Tema 8: Diseño de planes de pruebas

Hipólito Guzmán Miranda
Departamento de Ingeniería Electrónica
Universidad de Sevilla
hguzman@us.es

Acknowledgement to Ray Salemi,
Mentor Graphics

Contexto docente

BT03: Verificación funcional y formal de circuitos digitales

- Tema 6: Capacidades de verificación funcional en circuitos digitales
- Tema 7: Métodos de verificación formal para circuitos digitales
- Tema 8: Diseño de planes de pruebas

Conocimientos previos requeridos:

- Modelado a nivel de transacción
- Capacidades de verificación funcional

Objetivos de aprendizaje

- Ser capaz de analizar la descripción o especificaciones de un diseño para extraer una lista de funcionalidades a probar
- Saber desarrollar un plan de pruebas para un bloque o sistema digital
- Conocer las diferencias entre tests unitarios y tests a nivel de sistema

Repaso

Verificación formal:

- Especificación de propiedades y suposiciones
- Bounded Model Checking
- K-induction
- Equivalence Checking

Contenido

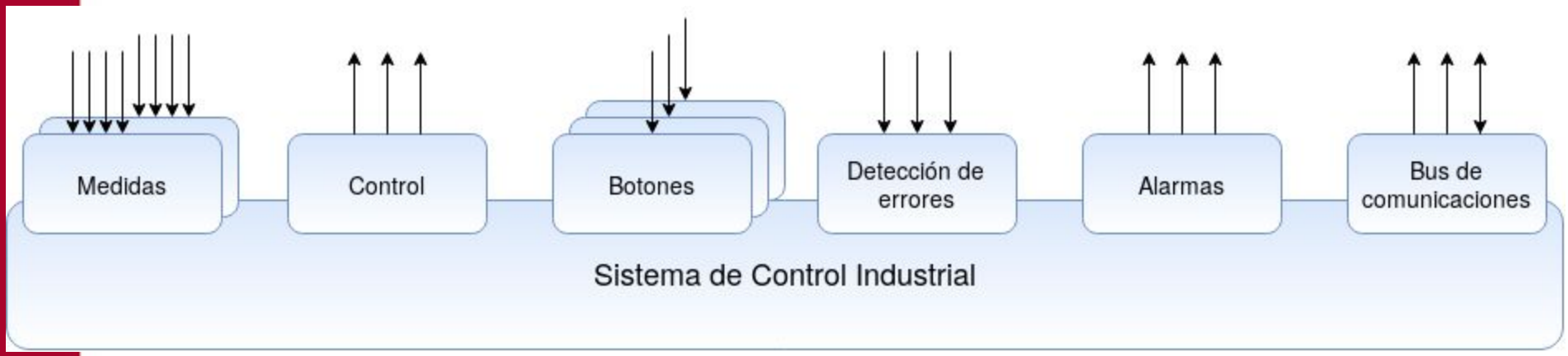
- Motivación
- Funcionalidad del diseño a probar
- Transacciones y operaciones
- Lista de funcionalidades
- Estímulos de entrada
- Respuestas de salida
- Lista de tests
- Tests unitarios
- Tests de sistemas complejos
- Conclusiones
- Bibliografía

Contenido

- Motivación
- Funcionalidad del diseño a probar
- Transacciones y operaciones
- Lista de funcionalidades
- Estímulos de entrada
- Respuestas de salida
- Lista de tests
- Tests unitarios
- Tests de sistemas complejos
- Conclusiones
- Bibliografía

¿Cómo abordar el test de un sistema complejo?

Necesitamos un enfoque sistemático



¿Por qué abordarlo de manera planificada?

- Identificar los aspectos más fundamentales del diseño
- Reducción de riesgo
- Disponer de un enfoque sistemático para probar todas las funcionalidades
  - El “voy probando lo que me va pareciendo” no escala en verificación
  - Necesitaremos pasar todos los tests si cambiamos cualquier cosa en el código: no queremos que se nos olvide nada

Queremos simular todas las sentencias del HDL antes de probar en la FPGA

Contenido

- Motivación
- Funcionalidad del diseño a probar
- Transacciones y operaciones
- Lista de funcionalidades
- Estímulos de entrada
- Respuestas de salida
- Lista de tests
- Tests unitarios
- Tests de sistemas complejos
- Conclusiones
- Bibliografía

Design Under Test

¿Cómo describir de manera sistemática las **funcionalidades** del DUT?

Estas funcionalidades son lo que nuestro plan de pruebas quiere asegurar que verificamos

Necesitamos saber **qué** queremos probar antes de poder planificar **cómo** lo probamos

Funcionalidad del DUT

¿Qué nivel de detalle necesitamos?

Demasiado detalle

- Especificación completa
- Código completo

Demasiado poco detalle

- Diagrama de bloques
- Descripción aproximada de la funcionalidad

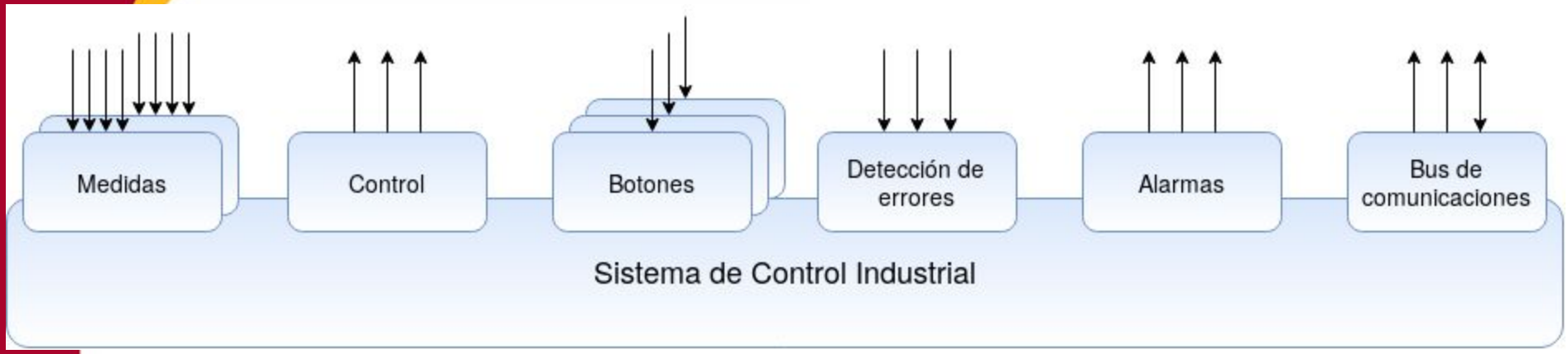
Trabajar desde los pines (descripción de caja negra)

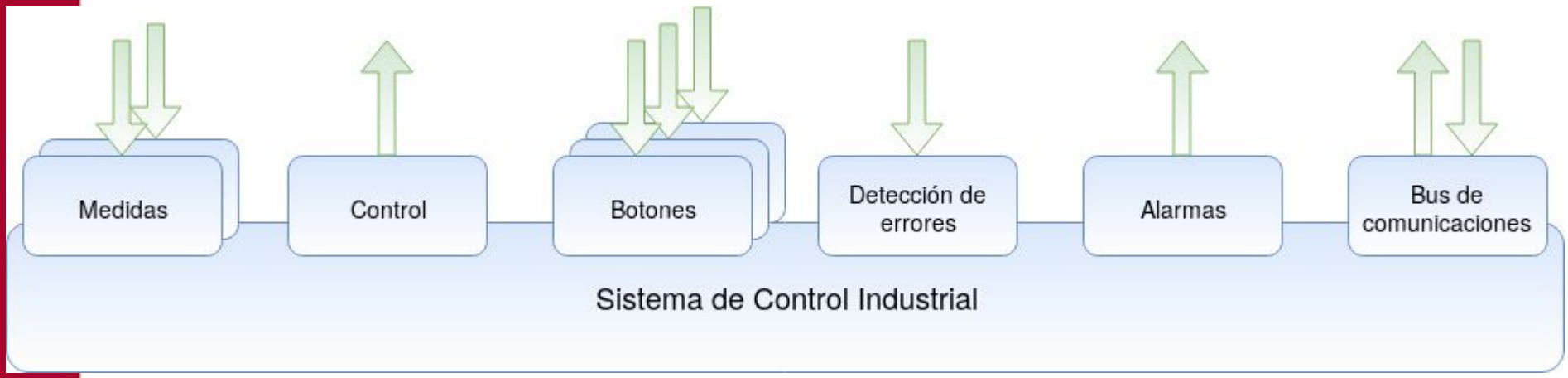
¿Qué hace el circuito? Dadas unas entradas, ¿qué salidas nos debe dar?

- Dividir los interfaces (conjuntos de pines) en canales
- Los canales son unidireccionales
- Puertos bidireccionales se describen como dos canales unidireccionales
- Sobre los canales ocurrirán transacciones

Funcionalidad del DUT

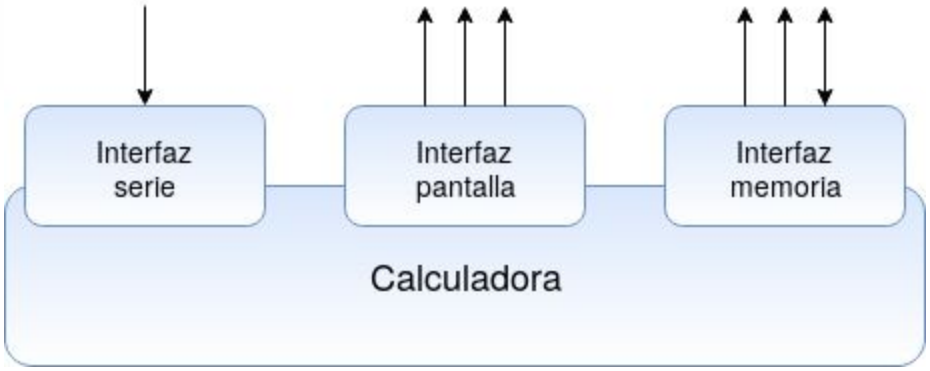
Ejemplo

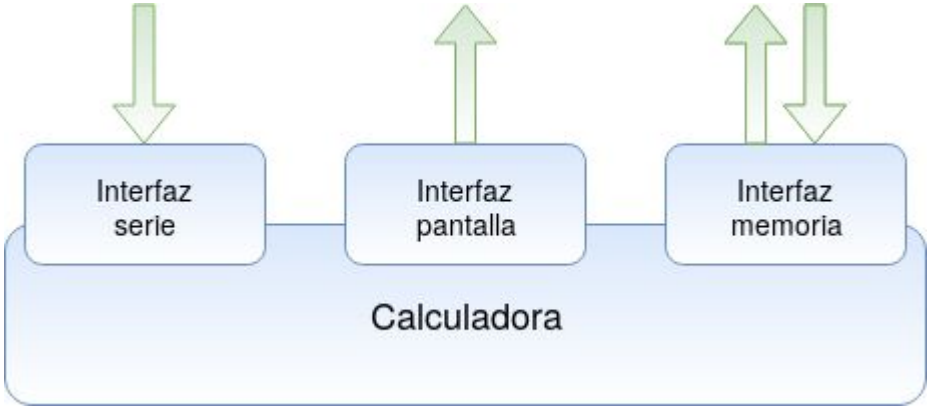




Funcionalidad del DUT

Ejemplo





Contenido

- Motivación
- Funcionalidad del diseño a probar
- Transacciones y operaciones
- Lista de funcionalidades
- Estímulos de entrada
- Respuestas de salida
- Lista de tests
- Tests unitarios
- Tests de sistemas complejos
- Conclusiones
- Bibliografía

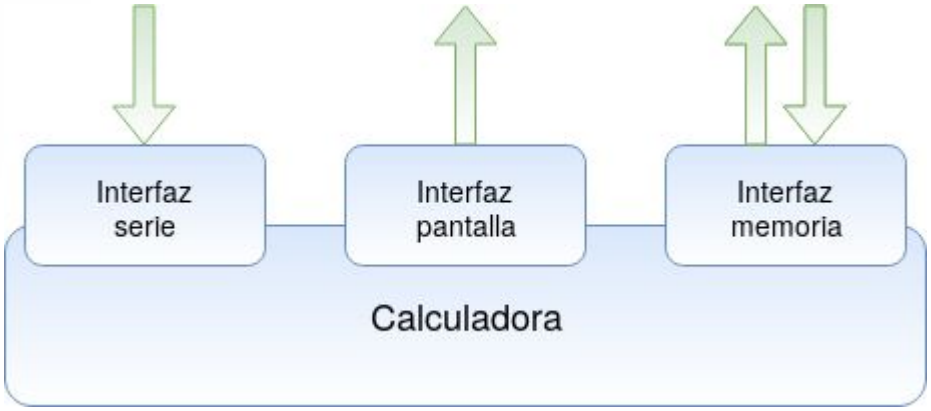
No toda operación es una transacción distinta

Recordemos que una transacción representa un movimiento de pines asociado a datos que se mueven por un interfaz

Una transacción puede codificar qué operación se realiza

Ejemplo

Interfaz serie:




```
receive(opcode, op1, op2)

receive(sum, 3, 5)
receive(multiply, 6, 7)
```

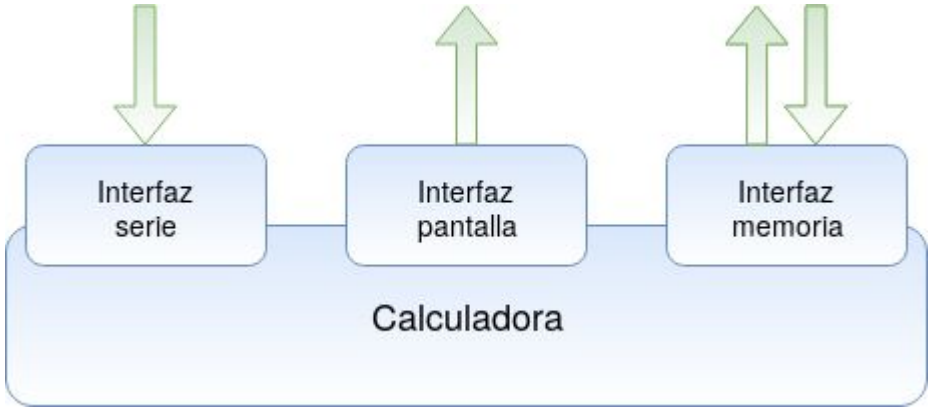
Contenido

- Motivación
- Funcionalidad del diseño a probar
- Transacciones y operaciones
- Lista de funcionalidades
- Estímulos de entrada
- Respuestas de salida
- Lista de tests
- Tests unitarios
- Tests de sistemas complejos
- Conclusiones
- Bibliografía

Describir la funcionalidad que queremos probar

Funcionalidad en función de transacciones es el nivel justo de detalle que necesitamos

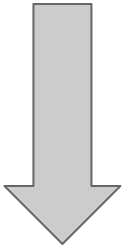
Transacción(es) de entrada → Transacción(es) de salida



Ejemplo

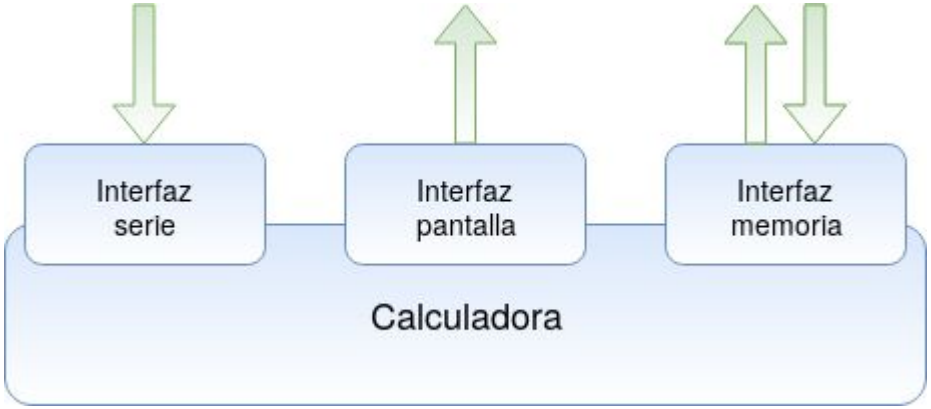
Interfaz serie:

```
receive(mult, 6, 7)
```



Interfaz pantalla:

```
display(42)
```



Contenido

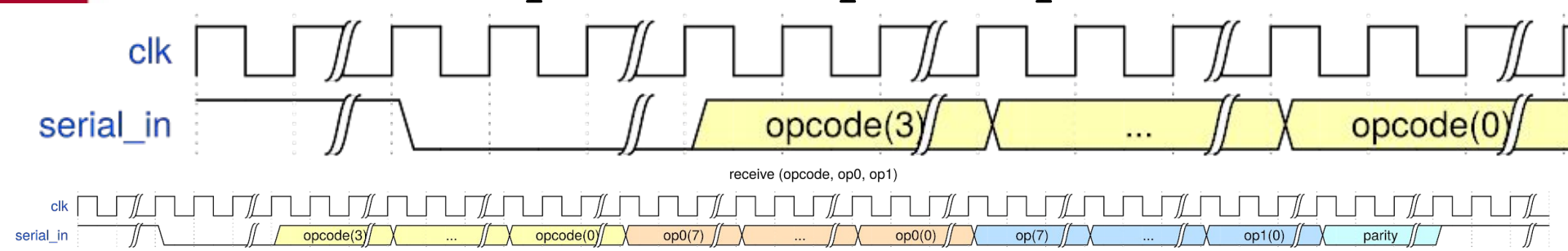
- Motivación
- Funcionalidad del diseño a probar
- Transacciones y operaciones
- Lista de funcionalidades
- Estímulos de entrada
- Respuestas de salida
- Lista de tests
- Tests unitarios
- Tests de sistemas complejos
- Conclusiones
- Bibliografía

Estímulos de entrada

tran(args) → mov. pines

Describir las transacciones de entrada en términos de movimiento de pines en los canales

```
receive(opcode, op0, op1)
```



Contenido

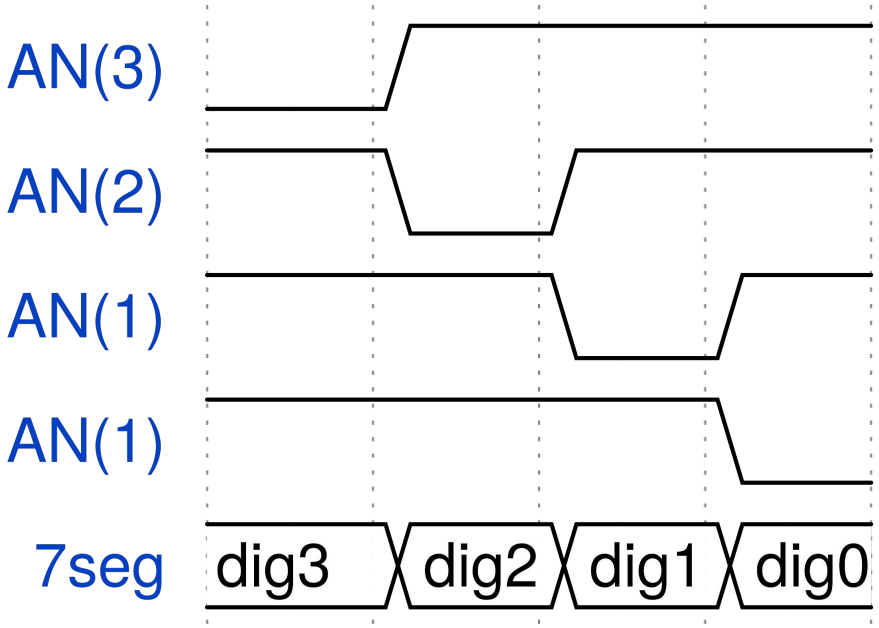
- Motivación
- Funcionalidad del diseño a probar
- Transacciones y operaciones
- Lista de funcionalidades
- Estímulos de entrada
- Respuestas de salida
- Lista de tests
- Tests unitarios
- Tests de sistemas complejos
- Conclusiones
- Bibliografía

Respuestas de salida

tran(args) → mov. pines

Describir las respuestas de salida en términos de movimiento de pines en los canales

```
display(value)
```



Contenido

- Motivación
- Funcionalidad del diseño a probar
- Transacciones y operaciones
- Lista de funcionalidades
- Estímulos de entrada
- Respuestas de salida
- Lista de tests
- Tests unitarios
- Tests de sistemas complejos
- Conclusiones
- Bibliografía

Lista de tests

Lista de funcionalidades + canales + transacciones → lista de tests

serie	display	calc2mem	mem2calc
receive(sum,3,5)	display(8)		
receive(mult,6,7)	display(42)		
receive(set,addr,value)		write(addr,value)	
receive(get,addr)	display(value)	read(addr)	result(value)
...	...	...	...

Contenido

- Motivación
- Funcionalidad del diseño a probar
- Transacciones y operaciones
- Lista de funcionalidades
- Estímulos de entrada
- Respuestas de salida
- Lista de tests
- Tests unitarios
- Tests de sistemas complejos
- Conclusiones
- Bibliografía

Probar una funcionalidad

- Tests sencillos, que prueban una única cosa
- Toda entidad debe tener al menos un testbench
- Se deben pasar los tests a cada commit
- La **gestión** de los tests es más importante que su estructura (se tiende a tener muchos tests pequeños)
- Podemos gestionarlos con Makefile, VUnit, etc

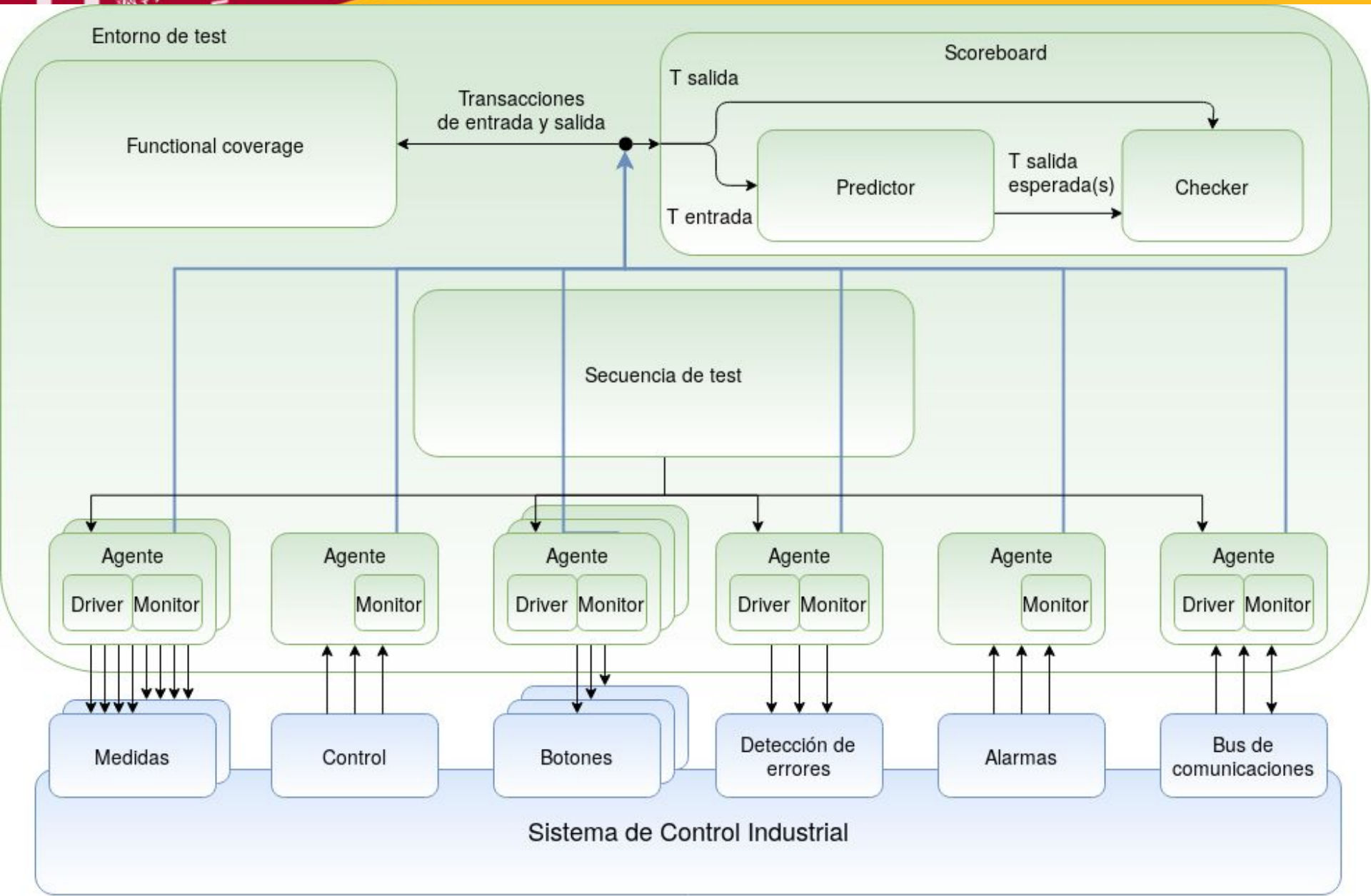
Contenido

- Motivación
- Funcionalidad del diseño a probar
- Transacciones y operaciones
- Lista de funcionalidades
- Estímulos de entrada
- Respuestas de salida
- Lista de tests
- Tests unitarios
- Tests de sistemas complejos
- Conclusiones
- Bibliografía

System-level or integration tests

- Simulación del sistema completo
- Construir un testbench complejo
- Es importante **estructurar** correctamente el testbench
  - UVM (Universal Verification Methodology) → SystemVerilog
  - UVVM (Universal VHDL Verification Methodology) → VHDL
- Las metodologías de verificación modernas suelen centrarse en este tipo de tests

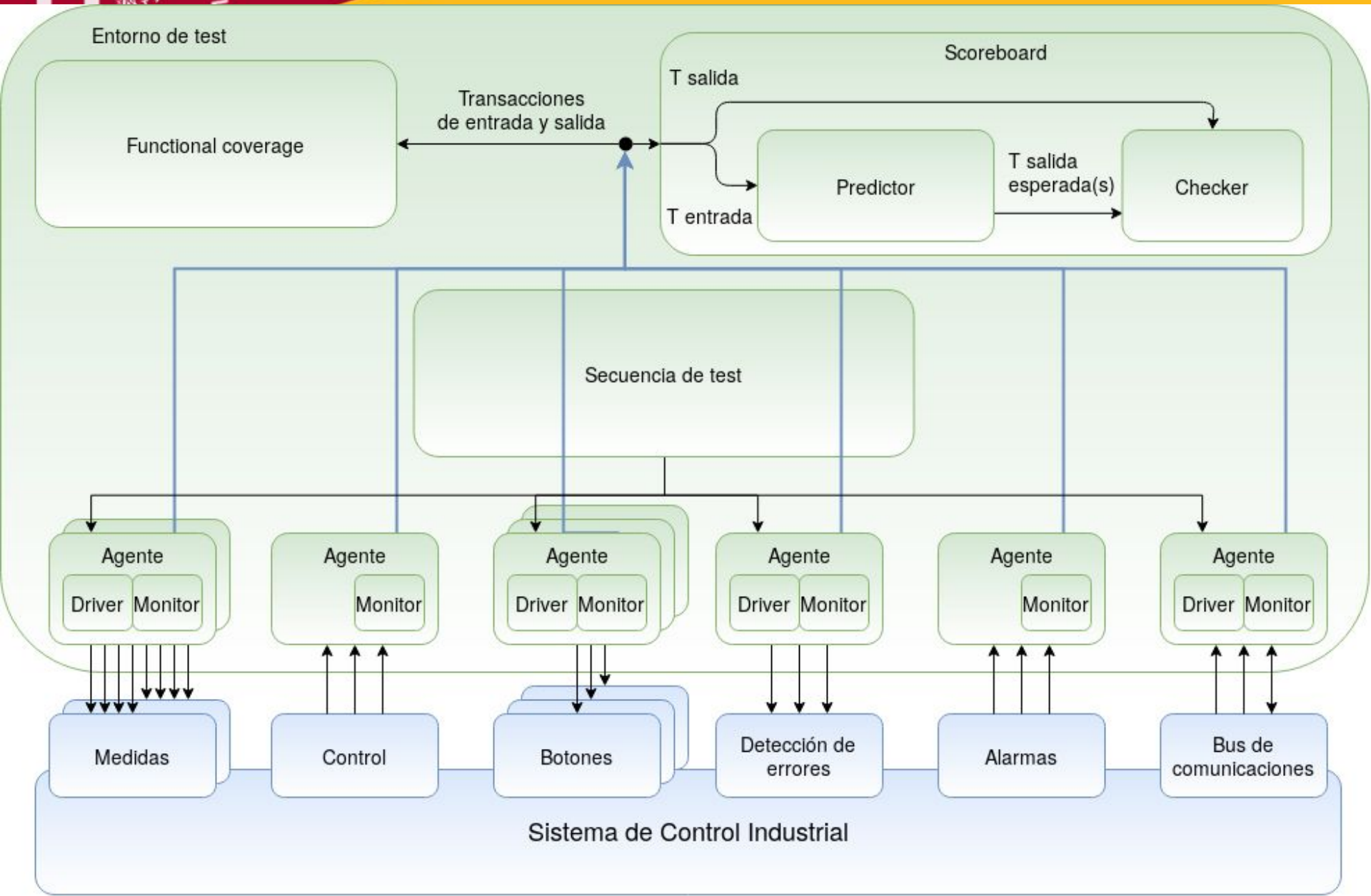
Tests de sistemas complejos



Testbench estructurado

- Un testbench así es lo más complejo que os podéis encontrar en verificación funcional
- Los monitores envían las transacciones de entrada y salida que detectan a los otros bloques (functional coverage, scoreboard)
  - No utilizamos directamente las transacciones que reciben los drivers (el monitor sirve para verificar el driver)
- El grueso del tiempo de desarrollo del testbench lo suele ocupar el desarrollo de los drivers, monitores y el predictor
- La secuencia de test puede cambiarse a través de generics, parameters o incluso ser leída de ficheros
  - De esta forma se pueden hacer tests probando cada interfaz por separado y tests globales con eventos en varios/todos los interfaces a la vez

Tests de sistemas complejos



Contenido

- Motivación
- Funcionalidad del diseño a probar
- Transacciones y operaciones
- Lista de funcionalidades
- Estímulos de entrada
- Respuestas de salida
- Lista de tests
- Tests unitarios
- Tests de sistemas complejos
- Conclusiones
- Bibliografía

Conclusiones

- En proyectos complejos, necesitamos una metodología para poder abarcar los tests del sistema
- Esto es obligatorio en industrias en las que un fallo es crítico (espacio, aviónica, biomédica, nuclear, etc) para pasar los procesos de certificación
- El modelado a nivel de transacción simplifica enormemente la extracción de funcionalidades a verificar
- Esta forma de planificar tests es independiente de cómo se implementen (VHDL, SystemVerilog, CoCoTb, VUnit, etc)

Contenido

- Motivación
- Funcionalidad del diseño a probar
- Transacciones y operaciones
- Lista de funcionalidades
- Estímulos de entrada
- Respuestas de salida
- Lista de tests
- Tests unitarios
- Tests de sistemas complejos
- Conclusiones
- Bibliografía

Bibliografía

- Ray Salemi, *FPGA Simulation: A Complete Step-by-Step Guide.* Boston Light Press, 2009

Resultados de aprendizaje

- Saber utilizar el modelado a nivel de transacción para extraer la funcionalidad a probar de un diseño
- Ser capaz de plantear una lista de tests para un diseño
- Ser capaz de distinguir entre tests unitarios y tests de integración
- Conocer la arquitectura de un test estructurado para sistemas complejos