Verificación de Sistemas en Chip

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

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

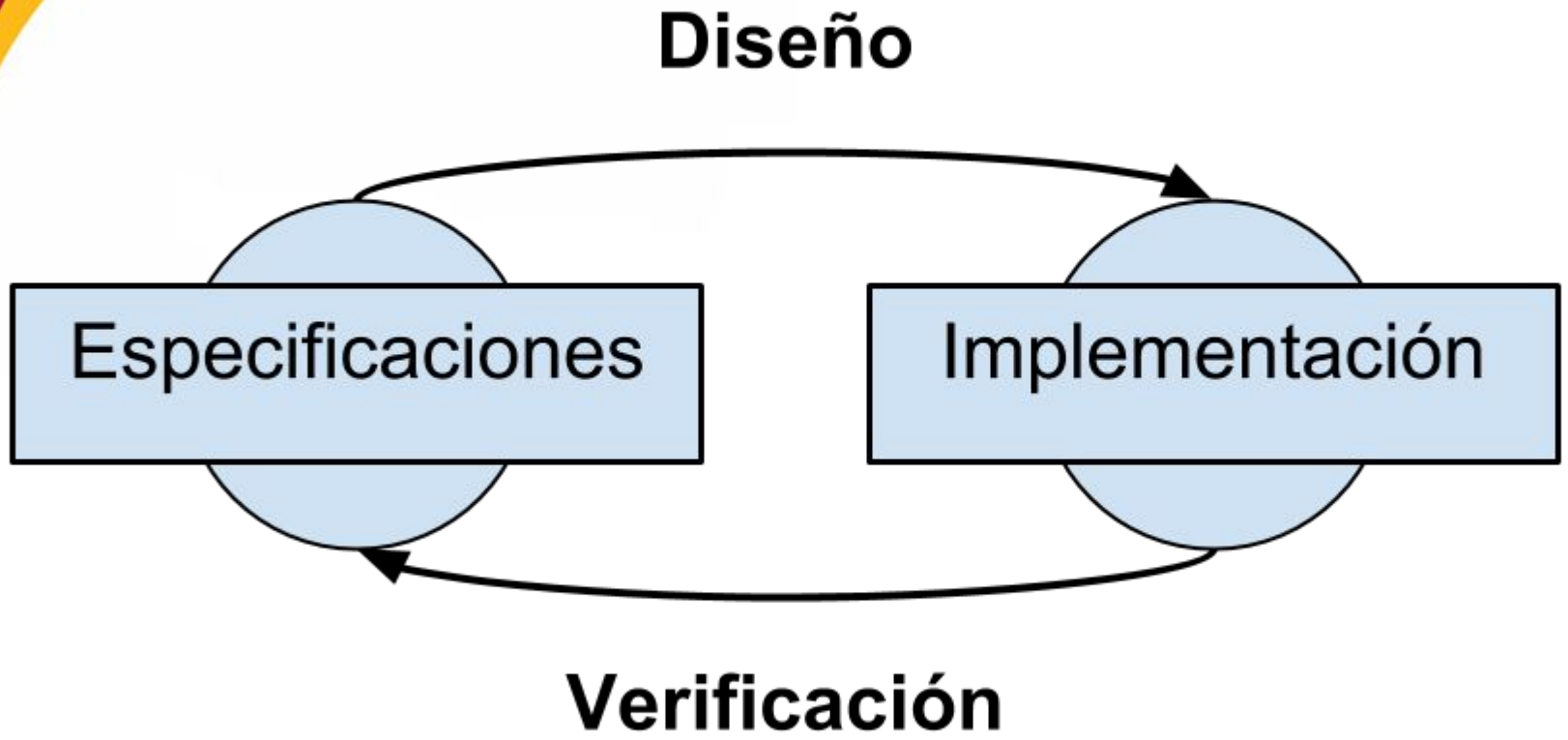
Contenido

- Motivación
- Test tradicional
- Capacidades de verificación
- Metodologías de verificación
- Consideraciones para la asignatura
- Bibliografía

Contenido

- Motivación
- Test tradicional
- Capacidades de verificación
- Metodologías de verificación
- Consideraciones para la asignatura
- Bibliografía

¿Qué es Verificar?



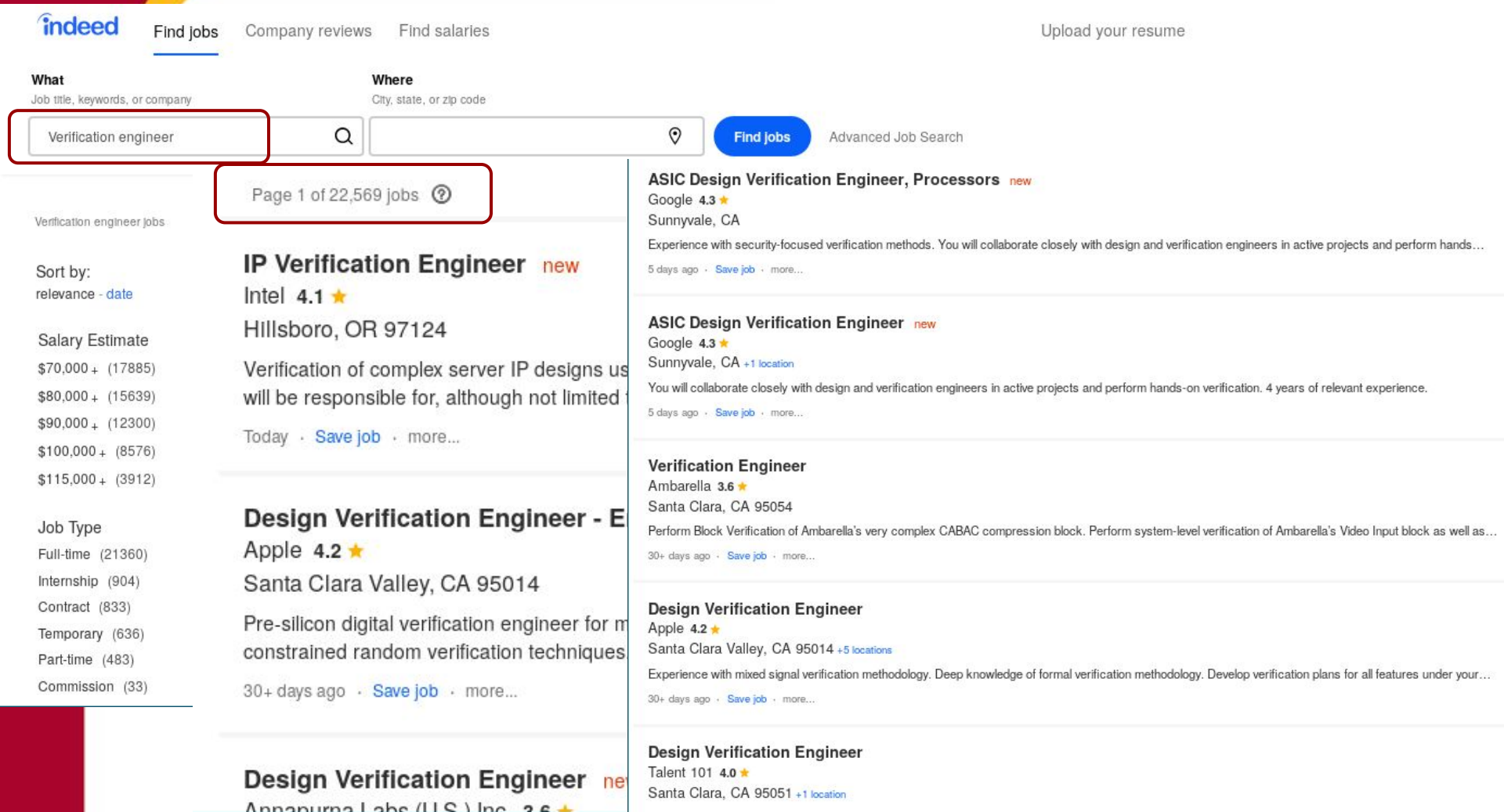
Comprobar que la implementación realizada realmente cumple con las especificaciones

¿Por qué verificar? 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
- Industrias donde los fallos deben evitarse a toda costa: espacio, aeronáutica, biomedicina, nuclear …
- Dificultad de diagnosticar y arreglar fallos sobre el prototipo FPGA
- Terminar (de verdad) los proyectos a tiempo

Motivación

¿Por qué aprender verificación?



Contenido

- Motivación
- Test tradicional
- Capacidades de verificación
- Metodologías de verificación
- Consideraciones para la asignatura
- Bibliografía

Test ‘tradicional’

Típico testbench:

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

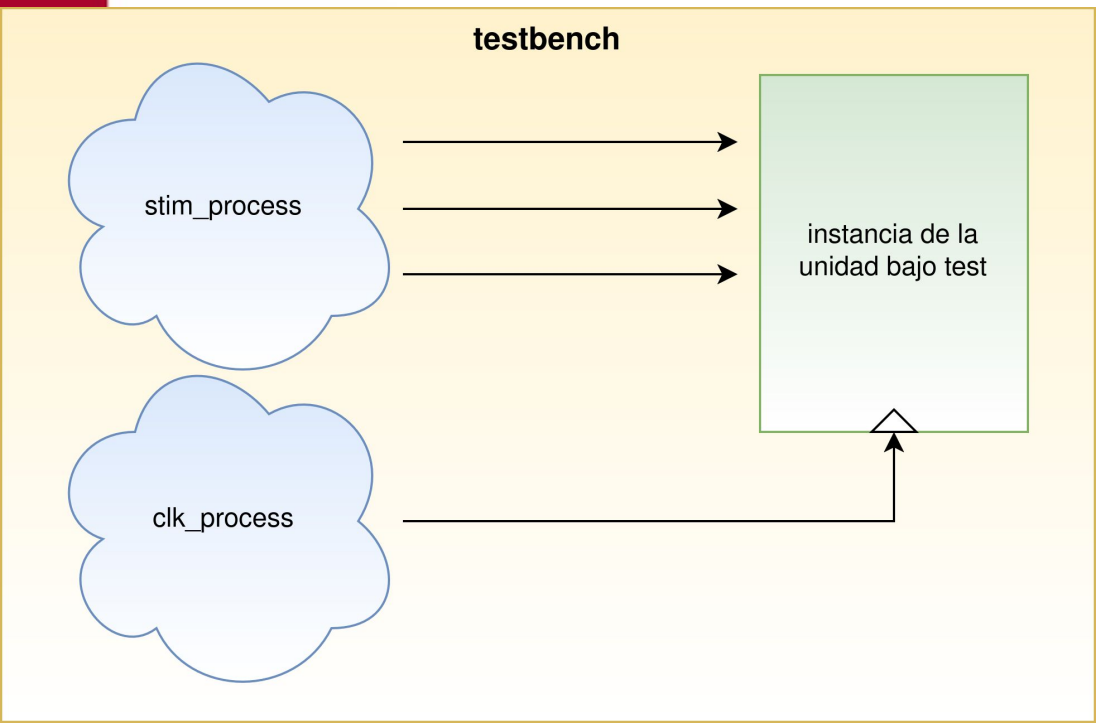
Este enfoque no escala para tests complejos

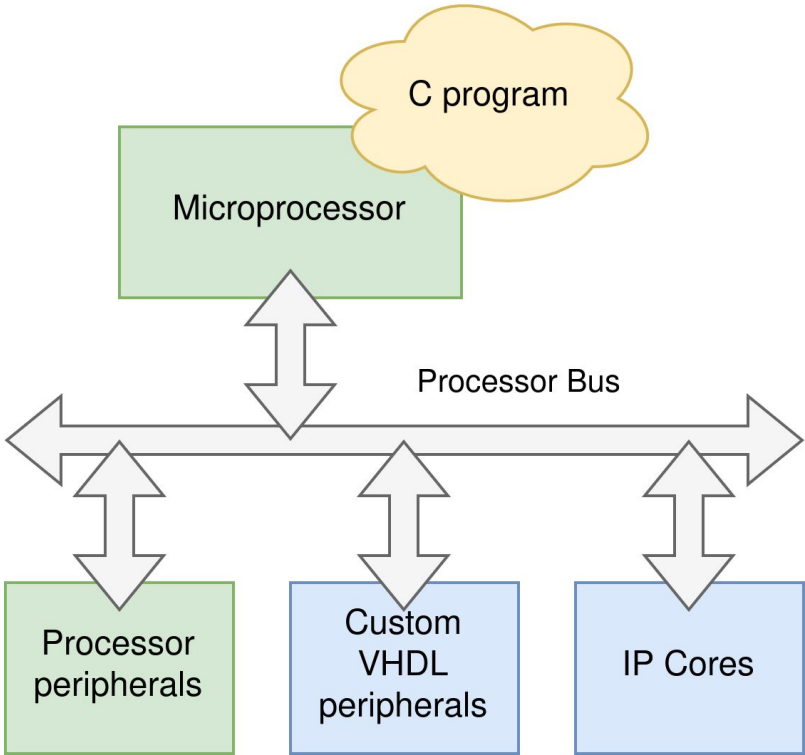
Ej: 200K estímulos y 1M ciclos de reloj de formas de onda que comprobar

Test ‘tradicional’

Difícilmente escalable

Además, un SoC puede tener múltiples interfaces, ¿cómo movemos eso con un único ‘stim_process’?





Contenido

- Motivación
- Test tradicional
- Capacidades de verificación
- Metodologías de verificación
- Consideraciones para la asignatura
- Bibliografía

Capacidades de verificación (I)

- Code coverage
  - Qué porcentaje del código ha sido realmente probado en simulación
- Assertions
  - Comprobar que se cumplen determinadas condiciones, dar un mensaje de error si no
    - Más eficiente que comprobar formas de onda a ojo
- Modelado a nivel de transacción (TLM)
  - Separar los datos que se mueven por los interfaces del movimientos de los pines asociados.
    - El concepto más importante en verificación de los últimos 20-30+ años

Capacidades de verificación (II)

- Testbenches auto-chequeantes
  - Testbenches que devuelven pass/fail sin intervención humana
    - Se pueden automatizar cientos de tests
    - Y ejecutarlos a cada cambio en RTL
- Estímulos aleatorios
  - Probar cada vez con entradas diferentes
- Alcance funcional
  - Relacionar las entradas y salidas que se han visto en la simulación con el plan de pruebas
    - Me he dejado alguna funcionalidad por probar?

Es una técnica automática

- La hace el simulador, compilando los ejecutables de simulación con ciertas opciones
- Soportado por ModelSim/Questa, Aldec, Vivado XSim (parcialmente), GHDL (parcialmente), etc…
- Categorías:
  - Statement (sentencias)
  - Branch (ramificaciones)
  - FSM (estados y transiciones)
  - Expression (posibilidades de `d <= a or (b and c);`)
  - Condition (posibilidades de `if (a or (b and c) then`)

Notifica si una condición no se cumple

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

4 niveles de gravedad:

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

Normalmente sólo para simulación. No impiden la síntesis, pero el sintetizador sólo puede comprobar los que dependen de condiciones estáticas.

Transaction-Level Modeling (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 Modeling (TLM)

Por ejemplo si enviamos datos por una FIFO:

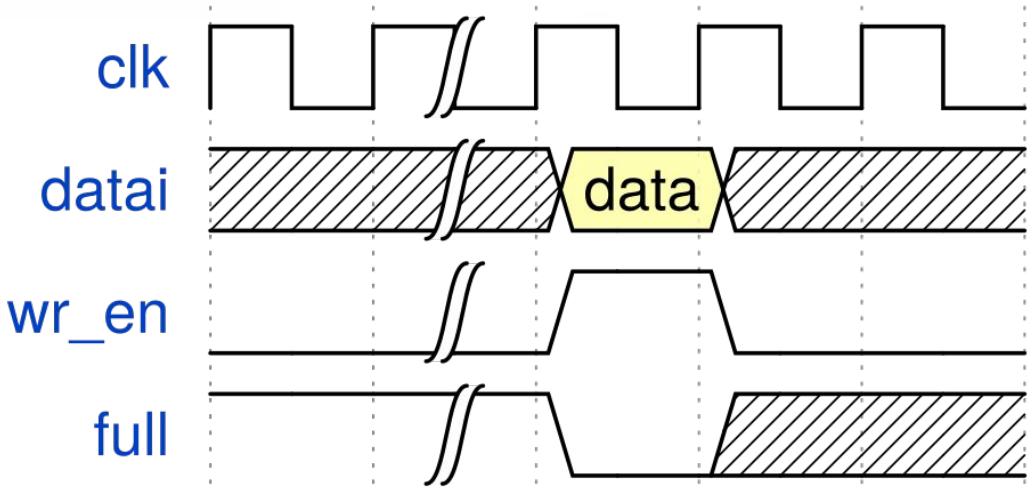
1. Esperamos a que no esté llena
2. Ponemos el dato
3. Activamos write_enable
4. Esperamos un ciclo de reloj
5. Desactivamos write_enable

Queremos separar el envío del dato (fifo_write) del movimiento de pines (full, write_enable, datai)

Transaction-Level Modeling

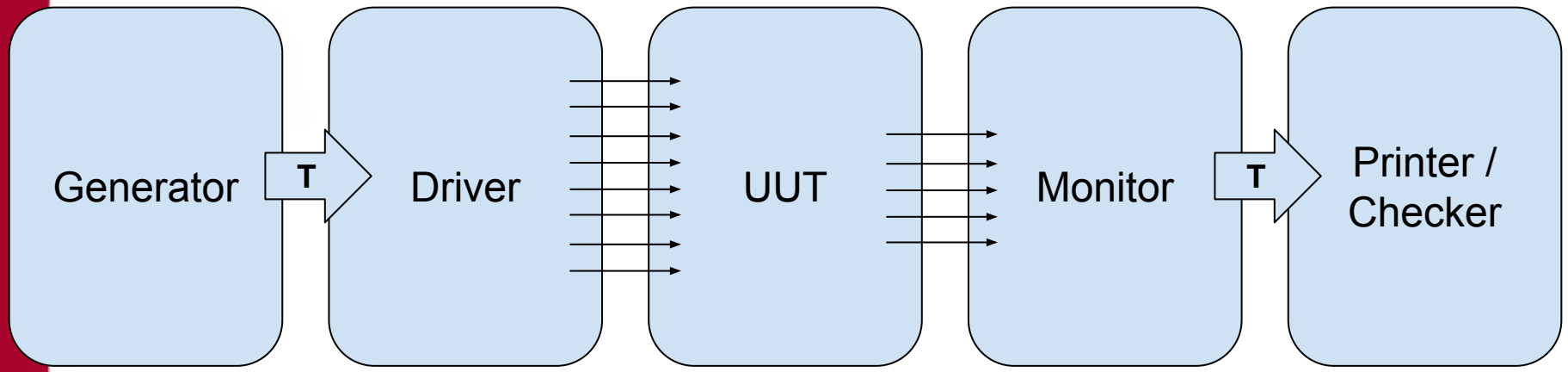
```
fifo_write(data);
```

← `data` se propaga por el interfaz



← movimiento de pines

Arquitectura de un testbench TLM

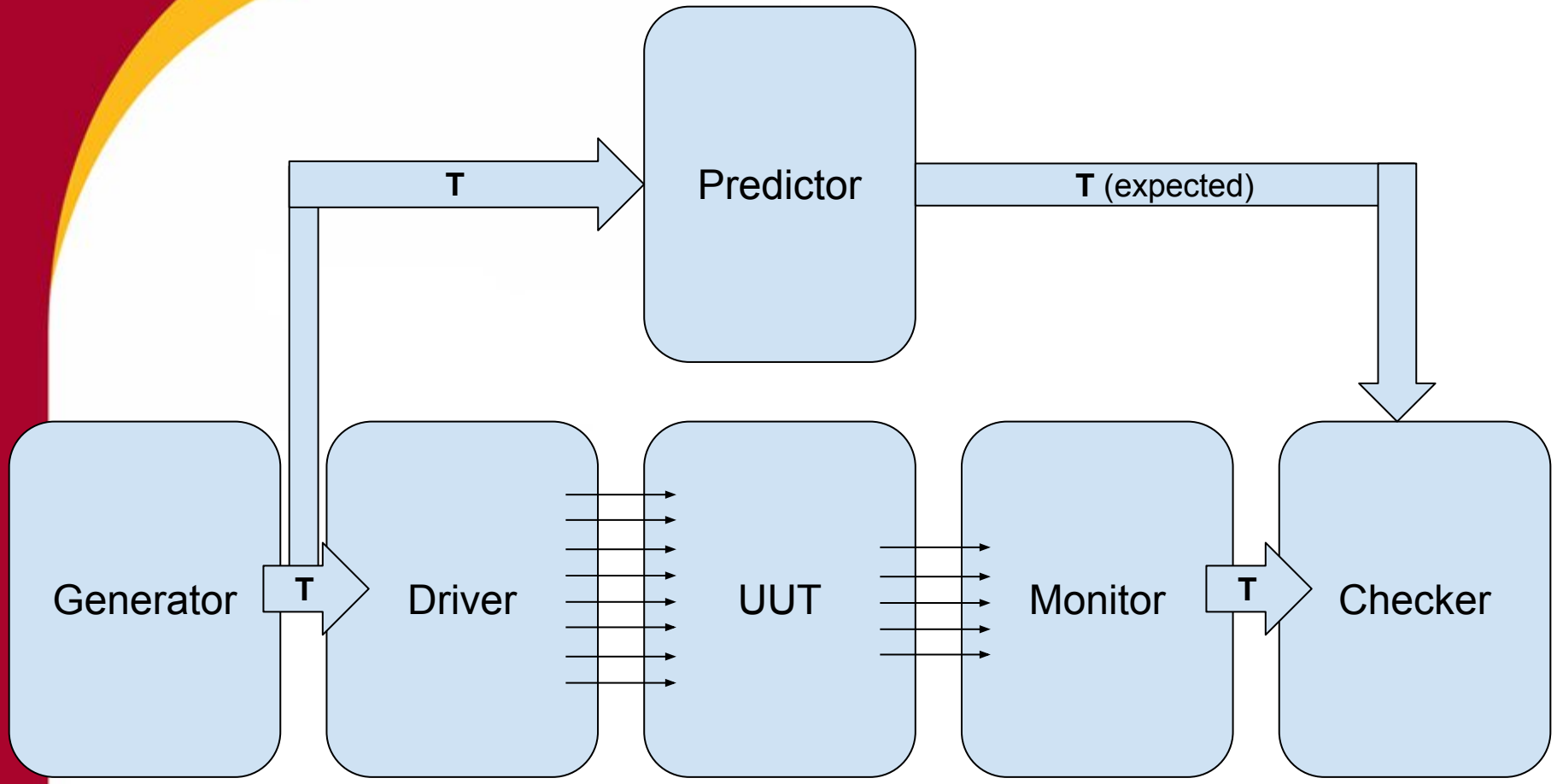


Self-checking Testbenches

- 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

Self-checking Testbenches



Estímulos automáticos

Si tenemos un testbench 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

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*

¿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

- Motivación
- Test tradicional
- Capacidades de verificación
- Metodologías de verificación
- Consideraciones para la asignatura
- Bibliografía

Metodologías de verificación

- Existen en la industria diferentes metodologías de verificación
  - Enfocadas en el ‘code reuse’ y en aplicar las capacidades anteriormente vistas
  - Pero aprenderlas nos requeriría más tiempo del que tenemos disponible en la asignatura
- Muy utilizadas para verificar ASIC y sistemas críticos
- UVM (SystemVerilog)
- OSVVM (VHDL)
- UVVM (VHDL)

Contenido

- Motivación
- Test tradicional
- Capacidades de verificación
- Metodologías de verificación
- Consideraciones para la asignatura
- Bibliografía

Para nuestros diseños

- No podemos simular el SoC completo porque no tenemos un modelo de simulación del ARM
- Pero podemos probar muy bien todos los periféricos que hagamos
- Para ello, necesitaremos generar e interpretar transacciones AXI para poder simular los periféricos
- En la práctica 5 usaremos un AXI VIP (Verification IP) escrito en python para no tener que escribir un driver+monitor desde cero
- El uso de python para verificación nos permite adoptar las capacidades mencionadas anteriormente con menos esfuerzo

Contenido

- Motivación
- Test tradicional
- Capacidades de verificación
- Metodologías de verificación
- Consideraciones para la asignatura
- Bibliografía

Bibliografía

- Ray Salemi, *FPGA Simulation: A Complete Step-by-Step Guide.* Boston Light Press, 2009
- Ray Salemi, 'Evolving FPGA verification capabilities', disponible en www.verificationacademy.com

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