUTOCHECK**, se prueba de forma automática el FIFO asincrónico, en este caso se recorre toda la memoria.
* **DEBUG**, se prueba de forma automática el FIFO asincrónico, en este caso se recorre toda la memoria, pero se guarda el dato en la posición de memoria especificada en la dirección.

Por último, el MBIST cuenta con diferentes patrones para realizar pruebas, estos son:

* **Todo en 0**, Se prueba que toda o solo una posición de memoria sea 0
* **Todo en 1**, Se prueba que toda o solo una posición de memoria sea 1
* **Checkboard**, Se prueba que toda o solo una posición de memoria tenga un patron intercalado de 0 y .
* **LSFR**, Se utiliza un generador pseudo-random con el fin de ingresar patrones aleatorios a partes o a toda la memoria.

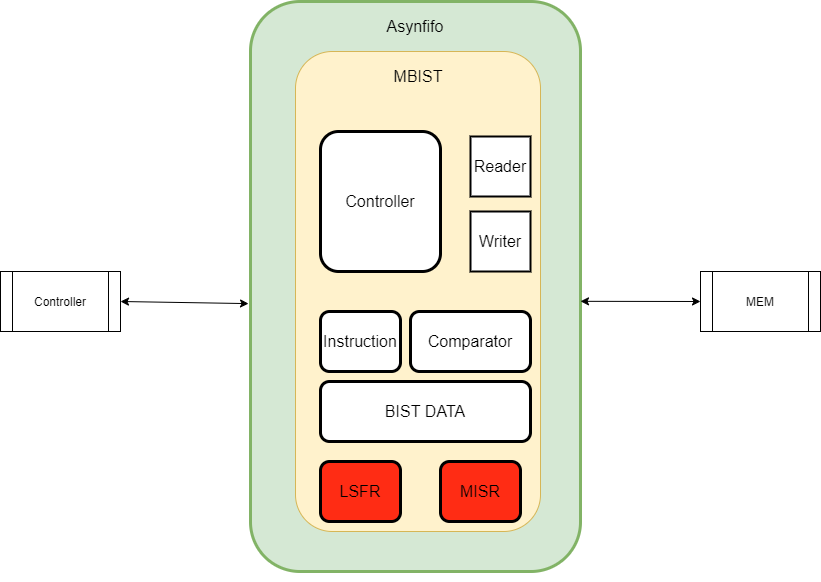


Figura 4. MBIST

## DUV: Controlador de memoria SDRAM

El DUV se conecta a la interfaz en el top. En este caso, el DUV viene empaquetado como módulos de Verilog. El controlador de SDRAM posee dos módulos principales, tal como se describe en la Figura 1: (1) adaptador de Wishbone, (2) adaptador de memoria.

Ante ello, la conexión con el módulo involucra tres partes: (1) el bus de aplicación, que involucra la abstracción introducida por el adaptador para transacciones, (2) el bus de configuración, que ayuda a modificar el comportamiento del adaptador en cuanto a tiempos, máscaras, entre otros, y (3) la interfaz de la SDRAM. Ante ello, la abstracción del control se realiza a **nivel de bits**.

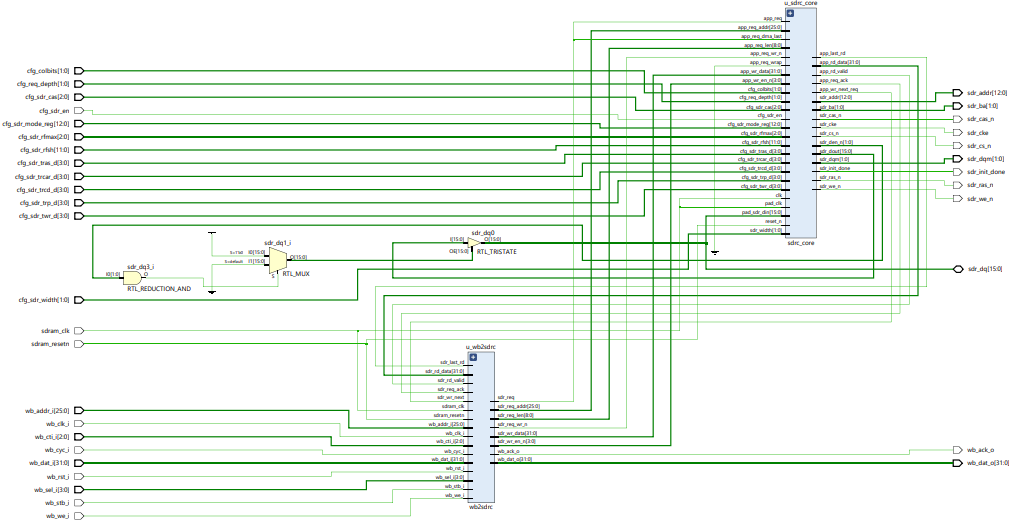


Figura 5. Controlador de Memoria

# Alcance del plan de verificación

## A nivel de arquitectura

**JTAG**

Las instrucciones disponibles son:

* BYPASS
* EXTEST
* SAMPLE\_PRELOAD
* INTEST
* RUNBIST
* IDCODE

**MBIST**

El MBIST solo funcionara para los FIFOS asincrónicos y el mismo cuenta con los patrones de:

* Todo en 0
* Todo en 1
* Checkboard
* LSFR

**Boundary Scan**

Se agrega boundary scan a todos los pines disponibles en el DUV

# Requerimientos de la verificación.

**Metas del proyecto.**

Para el proyecto, es necesario agregarle el DUV

* Controlador JTAG
* Boundary Scan
* MBIST
* DFX Extra

Por otro lado, es necesario desarrollar los siguientes elementos:

* Lista de comandos o archivo TCL para insertar scan en DC compiler.
* Netlist resultante de la inserción de scan.
* Archivos de verilog con el RTL modificado y el banco de pruebas.
* Documentación de la arquitectura, microarquitectura y plan de verificación de DFX incluyendo diagramas, registros, conexiones, procedimientos, pruebas, limitaciones, etc

Similar a la figura 6.

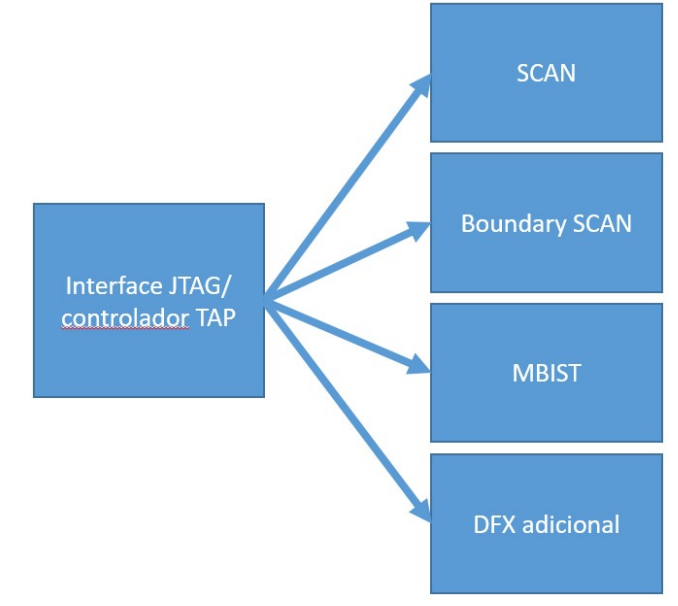


Figura 5. Requisitos del proyecto

# Herramientas requeridas.

Para este proyecto, se utilizará Synopsys para el diseño y verificación.

El lenguaje a utilizar es SystemVerilog y verilog.

# Riesgos y dependencias.

No hay

# Requerimientos de recursos.

A nivel de recursos se solicita dos ingenieros experimentados con énfasis en Sistemas Electrónicos, con conocimiento en Verilog, así como de diferentes alcances de “Design for testability”

# Calendarización.

No aplica.

# Arquitectura:

**SCAN**

**Procedimiento**

Se utilizan celdas para reemplazar todos los registros del DUV con el fin de obtener cadenas de SCAN que nos permiten ingresar información en los registros y así en los diferentes módulos.

Para generar esto es necesario utilizar los siguientes comandos en “Synopsis Design Compiler”

set target\_library lsi\_10k.db

set link\_library [list \* lsi\_10k.db]

set test\_default\_period 10

set test\_default\_bidir\_delay 0

set test\_default\_delay 0

set test\_default\_scan\_style "multiplexed\_flip\_flop"

set test\_default\_strobe 2

analyze -format verilog -lib WORk {../rtl\_bus/async\_fifo.v ../rtl\_bus/sdrc\_bank\_ctl.v ../rtl\_bus/sdrc\_bank\_fsm.v ../rtl\_bus/sdrc\_bs\_convert.v ../rtl\_bus/sdrc\_core.v ../rtl\_bus/sdrc\_define.v ../rtl\_bus/sdrc\_req\_gen.v ../rtl\_bus/sdrc\_top.v ../rtl\_bus/sdrc\_xfr\_ctl.v ../rtl\_bus/sync\_fifo.v ../rtl\_bus/wb2sdrc.v}

elaborate sdrc\_top -lib WORK -update

link

uniquify

compile -scan

set\_dft\_signal -view existing\_dft -type ScanClock -port sdram\_clk -timing {4.0 5.0}

set\_dft\_signal -view existing\_dft -type ScanClock -port wb\_clk\_i -timing {4.0 5.0}

set\_dft\_signal -view existing\_dft -type Reset -active 1 -port sdram\_resetn

set\_dft\_signal -view existing\_dft -type Reset -active 1 -port wb\_rst\_i

set\_dft\_signal -type TestData -port sdram\_clk

set\_dft\_signal -type TestData -port wb\_clk\_i

set\_dft\_signal -type TestData -port wb\_rst\_i

set\_dft\_signal -type TestData -port sdram\_resetn

create\_test\_protocol -infer\_clock -capture\_procedure multi\_clock

dft\_drc

set\_scan\_configuration -chain\_count 50

preview\_dft

insert\_dft

dft\_drc

change\_names -rules verilog -verbose -hierarchy

sh mkdir src

write -format verilog -hierarchy -out ./src/sdrc\_top\_struct.v

set test\_stil\_netlist\_format verilog

sh mkdir reports

write\_test\_protocol -output reports/protocol.spf

Estos comandos nos van a generar un total de 26 cadenas de SCAN.

**Tamaño en el diseño**

Agregar las celdas de Scan agrega un total de 2746 de área

**Tiempo estimado de ejecución de pruebas**

Cada cadena tiene un total aproximado de 48 registros, por lo que necesitaría un total de 48 clocks para generar el corrimiento completo de un dato

**Boundary Scan**

**Procedimiento**

Para agregar boundary scan a cada pin del DUV, es necesario utilizar una celda similar al de la figura 6, en donde este puede ser controlado por medio del controlador TAP del JTAG. Al colocar estos elementos en los pines es necesario agregar un bus de 253 bits con el fin de cubrir todos los elementos agregados e ingresar un patrón de pruebas.

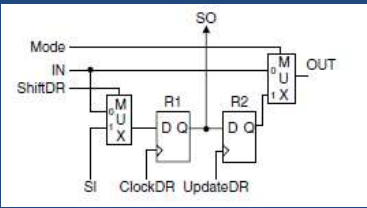


Figura 6. Boundary Scan Cell

**JTAG**

**Procedimiento**

Se genera la circuitería para implementar el boundary-scan, que se divide en 4 componentes principales:

* **TAP**, Se agregan las cuatro entradas obligatorias (TDI, TDO, TMS, TCK).
  + TCK: Es la entrada de reloj para sincronizar las operaciones de prueba entre los diferentes componentes del DUV.
  + TDI: Es una entrada que permite que las instrucciones de prueba y los datos de prueba se carguen en serie en el registro de instrucciones y en los diversos registros de datos de prueba, respectivamente.
  + TDO: Es la salida de datos que permite su extracción serial.
  + TMS: Es la única entrada de control de prueba para el controlador TAP. Todas las operaciones de prueba de boundary-scan, como el cambio, la captura y la actualización de datos de prueba, están controladas por la secuencia de prueba aplicada a esta entrada.
* **Controlador TAP,** Es una fsm de 16 estados que controla cada paso de las operaciones del boundary-scan. Los cuales se presentan a continuación:
  + Test–Logic–Reset: En este estado, el boundary-scan está desactivado y el sistema funciona en su modo normal.
  + Run-Test/Idle: En este estado, el circuito de boundary-scan está esperando algunas operaciones de prueba sincronizadas con el TCK.
  + Select-DR-Scan: Este es un estado temporal en preparación para ingresar datos al registro de datos
  + Capture-DR: En este estado, los datos se pueden cargar en paralelo a los registros de datos seleccionados por la instrucción actual.
  + Shift-DR: En este estado, los datos de prueba se escanean en serie a través de los registros de datos seleccionados por la instrucción actual.
  + Exit-DR: Este también es un estado temporal. Todos los datos cargados en paralelo (desde el estado Capture-DR) o desplazados (desde el estado Shift-DR) se mantienen en el registro de datos seleccionado en este estado en preparación para ingresar al estado de actualización o pausa.
  + Pause-DR: El boundary-scan detiene su función aquí para esperar algunas operaciones externas.
  + Exit2-DR: Este estado indica la finalización de la operación de captura / cambio y permite que el controlador TAP ingrese al estado de actualización
  + Update-DR: En este estado, los datos se bloquean en la salida paralela de los registros de datos de prueba seleccionados desde la ruta del registro de desplazamiento en el borde descendente del TCK
* **Registro de instrucciones**, se utiliza para almacenar la instrucción que se ejecutará. Según el estándar, este registro debe tener un diseño de dos etapas, de modo que cuando se cambie una nueva instrucción en la instrucción actual se pueda bloquear en la salida paralela del IR para evitar la posibilidad de tener un estado indeterminado en la salida de IR. Se implementaron las siguientes instrucciones:
  + **BYPASS**
  + **EXTEST**
  + **SAMPLE\_PRELOAD**
  + **INTEST**
  + **RUNBIST**
  + **IDCODE**

**Tamaño de diseño**

El controlador TAP tiene un área de 1731

**Limitaciones**

* Limitaciones para implementar un testeo serial de las cadenas de scan completas.

**Configuración**

Para configurar el controlador TAP, es necesario ingresar de manera serial mediante el TMS los datos para el uso de la máquina de estados.

Los datos que se desean ingresar son serializados mediante el TDI y también por desplazamiento es posible obtener los anteriores en el TDO.

**Diagrama**

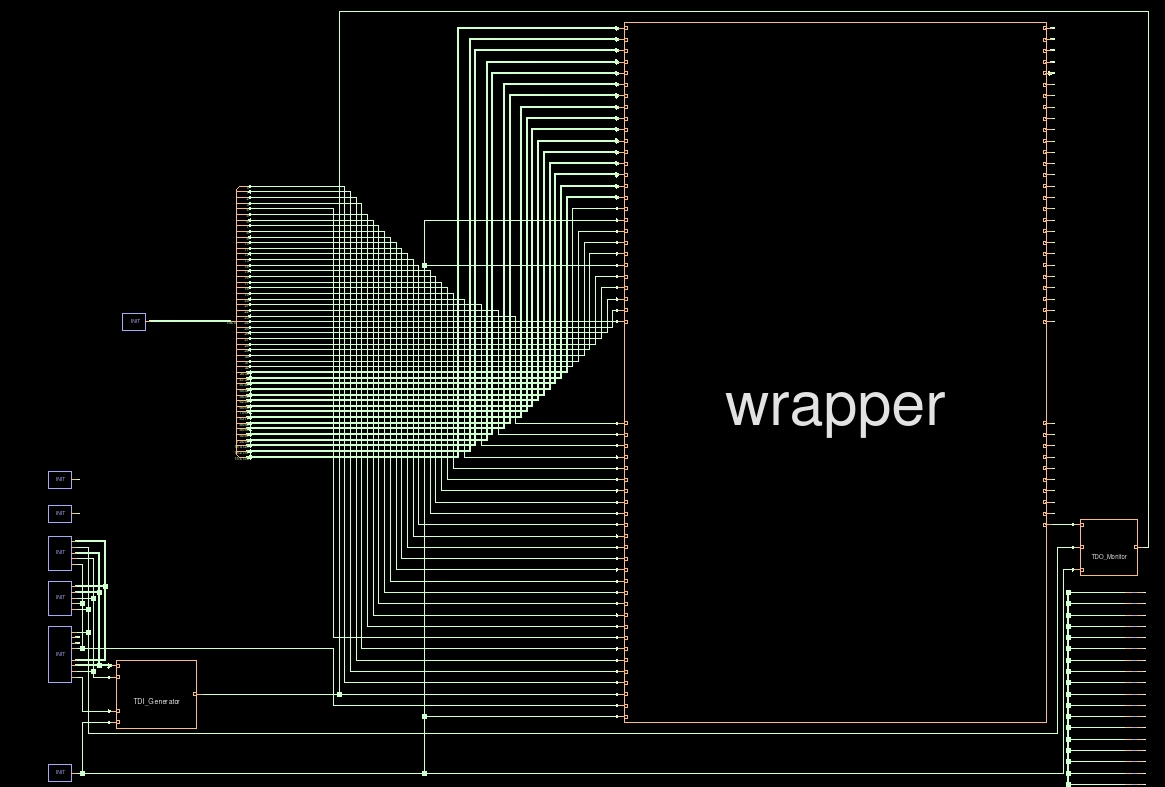


Figura 7 Diagrama de conexiones

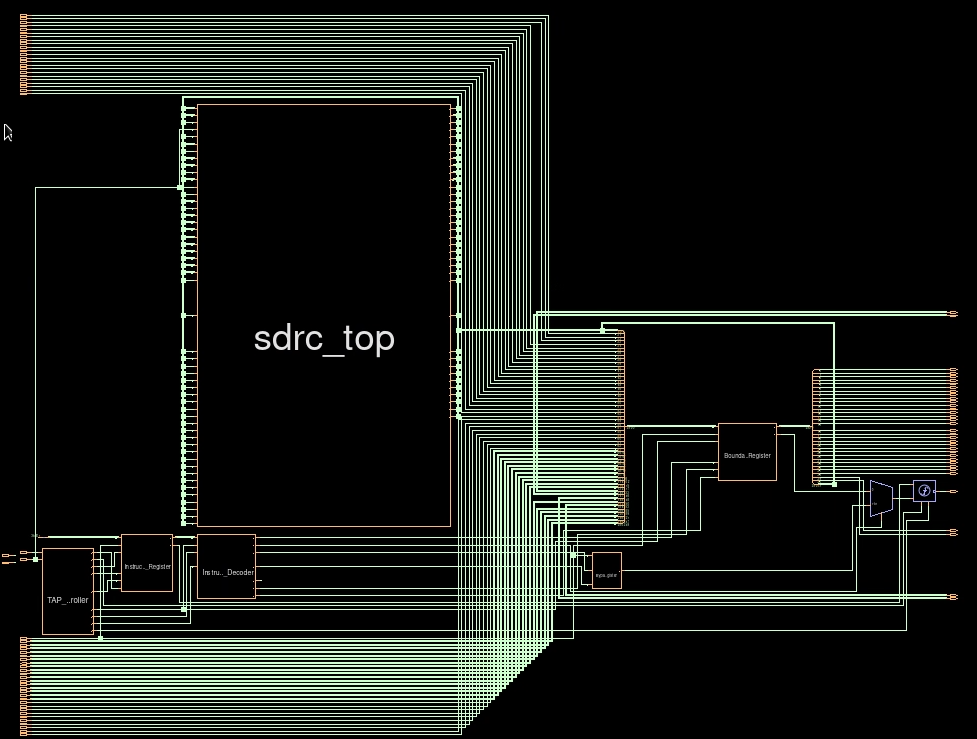
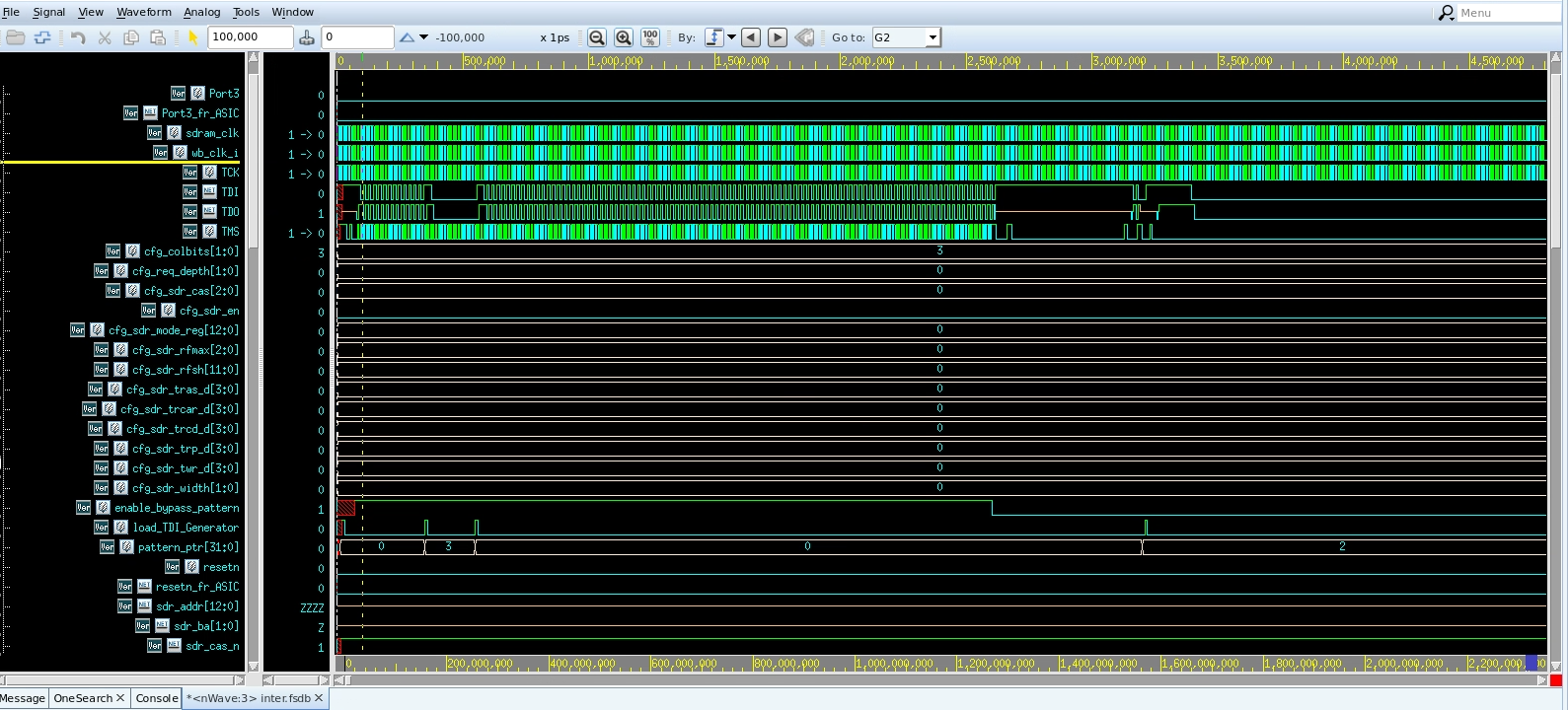


Figura 7 Diagrama de conexiones

**Testbench**:

A nivel de banco de pruebas se corre un reset y luego se hacen 48 desplazamientos para llenar la profundidad de la cadena de scan.

**Simulación:**

****

**MBIST**

**Procedimiento**

Se genera un controlador MBIST, el cual consiste en:

* **TOP**, Controla todos los eventos que pasan en el controlador
* **Writer**, Ejecuta una escritura en el FIFO. El mismo al tener una forma específica de funcionar, la cual sigue la siguiente forma
  + Reinicio de FIFO en alto (wr\_rst = 1)
  + Habilitar escritura FIFO (wr\_en = 1)
  + Colocar el dato de escritura (wr\_data = XX)
  + Deshabilitar escritura(wr\_en=0)

El FIFO al no tener entradas de punteros a la memoria, es necesario ejecutar la lista anterior la cantidad de profundidad específica del FIFO, en este caso 8 veces. Esto va a provocar que el contador interno el FIFO asincrónico, genere el puntero ascendentemente.

* **Reader,** Ejecuta una lectura en el FIFO. El mismo al tener una forma específica de funcionar, la cual sigue la siguiente forma
  + Reinicio de FIFO en alto (rd\_rst = 1)
  + Habilitar escritura FIFO (rd\_en = 1)
  + Colocar el dato de escritura (rd\_data = XX)
  + Deshabilitar escritura(rd\_en=0)

Este módulo se habilita luego de la escritura, esto ya que si se escribe más de 4 veces de manera secuencial el FIFO se llena y nos genera errores de overflow, por lo que es necesario estar liberado el FIFO por medio de estas lecturas. De igual forma, se generan tantas lecturas como escrituras se generaron

* **Comparador,** Ejecuta las comparaciones para cada patrón específico, por ejemplo si coloca el patrón de todo en 0, el comparador crea una registro de prueba el cual tiene el patrón deseado e indica si hay problemas en caso de que no exista una igualdad en el dato de lectura.
* **Data**, Registra la dirección y el patrón que falló por medio de las banderas generadas en el controlador.

**Tamaño de diseño**

El controlador MBIST tiene un área de 1063

**Limitaciones**

* No se puede acceder directamente a los punteros de la memoria, por lo que es necesario recorrerla en todo momento.
* No existe pila de instrucciones por lo que colocar dos instrucciones al mismo puede dañar el funcionamiento del mismo.

**Tiempo estimado de pruebas**

* Se genera un banco de pruebas que corre todos los patrones y modos aceptados y el mismo tiene una duración de 8.5 microsegundos

**Configuración**

Para configurar el MBIST, es necesario configurar 3 pines:

* CLK
* RESET
* Inst\_Bist

Inst\_bist, es que carga la instrucción en el MBIST y modifica el comportamiento del mismo, el mismo es una instrucción de 16 bits, donde los primeros 3, representan el modo los cuales son:

* IDLE: El MBIST reinicia todos los registros internos y coloca los multiplexores en modo normal, con el fin que el FIFO asincrónico, funcione como debería.
* Autocheck: El mismo recorre toda la memoria escribiendo, leyendo y comparando el dato según el patrón dado
* DEBUG: Este escribe, lee y compara en una dirección dada, la misma es dada en la instrucción

Los próximos 3 bits, representan, el patrón, los cuales son:

* Todo en 0, Se prueba que todos los datos en memoria sean ceros
* Todo en 1, Se prueba que todos los datos en memoria sean unos
* Checkboard, Se prueba que los valores de la memoria están intercalados ejemplo 0101
* LSFR: Utiliza un registro de desplazamiento retroalimentado para generar patrones pseudo aleatorios

Por último, los siguientes 3 bits proveen la dirección en caso de que se necesite el modo DEBUG.

**Diagrama**

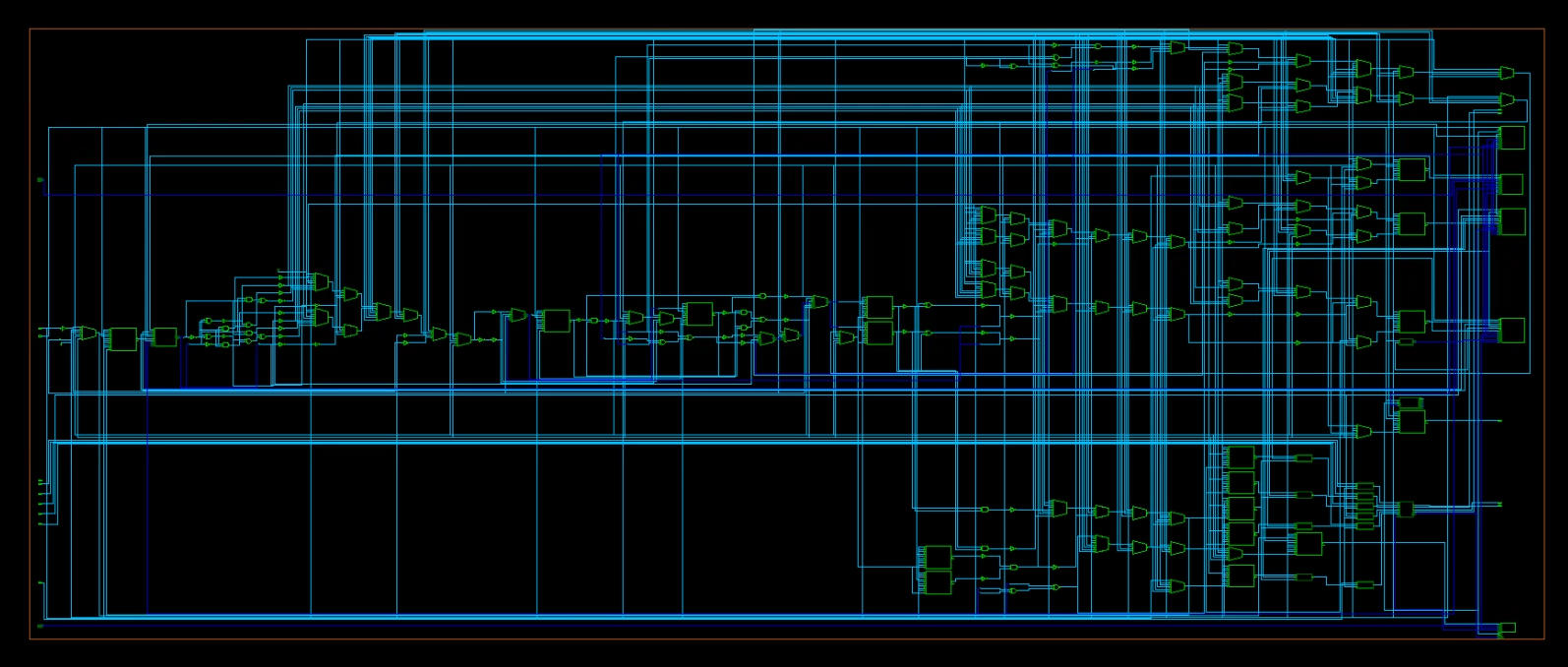


Figura 7 Diagrama de conexiones

**Testbench**:

A nivel de banco de pruebas se coloca la MBIST en los diferentes patrones, modos y direcciones para probar su funcionamiento este es:

// Code your testbench here

// or browse Examples

// Testbench

`timescale 1ns/1ps

parameter DWIDTH = 32;

parameter AWIDTH = 4;

parameter W = 4'd8;

parameter DP = 3'd4;

parameter WR\_FAST = 1'b1;

parameter RD\_FAST = 1'b1;

parameter FULL\_DP = DP;

parameter EMPTY\_DP = 1'b0;

parameter AW = (DP == 2) ? 1 :

(DP == 4) ? 2 :

(DP == 8) ? 3 :

(DP == 16) ? 4 :

(DP == 32) ? 5 :

(DP == 64) ? 6 :

(DP == 128) ? 7 :

(DP == 256) ? 8 : 0;

module test;

parameter P\_SYS = 10;

parameter P\_SDR = 20;

// General

reg RESETN;

reg sdram\_clk;

reg sys\_clk;

initial sys\_clk = 0;

initial sdram\_clk = 0;

always #(P\_SYS/2) sys\_clk = !sys\_clk;

always #(P\_SDR/2) sdram\_clk = !sdram\_clk;

reg clk\_bist;

reg arst\_bist;

reg [15:0] inst\_bist;

reg stas\_check;

reg check\_bist;

reg pass\_fail\_check;

/\*Inputs for FIFOS Memory\*/

reg w\_e;

reg r\_e;

reg [W-1 : 0] w\_data;

reg [AW:0] pntr;

reg [W-1 : 0] r\_data;

wire full;

wire afull;

wire empty;

wire aempty;

mbist\_controller top

(

.clk\_bist(sys\_clk),

.arst\_bist(arst\_bist),

.inst\_bist(inst\_bist),

.done\_check(stast\_check),

.check\_bist(check\_bist),

.pass\_fail\_check(pass\_fail\_check),

.wr\_clk(sys\_clk),

.wr\_reset\_n(RESETN),

.wr\_en(w\_e),

.wr\_data(w\_data),

.full(full),

.afull(afull),

.rd\_clk(sdram\_clk),

.rd\_reset\_n(RESETN),

.rd\_en(r\_e),

.empty(empty),

.aempty(aempty),

.rd\_data(r\_data)

);

initial begin

// Dump waves

$dumpfile("dump.vcd");

$dumpvars(0,top);

RESETN = 1;

#10;

#10 assign arst\_bist = 1;

#100 assign arst\_bist = 0;

#10 assign inst\_bist = 16'b001001;

#1000;

#10 assign inst\_bist = 16'b000001;

#1000;

#10 assign inst\_bist = 16'b010001;

#1000;

#10 assign inst\_bist = 16'b011001;

#1000;

#10 assign inst\_bist = 16'b000000;

#500

#10 assign inst\_bist = 16'b011000010;

#1000;

#10 assign inst\_bist = 16'b001001010;

#1000;

#10 assign inst\_bist = 16'b001010010;

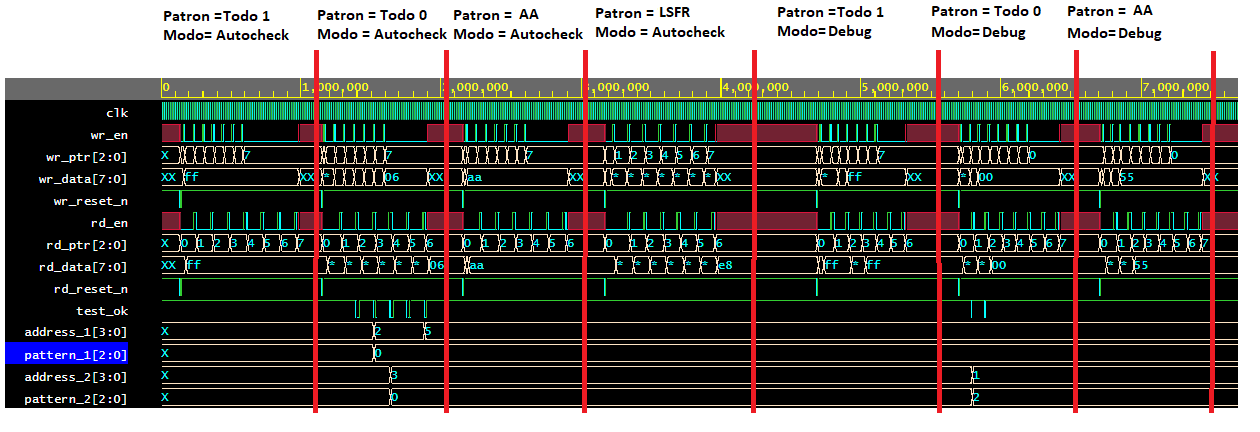
#1000;

$finish;

end

endmodule

**Simulación:**

****

# 13. Configuraciones

# 14. Casos de uso:

## Nivel de aplicación.

**No aplica**, ya que no corre ninguna aplicación ni software. El mínimo en este caso es que el controlador acceda a toda la memoria que se encuentra conectada. Esto se realiza configurando correctamente los límites a nivel de aplicación.

## Nivel de interfaz.

**No aplica**. En este caso, debe cumplir con las especificaciones mínimas para conectarse a una SDRAM.

## Protocolos y estándares.

**No aplica**. Se especifica únicamente que debe conectarse a memorias SDRAM. Sin embargo, no se especifica claramente el protocolo que debe seguir. Por definición, existen variantes como PC100 y PC133, que son los antecesores a DDR. Se tomará como **suficiente que el controlador pueda hacer lecturas y escrituras** a los **tres módulos de RAM usados** para la verificación.

## Rendimiento.

**No aplica.** No se valida el rendimiento del controlador ya que las validaciones deseadas va por el camino del funcionamiento base del controlador y no necesariamente cumplir con alguna medida de velocidad o bien de cantidad de información leída y escrita por segundo, por mencionar ejemplos.

# 15. Interrupciones:

**No aplica.** El DUV no tiene interrupciones ni las genera. Los únicos elementos que generan interrupciones son las memorias, las cuales pueden hacer que por ejemplo se demore un poco el refrescamiento automático de la misma memoria.

# 16. Manejo de Errores:

**No aplica**. El controlador de memoria no involucra ningún manejo de errores. Lo más cercano es el manejo de cruces de página (*page crossovers*) y puede incurrir a defectos en la señal.

# 17. Funciones

**Observaciones generales:**