

Metodologías de diseño digital avanzado

Hipólito Guzmán Miranda
Profesor Contratado Doctor
Universidad de Sevilla

Contexto

- 50 años de Ley de Moore (1965)
- El 'Design Gap'
- El 'Verification Gap'

Pero antes de todo esto...

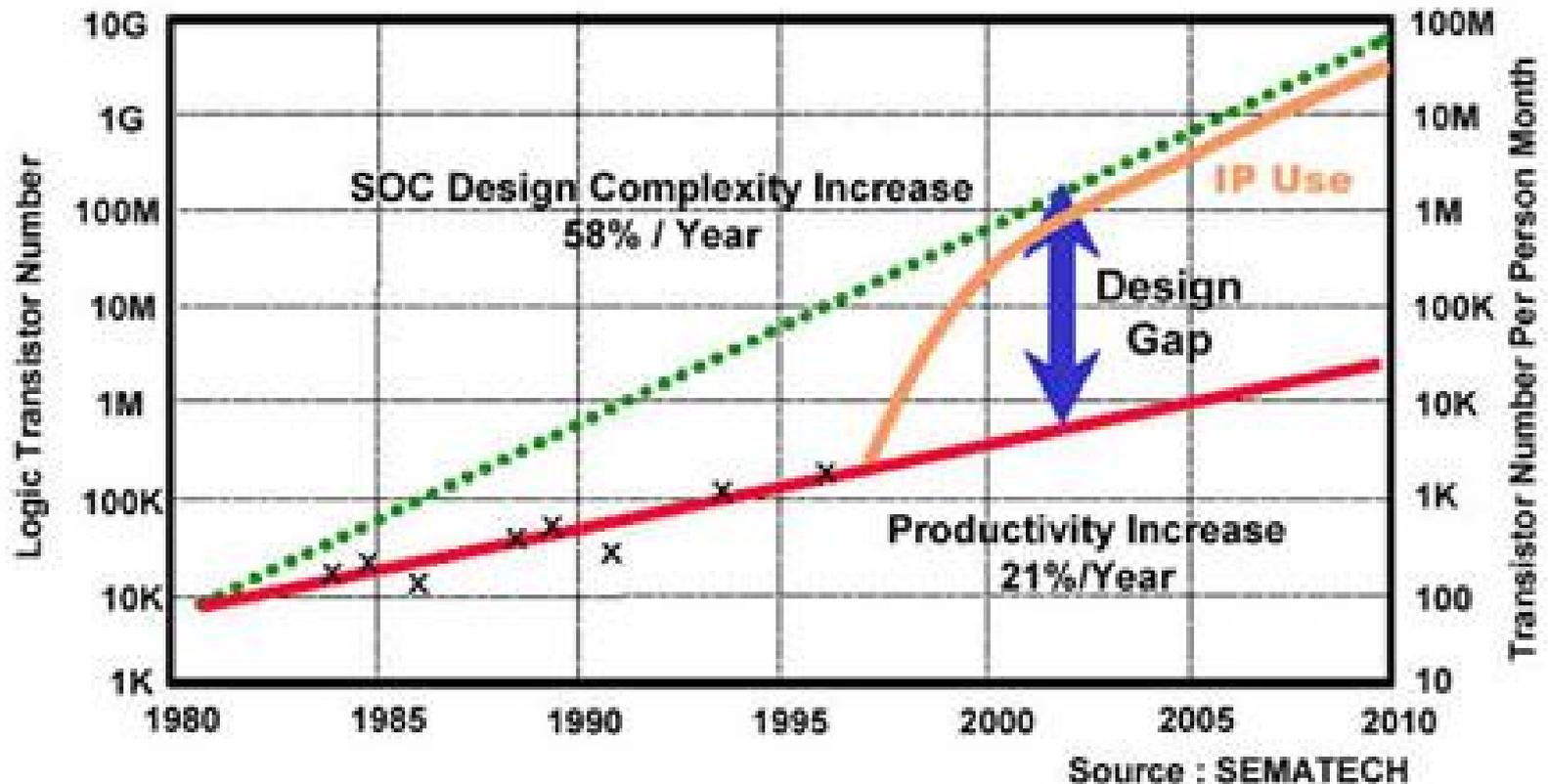
¿Sabéis cómo es una FPGA por dentro?

Sistemas digitales complejos

Escalado de complejidad:

- Design gap
- Verification gap

El 'Design Gap'

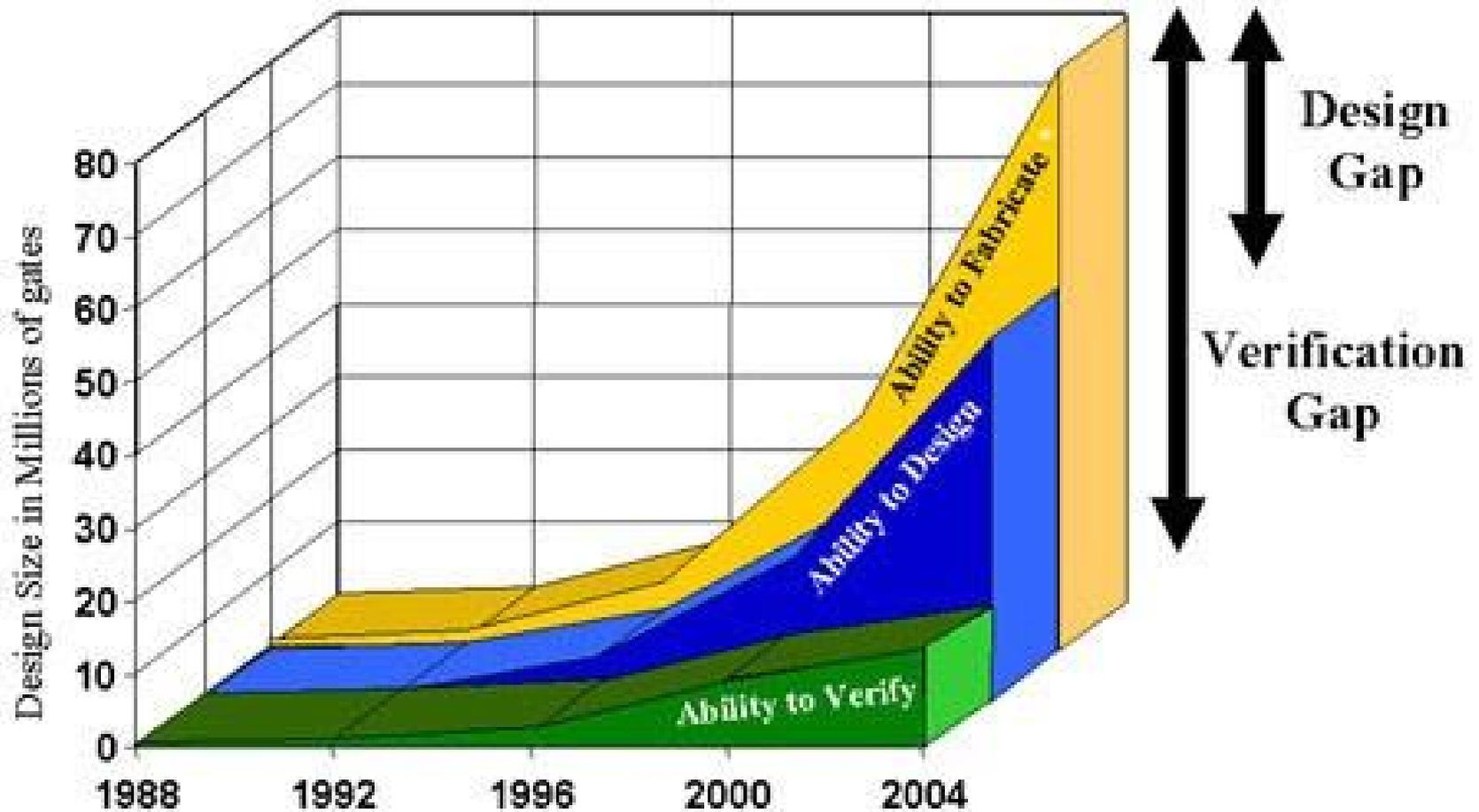


El 'Design Gap'

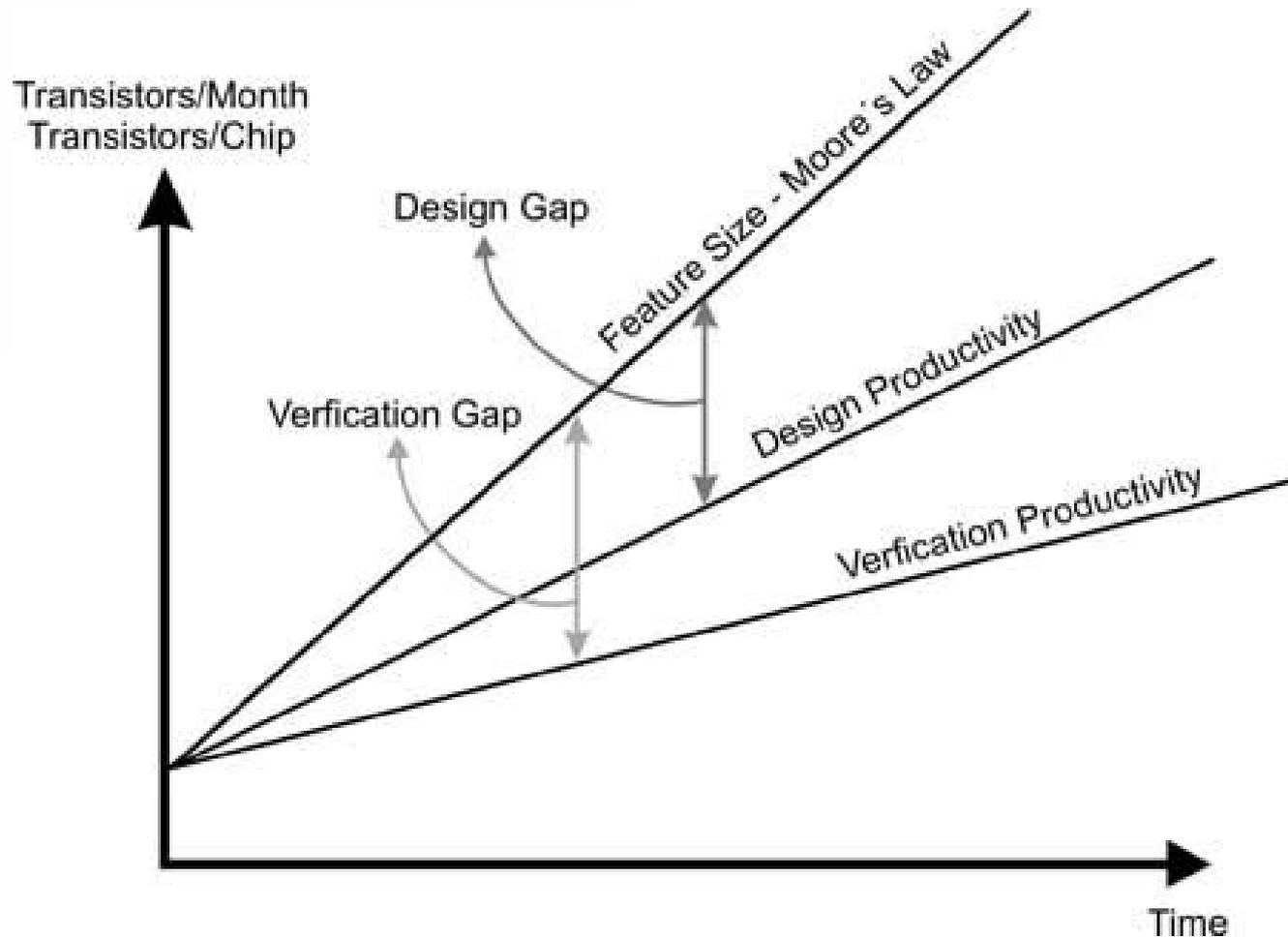
- La capacidad de diseño crece más lento que la capacidad de fabricación
- Si un diseñador diseña a 100 puertas por día, y en un chip caben 10M puertas... tardamos 100K días en diseñar el sistema completo : 500 ingenieros * 1 año

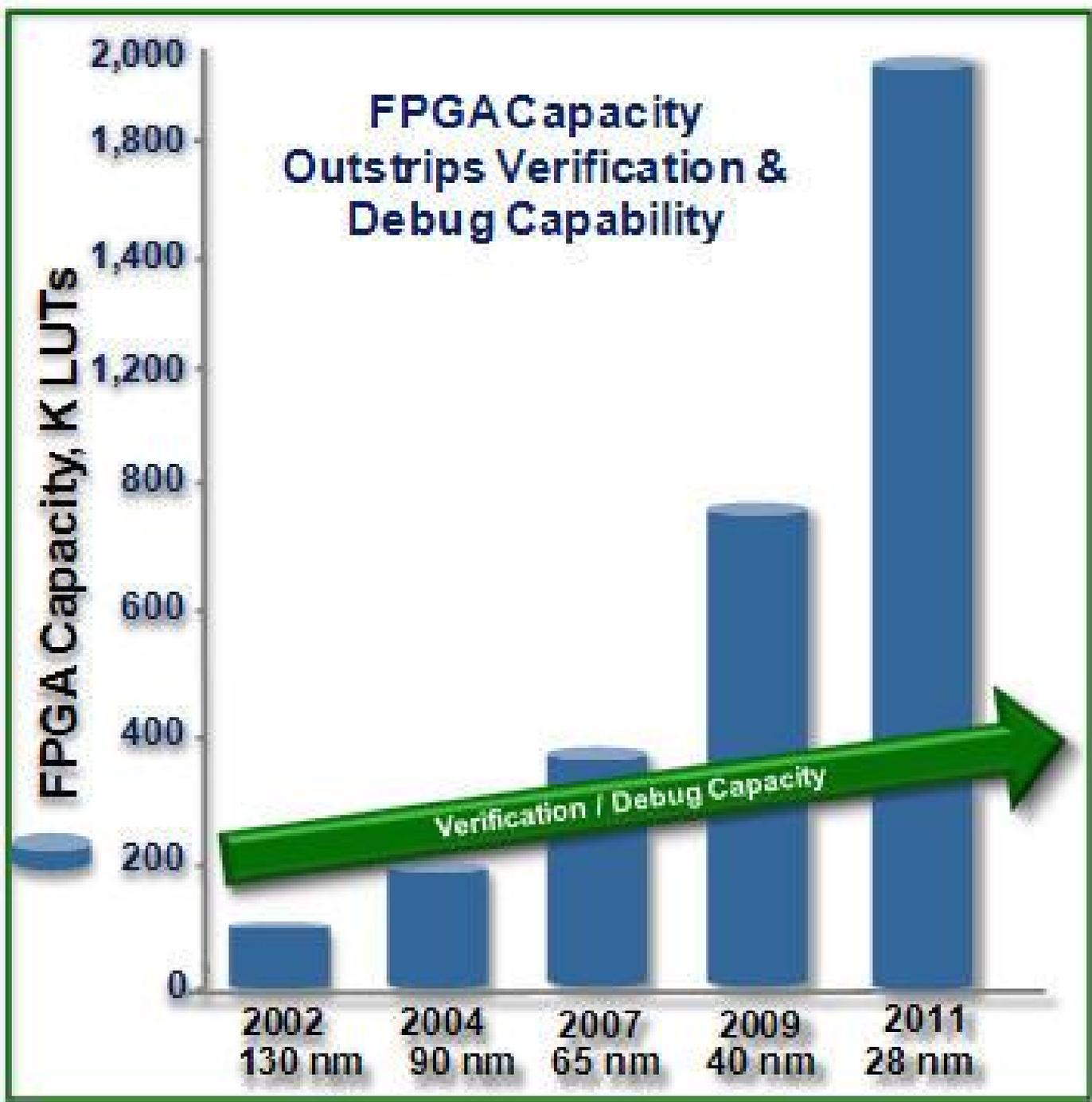
El 'Verification Gap'

El 'Verification Gap'



El 'Verification Gap'





Desarrollo de sistemas digitales complejos

- Especificaciones
 - Describir de forma clara y no ambigua qué se quiere hacer
- Diseño
 - Implementar el sistema
- Verificación
 - Comprobar que el diseño cumple con las especificaciones

Retos en especificación

- Especificar de manera no ambigua
- “Moving targets”: cambios que pide el cliente durante la duración del proyecto
- Incremento de la complejidad de los sistemas -> incremento exponencial de las interacciones entre elementos y modos de fallo
- Mantener el documento de requisitos vivo durante la duración del proyecto

Retos en diseño

- Altos requerimientos en prestaciones
 - Alto throughput
 - Alto ancho de banda
 - Alta frecuencia de operación
- Conocimientos técnicos de los equipos
- Protocolos y primitivas específicas
- Diseños mixtos Hardware/Software

Retos en verificación

- Verificación funcional: ¿es la funcionalidad correcta:
 - En todos los casos de uso posibles?
 - Para todas las configuraciones posibles?
- Coverage: ¿hemos probado todo nuestro diseño?
 - ¿Todo el código?
 - ¿Toda la funcionalidad?
- ¿Podemos hacer esto en proyectos de complejidad exponencialmente creciente?

El 'Design Gap': soluciones

- Uso de soft processors
- Uso de IP cores
- Síntesis de alto nivel

Ojo! Diseños más complejos implica mayor esfuerzo en **verificación!**

El 'Verification Gap': soluciones

- Métricas de verificación (¿cuándo sé que he terminado de verificar?)
- Metodologías de verificación (OVM, OSVVM, UVM)
- Uso de Verification IP
- Uso de aceleradores hardware



HAPS-54
Virtex-5 LX330
(May-2007)

Metodologías de diseño

- FPGAs como System-on-Chip: Soft processors y hard macros
- Diseño con IP cores
- Higher Level Synthesis

Metodologías de diseño

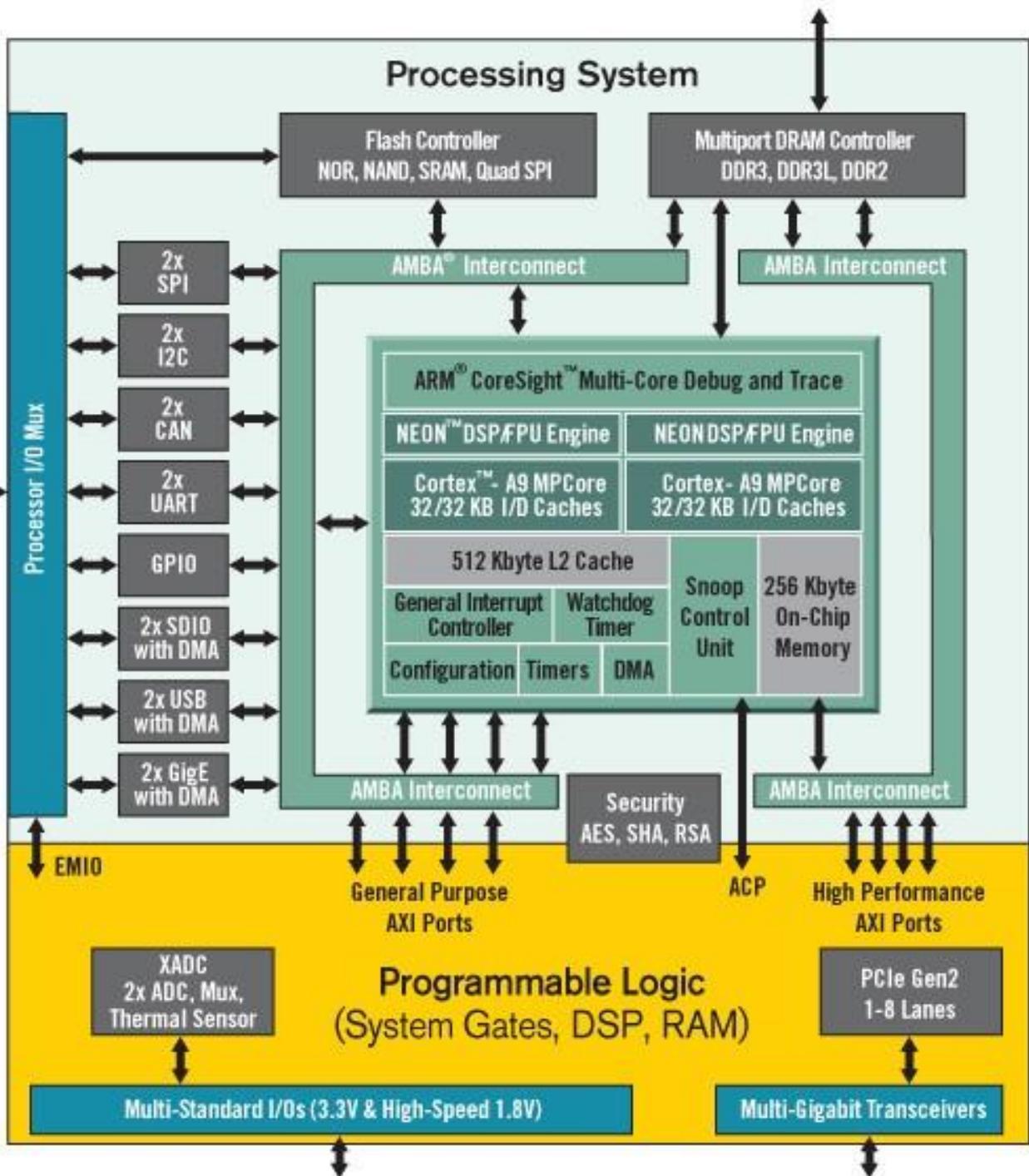
- **FPGAs como System-on-Chip: Soft processors y hard macros**
- **Diseño con IP cores**
- **Higher Level Synthesis**

FPGAs como System-on-Chip

- Evolución de las tecnologías microelectrónicas (Ley de Moore)
- Nos lleva a la siguiente arquitectura de una FPGA moderna:

Arquitectura de una FPGA moderna

- Además de IOBs, CLBs y recursos de rutado:
- Memorias empotradas (Block RAMs)
- Conexiones de alta velocidad (Gigabit transceivers, PCIeexpress, ...)
- Microprocesadores!



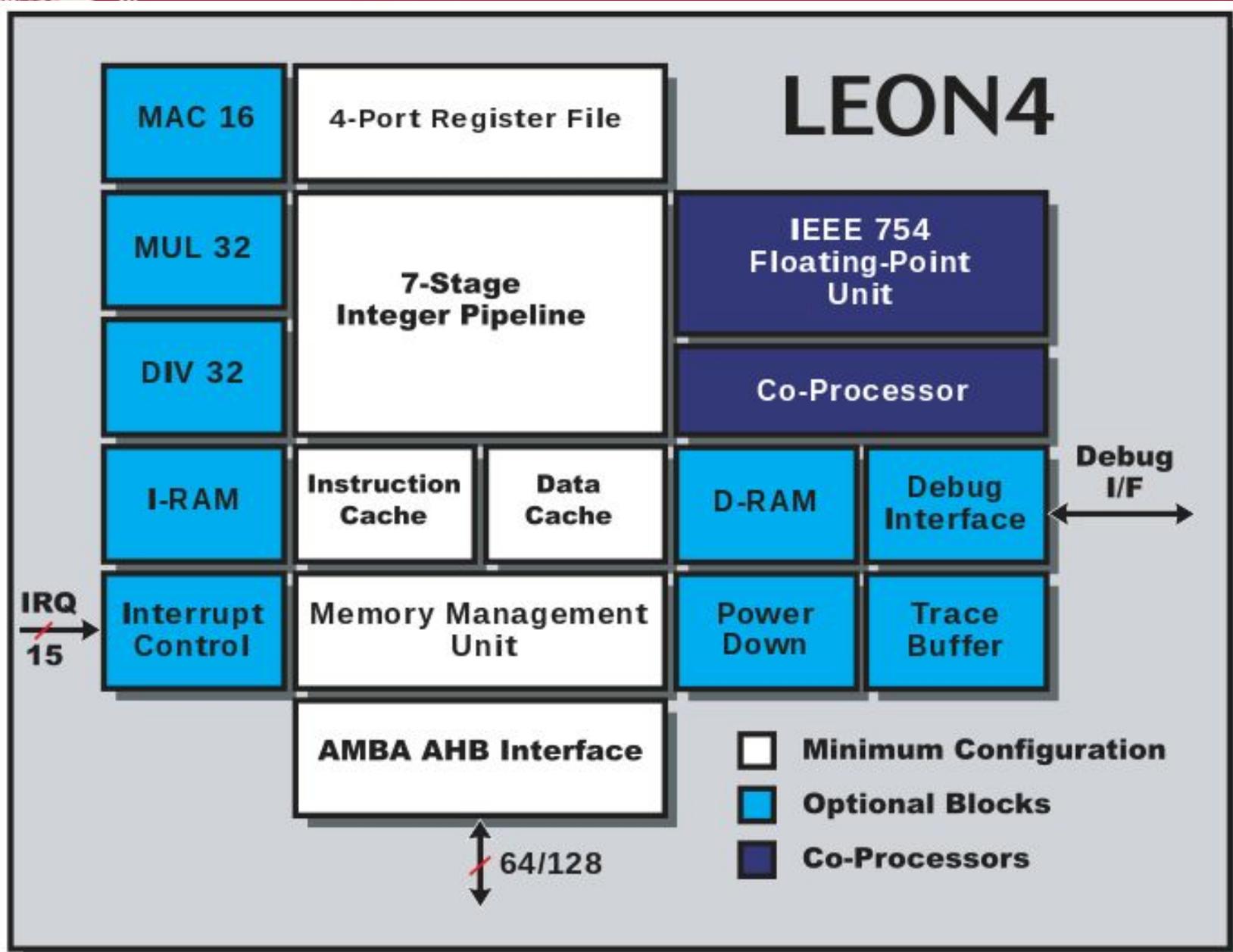
Familia Zynq

ARM
Dual-core
Integrados en
el silicio

Soft processors

- No necesitamos una FPGA de última generación para tener un microprocesador
- Si tuviéramos una descripción en VHDL/Verilog de una ALU, registros, contador de programa, decodificador de instrucciones... = **un microprocesador**
- Aunque no esté implementado en silicio (como una BRAM), podríamos dedicar una parte de la FPGA a implementar un microprocesador

Soft Processors



No necesitamos crearlo desde cero

Algunos soft processors:

- Microblaze (Xilinx)
- Nios II (Altera)
- Leon 4 (Aeroflex Gaisler)
- Plasma (Opencores)
- OpenSparc
- OpenRisc
- ...

Algunos problemas:

- Configurar los periféricos y mapa de memoria
- Disponer de un toolchain completo (compilador, linker, etc)
- Sistema operativo o programa 'standalone'
- Configurar las BRAM con el ejecutable
- Desarrollo de periféricos 'custom'
- Disponer de un modelo de simulación
- Conseguir que arranque el micro!

Algunas ventajas:

- Diseño óptimo: HW y SW se encargan cada uno de lo que les es más eficiente
- Simplicidad en las comunicaciones de tu diseño HDL con el exterior: USB, TCP/IP, ...
- Una actualización no es sólo cambiar el programa: podemos añadir periféricos nuevos (por ejemplo un timer)

Consejos

- HW para tareas paralelas
- SW para tareas secuenciales
- Utilizar HW para funcionalidad crítica: si se cuelga el SW, el HW seguirá funcionando
- Utilizar SW para comunicaciones (TCP/IP): conexión de tu VHDL al exterior

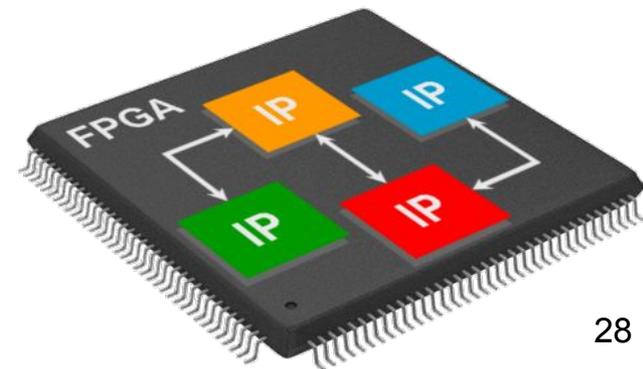
Metodologías de diseño

Además de dominar VHDL avanzado:

- FPGAs como System-on-Chip: Soft processors y hard macros
- **Diseño con IP cores**
- Higher Level Synthesis

Diseño con IP cores

- Esfuerzo evelado de desarrollar un sistema complejo
- Reutilizar módulos que ya estén probados
- Reducción del esfuerzo de diseño
- Principal problema es la **integración** de los módulos:
 - Interfaces
 - Calidad de la documentación
 - Configuración de los IP cores



IP (Intellectual Property) Cores deben ser:

- Reusables
- Configurables
- Simulables con los simuladores estándares de la industria
- Con interfaces basadas en estándares
- Verificados con un alto nivel de confianza
- Completamente documentados

Buses para microprocesadores empotrados

IP cores específicos tienen interfaces específicos. Un softcore específico soportará uno o varios buses:

- PLB, AXI4 (MicroBlaze)
- Wishbone (estándar en Opencores)
- AMBA (Leon)

Buses para microprocesadores empotrados

Los IP cores de terceros que integremos y los que desarrollaremos nosotros deberían tener un interfaz con el bus elegido

Aunque si no necesitamos velocidad podremos utilizar una conexión por GPIO

De esta forma: diseño de System-On-Chip complejos utilizando IP cores basados en estándares

Consejos

- Un buen integrador acelera el diseño tanto o más que un buen diseñador
- Hay que entender los 'quirks' de los fabricantes/proveedores
- Es extremadamente recomendable simular casos básicos para hacerse a los bloques
- Leer mucho y probar poco a poco!!

Metodologías de diseño

- FPGAs como System-on-Chip: Soft processors y hard macros
- Diseño con IP cores
- **Higher Level Synthesis**

Higher Level Synthesis

- VHDL ya es 'alto nivel', no obstante en la actualidad existen herramientas (en distintos estados de madurez) que traducen de código de más alto nivel (C, SystemC) a hardware

Higher Level Synthesis

Code

```

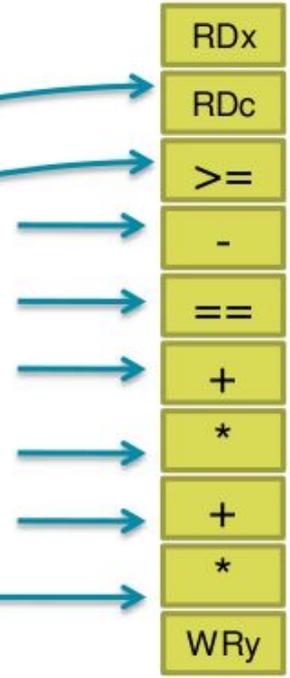
void fir (
  data_t *y,
  coef_t c[4],
  data_t x
){

  static data_t shift_reg[4];
  acc_t acc;
  int i;

  acc=0;
  loop: for (i=3;i>=0;i--) {
    if (i==0) {
      acc+=x*c[0];
      shift_reg[0]=x;
    } else {
      shift_reg[i]=shift_reg[i-1];
      acc+=shift_reg[i]*c[i];
    }
  }
  *y=acc;
}
  
```

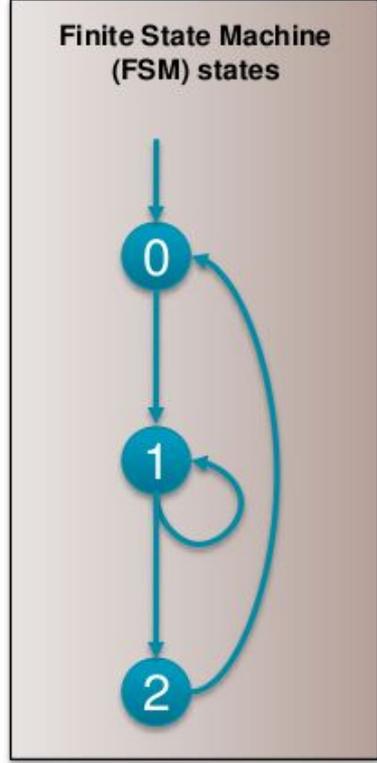
From any C code example ..

Operations



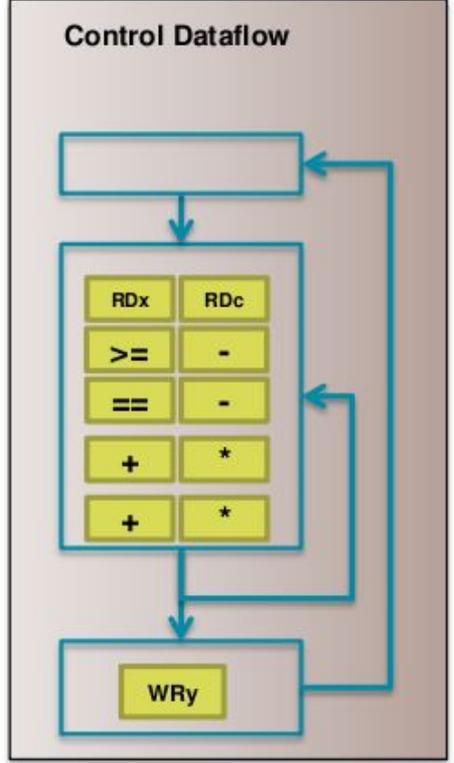
Operations are extracted...

Control Behavior



The control is known

Control & Datapath Behavior



A unified control datapath behavior is created.

Funciones: representan la jerarquía del diseño

Entradas top-level: los argumentos de la función top-level determinan los puertos del hardware generado

Tipos: los tipos de los datos tienen influencia en el área y prestaciones

Arrays: pueden influir en la E/S del dispositivo y convertirse en cuellos de botella

Operadores: Los operadores en el código C se implementan en hardware y pueden ser compartidos por diferentes partes de la implementación

```

void fir (
    data_t *y,
    coef_t c[4],
    data_t x
){
    static data_t shift_reg[4];
    acc_t acc;
    int i;
    acc=0;
    loop: for (i=3;i>=0;i--) {
        if (i==0) {
            acc+=x*c[0];
            shift_reg[0]=x;
        } else {
            shift_reg[i]=shift_reg[i-1];
            acc+=shift_reg[i] * c[i];
        }
    }
    *y=acc;
}
    
```

Higher Level Synthesis

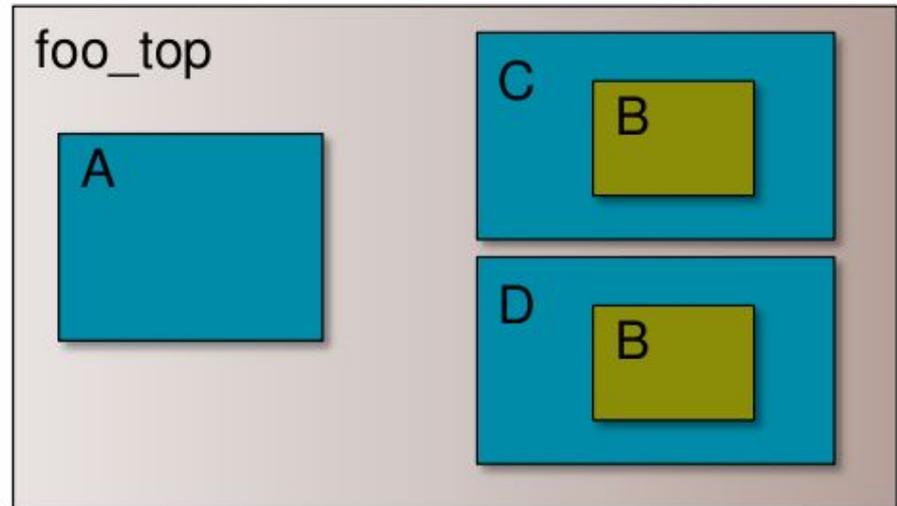
Source Code

```
void A() { ..body A..}  
void B() { ..body B..}  
void C() {  
    B();  
}  
void D() {  
    B();  
}  
  
void foo_top() {  
    A(...);  
    C(...);  
    D(...)  
}
```

my_code.c



RTL hierarchy



Each block can be shared like any other component provided it's not in use at the same time

Consejos

- Al igual que cuando usas un compilador, tienes mayor facilidad de diseño pero menor control sobre el resultado final
- El correcto uso de las directivas de síntesis es fundamental para garantizar una implementación eficiente
- 2x area, $\frac{1}{2}$ velocidad con respecto a implementaciones VHDL

Referencias

- [Xilinx Large FPGA Methodology Guide](#)
(UG872)
- [MicroBlaze Processor Reference Guide](#)
(UG081, v11.0 - EDK 12.1)
- [EDK Concepts, Tools, and Techniques: A Hands-On Guide to Effective Embedded System Design](#) (UG683 - EDK 12.1)

Referencias (II)

- Xilinx Vivado Design Suite User Guide: [High Level Synthesis](#)
- Michael Keating, Pierre Bricaud, [Reuse Methodology Manual for System-on-a-Chip Designs](#)