

# Capacidades de Verificación Funcional en Circuitos Digitales

Hipólito Guzmán Miranda  
Departamento de Ingeniería Electrónica  
Universidad de Sevilla

[hguzman@us.es](mailto:hguzman@us.es)

Acknowledgement to Ray Salemi,  
Mentor Graphics

# Contexto docente

- Arquitectura de FPGAs
- VHDL avanzado
- Capacidades de verificación funcional en Circuitos Digitales

Conocimientos previos requeridos:

- VHDL avanzado

## Objetivos de aprendizaje

- Conocer las limitaciones de los testbenches clásicos
- Conocer las métricas de verificación más comunes, como pueden ser el alcance de código y el alcance funcional
- Comprender el concepto de modelado a nivel de transacción
- Adquirir las capacidades conceptuales para construir paso a paso un testbench estructurado

# Repaso

VHDL avanzado:

- Otra forma de estructurar nuestro código
- Atención a la ocupación de recursos en síntesis

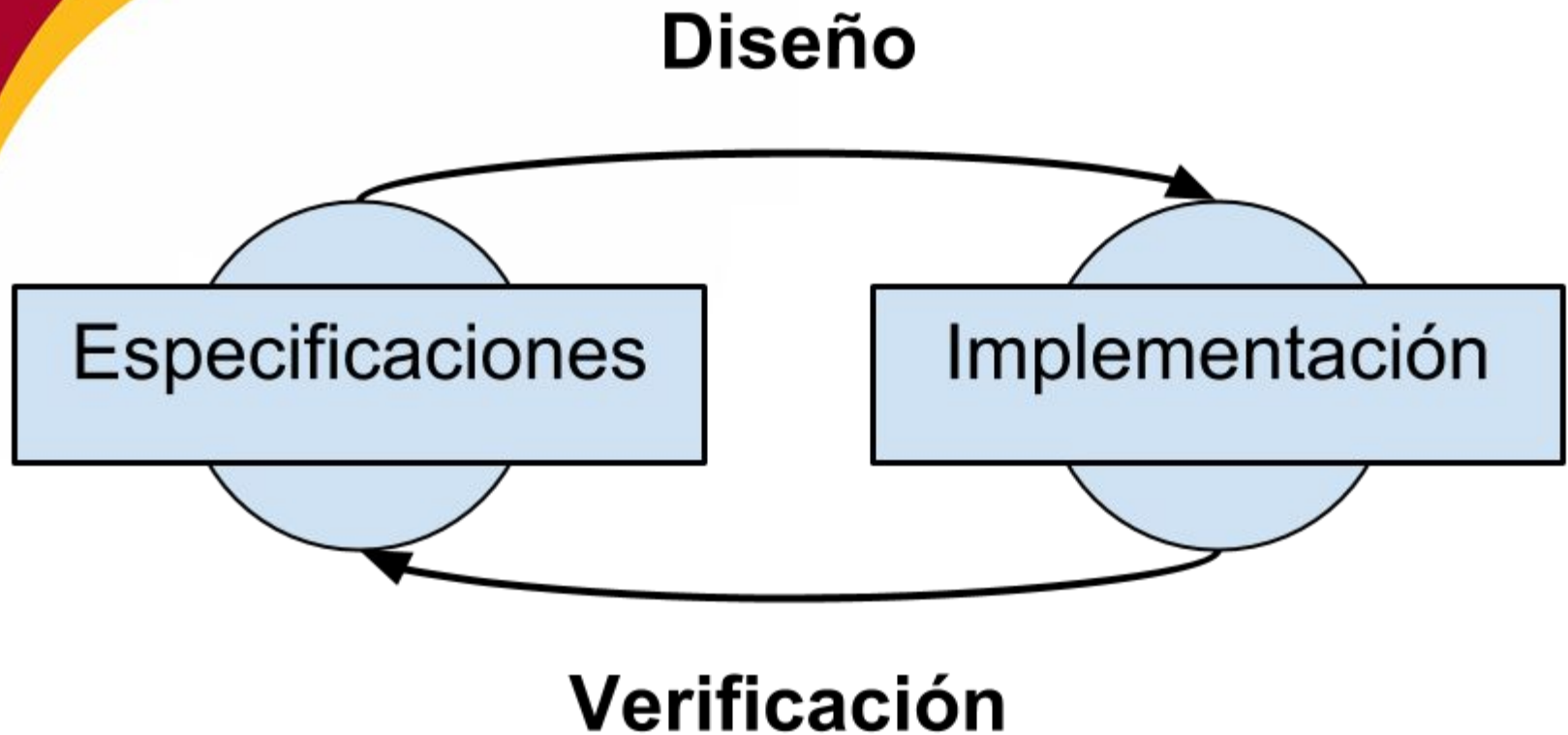
## Contenido

- ¿Qué es la verificación?
- ¿Por qué verificar?
- 0.- Test dirigido
- 1.- Code coverage
- 2.- Assertions
- 3.- Modelado a nivel de transacción
- 4.- Self-checking testbenches
- 5.- Estímulos automáticos
- 6.- Functional coverage
- Bibliografía

## Contenido

- ¿Qué es la verificación?
- ¿Por qué verificar?
- 0.- Test dirigido
- 1.- Code coverage
- 2.- Assertions
- 3.- Modelado a nivel de transacción
- 4.- Self-checking testbenches
- 5.- Estímulos automáticos
- 6.- Functional coverage
- Bibliografía

# ¿Qué es la verificación?



Comprobar que la implementación realizada realmente cumple con las especificaciones

## Contenido

- ¿Qué es la verificación?
- ¿Por qué verificar?
- 0.- Test dirigido
- 1.- Code coverage
- 2.- Assertions
- 3.- Modelado a nivel de transacción
- 4.- Self-checking testbenches
- 5.- Estímulos automáticos
- 6.- Functional coverage
- Bibliografía



## Algunas buenas razones...

- Verification gap
- Salud mental
- Verificación puede ser entre el 50% y el 80% del tiempo total del desarrollo
- Costes de fabricación de ASIC
- Dificultad de diagnosticar y arreglar fallos sobre el prototipo FPGA
- Terminar (de verdad) los proyectos a tiempo



# Motivación

## ¿Por qué aprender verificación?

indeed

Find jobs

Company reviews

Find salaries

Upload your resume

What

Job title, keywords, or company

Where

City, state, or zip code

Verification engineer



Find jobs

Advanced Job Search

Page 1 of 22,569 jobs

Verification engineer jobs

Sort by:

relevance - date

Salary Estimate

\$70,000+ (17885)

\$80,000+ (15639)

\$90,000+ (12300)

\$100,000+ (8576)

\$115,000+ (3912)

Job Type

Full-time (21360)

Internship (904)

Contract (833)

Temporary (636)

Part-time (483)

Commission (33)

### IP Verification Engineer new

Intel 4.1 ★

Hillsboro, OR 97124

Verification of complex server IP designs us  
will be responsible for, although not limited

Today · [Save job](#) · more...

### Design Verification Engineer - E

Apple 4.2 ★

Santa Clara Valley, CA 95014

Pre-silicon digital verification engineer for m  
constrained random verification techniques

30+ days ago · [Save job](#) · more...

### Design Verification Engineer new

Annappurna Labs (U.S.) Inc. 3.6 ★

### ASIC Design Verification Engineer, Processors new

Google 4.3 ★

Sunnyvale, CA

Experience with security-focused verification methods. You will collaborate closely with design and verification engineers in active projects and perform hands...

5 days ago · [Save job](#) · more...

### ASIC Design Verification Engineer new

Google 4.3 ★

Sunnyvale, CA +1 location

You will collaborate closely with design and verification engineers in active projects and perform hands-on verification. 4 years of relevant experience.

5 days ago · [Save job](#) · more...

### Verification Engineer

Ambarella 3.6 ★

Santa Clara, CA 95054

Perform Block Verification of Ambarella's very complex CABAC compression block. Perform system-level verification of Ambarella's Video Input block as well as...

30+ days ago · [Save job](#) · more...

### Design Verification Engineer

Apple 4.2 ★

Santa Clara Valley, CA 95014 +5 locations

Experience with mixed signal verification methodology. Deep knowledge of formal verification methodology. Develop verification plans for all features under your...

30+ days ago · [Save job](#) · more...

### Design Verification Engineer

Talent 101 4.0 ★

Santa Clara, CA 95051 +1 location

## Contenido

- ¿Qué es la verificación?
- ¿Por qué verificar?
- 0.- Test dirigido
- 1.- Code coverage
- 2.- Assertions
- 3.- Modelado a nivel de transacción
- 4.- Self-checking testbenches
- 5.- Estímulos automáticos
- 6.- Functional coverage
- Bibliografía

## Test 'tradicional'

Típico testbench:

- Estímulos definidos a mano
- Siempre los mismos estímulos
- Comprobación mirando “a ojo” la forma de onda

Este enfoque no escala para tests complejos

Ej: 200K estímulos y 1M ciclos de reloj de formas de onda que comprobar

## Test dirigido vs aleatorio

- Los tests pueden ser de dos tipos:
  - Directed
    - Estímulos determinados de antemano
    - (puedo haber pre-calculado la salida esperada)
  - Random
    - Estímulos generados cada vez que se lanza la simulación
    - (¿cómo sé si la salida es correcta?)

## Test dirigido

- En ambos casos, ¿es difícil estar seguro de que lo hemos probado todo!
- ¿Cuándo sé que he acabado de verificar?

## Contenido

- ¿Qué es la verificación?
- ¿Por qué verificar?
- 0.- Test dirigido
- 1.- Code coverage
- 2.- Assertions
- 3.- Modelado a nivel de transacción
- 4.- Self-checking testbenches
- 5.- Estímulos automáticos
- 6.- Functional coverage
- Bibliografía

## Es una técnica automática

- La hace el simulador, compilando los ejecutables de simulación con ciertas opciones
- No soportado por Xilinx ISIM :(
- Soportado por ModelSim/Quarta, Aldec, GHDL (parcialmente), etc...



## Identificar código que no ha sido probado

- Mnemónico :  
“**S**ome **B**eers **F**or **E**xtra **C**ourage”
- Statement
- Branch
- FSM
- Expression
- Condition

## Statement Coverage

- Qué sentencias se han ejecutado y cuáles no
- Sentencia es cualquier cosa que termine en punto y coma
- Una sentencia (como mucho) por línea facilita la labor de cálculo del coverage al simulador

Testplan Design DesUnits

- [-] tb\_bit2symb
- [-] edelweiss\_common
- [-] vhdl\_verification (no coverage)
- [-] image\_pkg
- [-] txt\_util
- [-] tb\_d\_ff
- [-] tb\_fadapt
  - [-] clkmanager\_inst
  - [-] datagen\_inst
  - [-] uut
  - [-] datacompare\_inst
  - [-] throughputchecker\_inst
- [-] tb\_fifo
- [-] tb\_pulse\_shaping
- [-] tb\_qdelay
- [-] tb\_symb2chip
- [-] tb\_top\_tx
- [-] tb\_upsampling
- [-] tb\_dem\_filter
- [-] tb\_downsampling
- [-] tb\_tap

```

144     32     n_ppdu <= PHR (bit_count);
145     32     n_state <= count_cycles_header;
146         if (bit_count = 7) then
147             4     n_state <= wait_for_data;
148             4     n_byte_count <= 0;
149             4     n_bit_count <= 0;
150         end if;
151     end if;
152     when wait_for_data =>
153         if (remaining = 0) then
154             3     n_state <= idle;
155             3     n_count <= THROUGHPUT - 3; -- conserve throughput between frames
156         elsif (count > 1) then
157             1275534 n_count <= count - 1;
158         elsif (not empty = '1') then
159             3080     rden <= '1';
160             3080     n_count <= THROUGHPUT - 1;
161             3080     n_state <= data;
162         end if;
163     when data =>
164         5048     n_ppdu <= fifo_ppdu;
165         5048     n_ppdu_valid <= '1';
166         5048     n_remaining <= remaining-1;
167         5048     n_state <= wait_for_data;
168     when others =>
169         0     n_state <= idle;
170         0     n_bit_count <= 0;
171         0     n_byte_count <= 0;
172         0     rden <= '0';
173         0     n_count <= THROUGHPUT - 1;
174     end case;
175 end process;
176
177
178 sinc: process(clk, rst)
179 begin
180     if (rst = '1') then
181         22     state <= idle;
182         22     bit_count <= 0;
183         22     byte_count <= 0;
184         22     ppdu <= '0';
185         22     pddu_valid <= '0';

```

## Branch Coverage

- Puede ser que entremos en un `if`, pero, ¿por cuál `if` / `elsif` / `else` salimos? ¿por cuál `'when X =>'` ?
- Evalúa si se han alcanzado las distintas ramificaciones de nuestro código

# Branch Coverage

case state is

**83.33%**

Branch	Source	Hits	Status
TRUE	when idle =>	13	Covered
TRUE	when count_cycles_header =>	79676	Covered
TRUE	when header =>	192	Covered
TRUE	when wait_for_data =>	1278618	Covered
TRUE	when data =>	5048	Covered
TRUE	when others =>	0	<b>ZERO</b>

if ((Looped = true) or (Head /= Tail)) then

**50.00%**

Branch	Source	Hits	Status
IF	if ((Looped = true) or (Head /= Tail)) then	385	Covered
ALL FALSE	if ((Looped = true) or (Head /= Tail)) then	0	<b>ZERO</b>

## FSM Coverage

- Al verificar máquinas de estado, nos interesa saber si hemos cubierto:
  - Los estados
  - Las posibles transiciones entre estados
- Para el 'when others =>' normalmente hay que añadir una excepción

<a href="#">state</a>		<b>83.33%</b>
States / Transitions	Hits	Status
State: idle	390214	<b>Covered</b>
Trans: idle -> count_cycles_header	4	<b>Covered</b>
Trans: idle -> idle	390209	<b>Covered</b>
State: count_cycles_header	79676	<b>Covered</b>
Trans: count_cycles_header -> header	192	<b>Covered</b>
Trans: count_cycles_header -> idle	0	<b>ZERO</b>
Trans: count_cycles_header -> count_cycles_header	79484	<b>Covered</b>
State: header	192	<b>Covered</b>
Trans: header -> count_cycles_header	188	<b>Covered</b>
Trans: header -> wait_for_data	4	<b>Covered</b>
Trans: header -> idle	0	<b>ZERO</b>
Trans: header -> header	0	<b>ZERO</b>
State: wait_for_data	1673819	<b>Covered</b>
Trans: wait_for_data -> idle	3	<b>Covered</b>
Trans: wait_for_data -> data	3080	<b>Covered</b>
Trans: wait_for_data -> wait_for_data	1670735	<b>Covered</b>
State: data	3080	<b>Covered</b>
Trans: data -> wait_for_data	3080	<b>Covered</b>
Trans: data -> idle	0	<b>ZERO</b>
Trans: data -> data	0	<b>ZERO</b>

## Expression Coverage

Cuando asignamos:

```
salida <= a OR (b AND c);
```

Si salida = '1'...

- ¿Es porque a = '1' ?
- ¿Es porque b = '1' y c = '1' ?

Si salida = '0'...

- ¿Es porque a, b = '0'?
- ¿Es porque a, c = '0'?

Queremos asegurarnos de que hemos probado todos los casos



## Condition Coverage

Como Expression Coverage, pero en las condiciones en lugar de las asignaciones:

```
if(a='1' OR (b='1' AND c='1')) then
```

- ¿a = '1'?
- ¿b='1' y c='1'?

**else**

- ¿a = '0' y b = '0'?
- ¿a = '0' y c = '0'?



# Condition Coverage

## FEC : Focused Expression Coverage

<b>FEC Condition:</b> <u>if (i_index = 30 AND q_index = 31)</u> <u>then</u>			<b>50.00%</b>
Input Term	Covered	Reason For No Coverage	Hint
(i_index = 30)	Yes		
(q_index = 31)	No	'_0' not hit	Hit '_0'
Rows	FEC Target	Hits	Matching Input Patterns
Row 1	(i_index = 30)_0	2	{ 0- }
Row 2	(i_index = 30)_1	2	{ 11 }
Row 3	(q_index = 31)_0	0	{ 10 }
Row 4	(q_index = 31)_1	2	{ 11 }

## Contenido

- ¿Qué es la verificación?
- ¿Por qué verificar?
- 0.- Test dirigido
- 1.- Code coverage
- 2.- Assertions
- 3.- Modelado a nivel de transacción
- 4.- Self-checking testbenches
- 5.- Estímulos automáticos
- 6.- Functional coverage
- Bibliografía

## Notifica si una condición no se cumple

```
assert condition report string  
severity severity_level;
```

4 niveles de gravedad:

- note
- warning
- error
- failure (stops simulation)

## ¿Son sintetizables?

- No son sintetizables, pero tampoco impiden la síntesis
- El sintetizador en general no mira los assertions
  - Sólo puede mirar aquellos assertions cuya condición sea estática (por ejemplo para evitar síntesis con GENERICS inválidos), y realiza el chequeo en tiempo de síntesis
- Sólo los tiene en cuenta el simulador

## Tipos de assertions

- **Firewall assertions**
  - Para asegurar que tus bloques están siendo usados correctamente
  - Los suele añadir el ingeniero de diseño
- **Protocol monitor**
  - Para asegurar que diferentes bloques se están comunicando entre sí correctamente (están cumpliendo el protocolo)
  - Los suele añadir el ingeniero de verificación
  - Un protocol monitor tiene más que assertions

(Pero se usan las mismas sentencias VHDL)

## Ejemplos

- **En VHDL:**

```
assert (cont >= 0 and cont <= 7)  
  report "cont overflow, should  
never happen!" severity failure;
```

También hay assertions en otros lenguajes como PSL o SystemVerilog

```
assert DATA_LENGTH > 0
```

```
report "fadapt : DATA_LENGTH must be a positive  
non-zero integer"  
severity failure;
```

```
assert (NOT (ifull = '1' and wr_en='1' and  
falling_edge(clk)))
```

```
report "fadapt : Trying to write in a full fifo: data  
will be lost. Check throughput of blocks"  
severity failure;
```

```
assert (NOT (empty = '1' and rd_en='1' and  
falling_edge(clk)))
```

```
report "fadapt : Trying to read from an empty fifo:  
invalid data will be processed"  
severity failure;
```



## Si la condición es compleja, mejor en un `if`

- También se puede usar `report` sin `assert`:

```
if (output /= expected) then  
    report (“error in data”)  
    severity error;  
end if;
```

## Contenido

- ¿Qué es la verificación?
- ¿Por qué verificar?
- 0.- Test dirigido
- 1.- Code coverage
- 2.- Assertions
- 3.- Modelado a nivel de transacción
- 4.- Self-checking testbenches
- 5.- Estímulos automáticos
- 6.- Functional coverage
- Bibliografía

## Transaction-Level Modelling (TLM)

Es elevar el nivel de abstracción en verificación, separando:

- Los datos que se mueven por los interfaces

de

- El movimiento de pines y señales de control asociado

## Transaction-Level Modelling (TLM)

Por ejemplo si enviamos datos por una fifo:

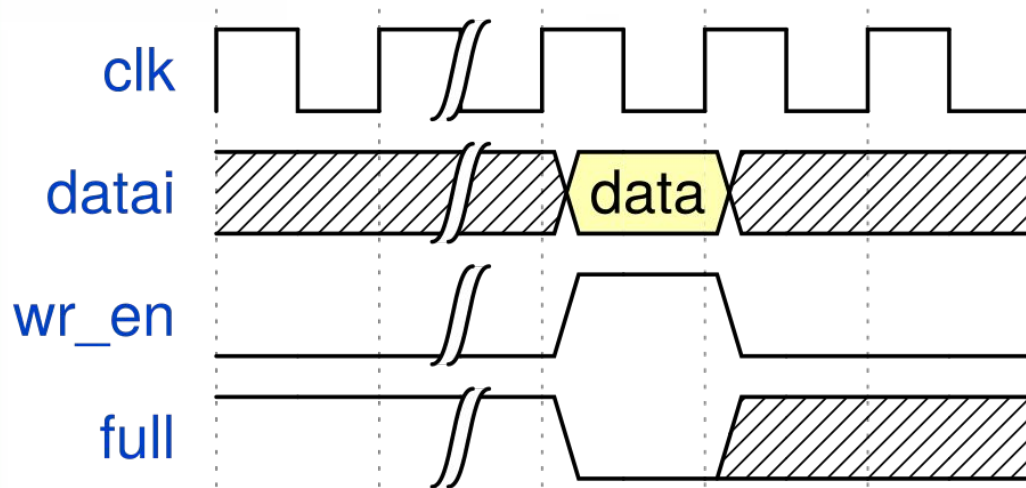
1. Esperamos a que no esté llena
2. Ponemos el dato
3. Activamos write\_enable
4. Esperamos un ciclo de reloj
5. Desactivamos write\_enable

Queremos separar el envío del dato (fifo\_write) del movimiento de pines (full, write\_enable, datai)

## Transaction-Level Modelling (TLM)

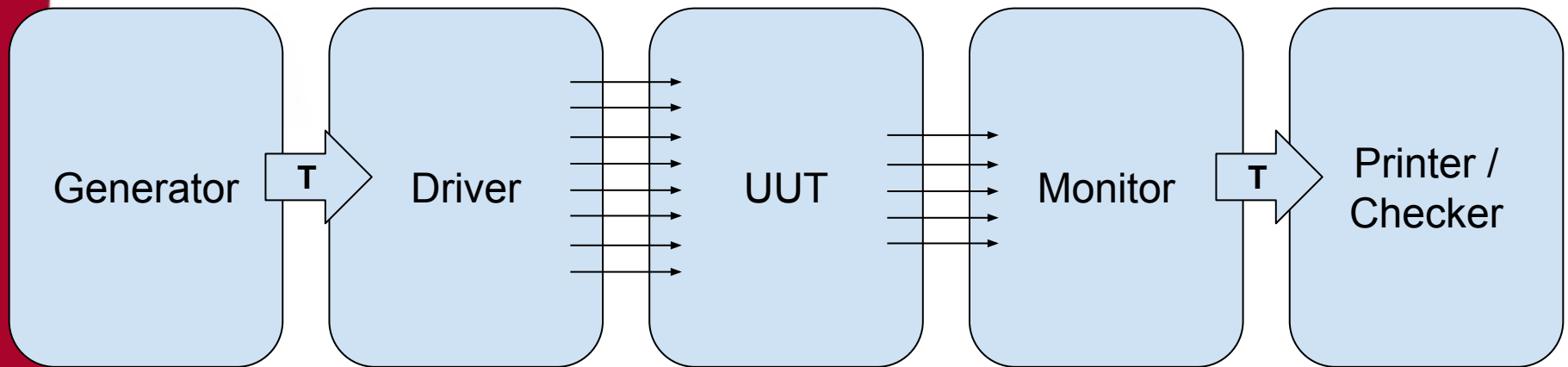
```
fifo_write(data);
```

← data se propaga por el interfaz



← movimiento de pines

## Arquitectura de un testbench TLM



## ¿Cómo se hace?

Definimos un record (o un protected type en VHDL-2002/2008) con los datos asociados a cada canal:

```
type input_tran is
  record
    a   : std_logic_vector (7 downto 0);
    b   : std_logic_vector (7 downto 0);
    op  : op_type;
  end record;
```

## ¿Cómo se hace?

- Definimos entidad (basada en processes/procedures) para convertir las transacciones en movimiento de pines ('driver')
- Describimos entidad (basada en processes/procedures) para convertir el movimiento de pines en transacciones ('monitor')



## Plan de pruebas

Ahora es más fácil definir un plan de pruebas:

- Ante X transacción(es) de entrada, se espera Y transacción(es) de salida

Más información en el Tema 8

## Contenido

- ¿Qué es la verificación?
- ¿Por qué verificar?
- 0.- Test dirigido
- 1.- Code coverage
- 2.- Assertions
- 3.- Modelado a nivel de transacción
- 4.- Self-checking testbenches
- 5.- Estímulos automáticos
- 6.- Functional coverage
- Bibliografía

## Self-checking Testbenches

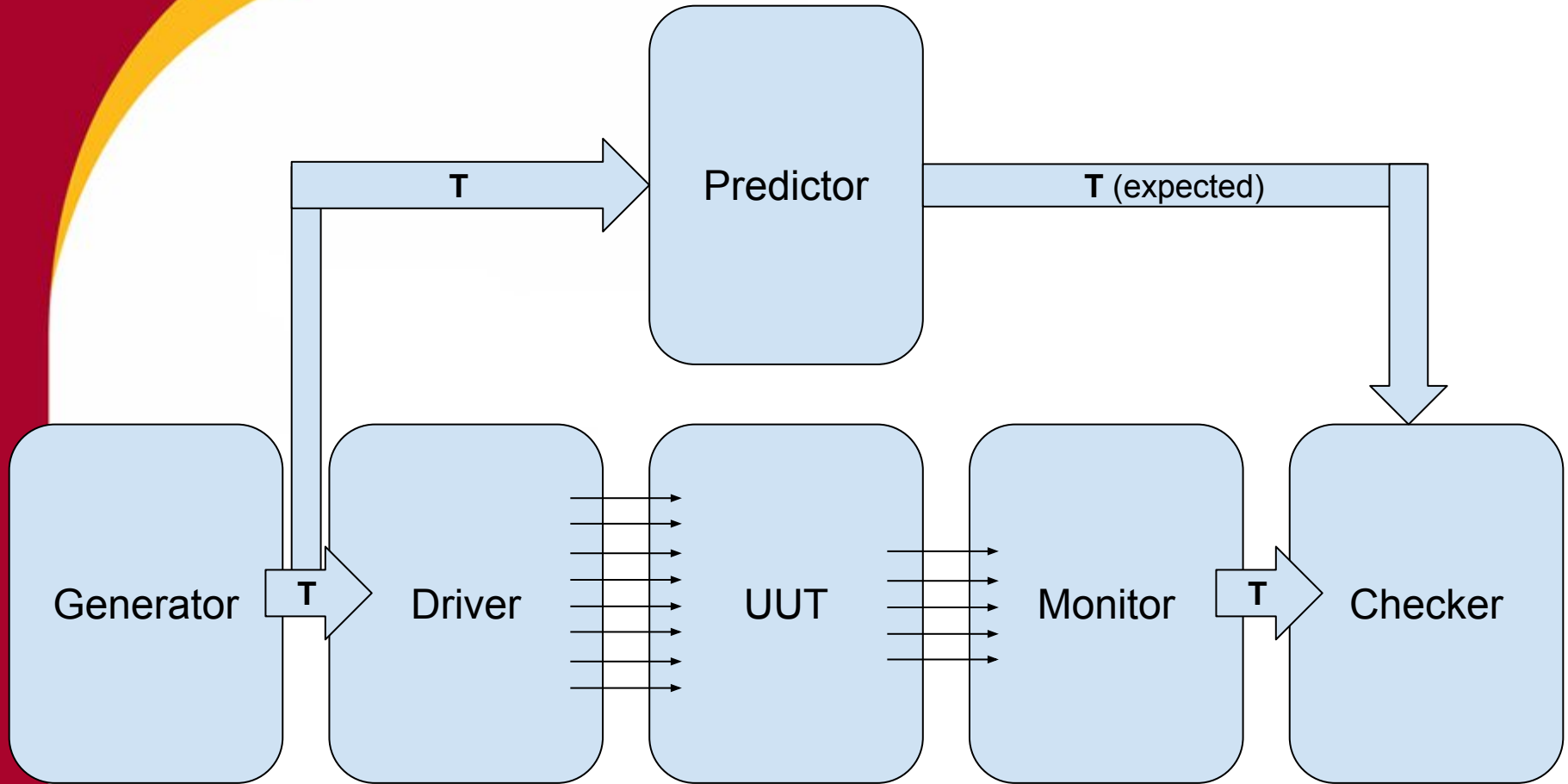
- En lugar de comprobar a mano las formas de onda, insertamos en el testbench comprobaciones de:
  - Si los movimientos de pines son correctos (assertions del protocol monitor)
  - Si los datos de salida son correctos
- Predictor: predice las transacciones de salida esperadas
- Checker: comprueba si las transacciones son correctas.

## Diseño del checker

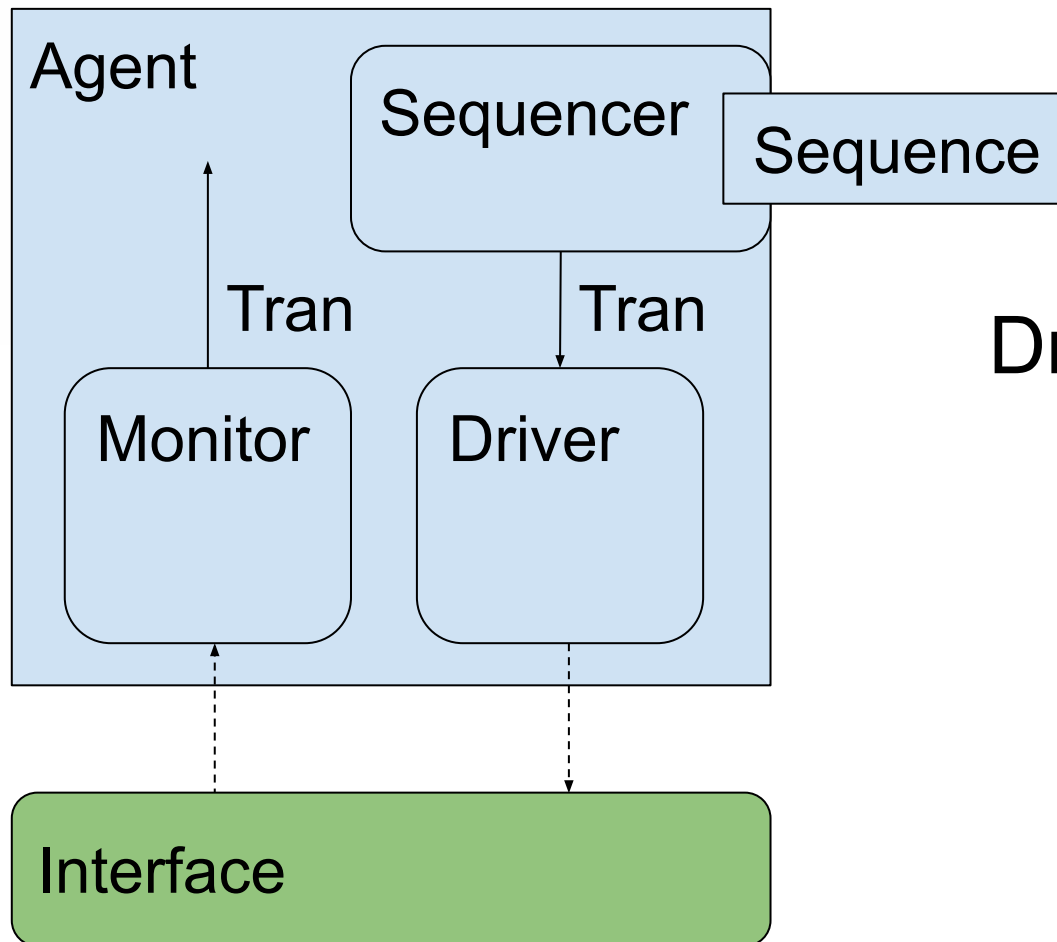
Dos opciones:

1. Generar ficheros de salida 'gold' provenientes de un modelo de alto nivel
  - Por ejemplo, para crosscheck con Matlab/octave
2. Integrar modelo de alto nivel en la simulación
  - Modelo realizado en VHDL o (System)Verilog
  - Interfaz QuestaSim-Matlab
  - Interfaz VHDL-C (GHDL, QuestaSim)
  - Python (CoCoTb)

# Self-checking Testbenches



# Agente de verificación



Driver + Monitor

## Contenido

- ¿Qué es la verificación?
- ¿Por qué verificar?
- 0.- Test dirigido
- 1.- Code coverage
- 2.- Assertions
- 3.- Modelado a nivel de transacción
- 4.- Self-checking testbenches
- 5.- Estímulos automáticos
- 6.- Functional coverage
- Bibliografía

## Estímulos automáticos

Si tenemos un 'checker' que incluye un modelo de alto nivel con el que comparar:

- Podemos generar estímulos aleatorios
- 'Test random' como contraposición a 'test dirigido'
- En realidad es 'constrained random' porque se aplican restricciones a los estímulos generados



## Contenido

- ¿Qué es la verificación?
- ¿Por qué verificar?
- 0.- Test dirigido
- 1.- Code coverage
- 2.- Assertions
- 3.- Modelado a nivel de transacción
- 4.- Self-checking testbenches
- 5.- Estímulos automáticos
- 6.- Functional coverage
- Bibliografía

## Alcance Funcional

Code coverage es muy útil pero NO nos dice:

- Si la ejecución fue correcta o no
- Si hemos probado todos los 'corner cases': valores, rangos, etc
- Si estamos aplicando los estímulos en secuencias correctas

Functional coverage indica si estamos cubriendo todo el plan de pruebas

## Alcance Funcional

Ejemplo:

- Multiplicador 32 bits, 4G casos posibles
- Al menos deberíamos probar:
  - positivo \* positivo
  - positivo \* negativo
  - negativo \* positivo
  - negativo \* negativo
  - positivo \* cero
  - negativo \* cero
  - cero \* positivo
  - cero \* negativo
  - cero \* cero

## ¿Cómo se hace?

- Se definen 'bins' (contenedores)
- Cuando se genera la transacción de entrada se anota a qué bin pertenece la transacción generada
- Al final de la simulación se genera un informe del coverage de cada bin (número de veces que se alcanza cada una)

Normalmente se utilizan packages de terceros que ya dan esta funcionalidad (OSVVM CoveragePkg en VHDL)

## Contenido

- ¿Qué es la verificación?
- ¿Por qué verificar?
- 0.- Test dirigido
- 1.- Code coverage
- 2.- Assertions
- 3.- Modelado a nivel de transacción
- 4.- Self-checking testbenches
- 5.- Estímulos automáticos
- 6.- Functional coverage
- Bibliografía

## Bibliografía

- Ray Salemi, *FPGA Simulation: A Complete Step-by-Step Guide*. Boston Light Press, 2009
- Mentor Graphics, [Verification Academy](#)

## Resultados de aprendizaje

- ¿Para qué sirven las métricas de verificación?
- Diferencias entre code coverage y functional coverage
- ¿Por qué tiene sentido hacer test con entradas aleatorias con restricciones (constrained random)?
- Conocer la importancia de los assertions
- Conocer qué es el modelado a nivel de transacción y cómo influye en la construcción de testbenches estructurados