Tema 2: Arquitectura de FPGAs avanzadas

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

Contexto docente

BT01: Componentes y sistemas digitales para comunicaciones

- Tema 1: Introducción al diseño digital de sistemas de comunicaciones
- Tema 2: Arquitectura de FPGAs avanzadas

Conocimientos previos requeridos:

- Electrónica digital básica
- Electrónica analógica básica

Objetivos de aprendizaje

- Conocer la arquitectura interna de las FPGAs
- Saber qué es el “design gap” y cómo puede mitigarse
- Conocer las arquitecturas tipo System-on-Chip
- Distinguir entre un microprocesador soft-core y hard macro

Repaso

Diseño digital de sistemas de comunicaciones:

- Diseño de bloques digitales de procesado de señal
- Trabajo de curso
- Throughput y latencia
- Pipeline
- Optimización en tiempo y área
- Compartición de recursos

Contenido

- Arquitectura de una FPGA
- Bloques constituyentes
- El ‘Design Gap’
- FPGAs como System-on-Chip
- Procesadores soft-core y hard macro
- Diseño con IP cores
- Conclusiones
- Bibliografía

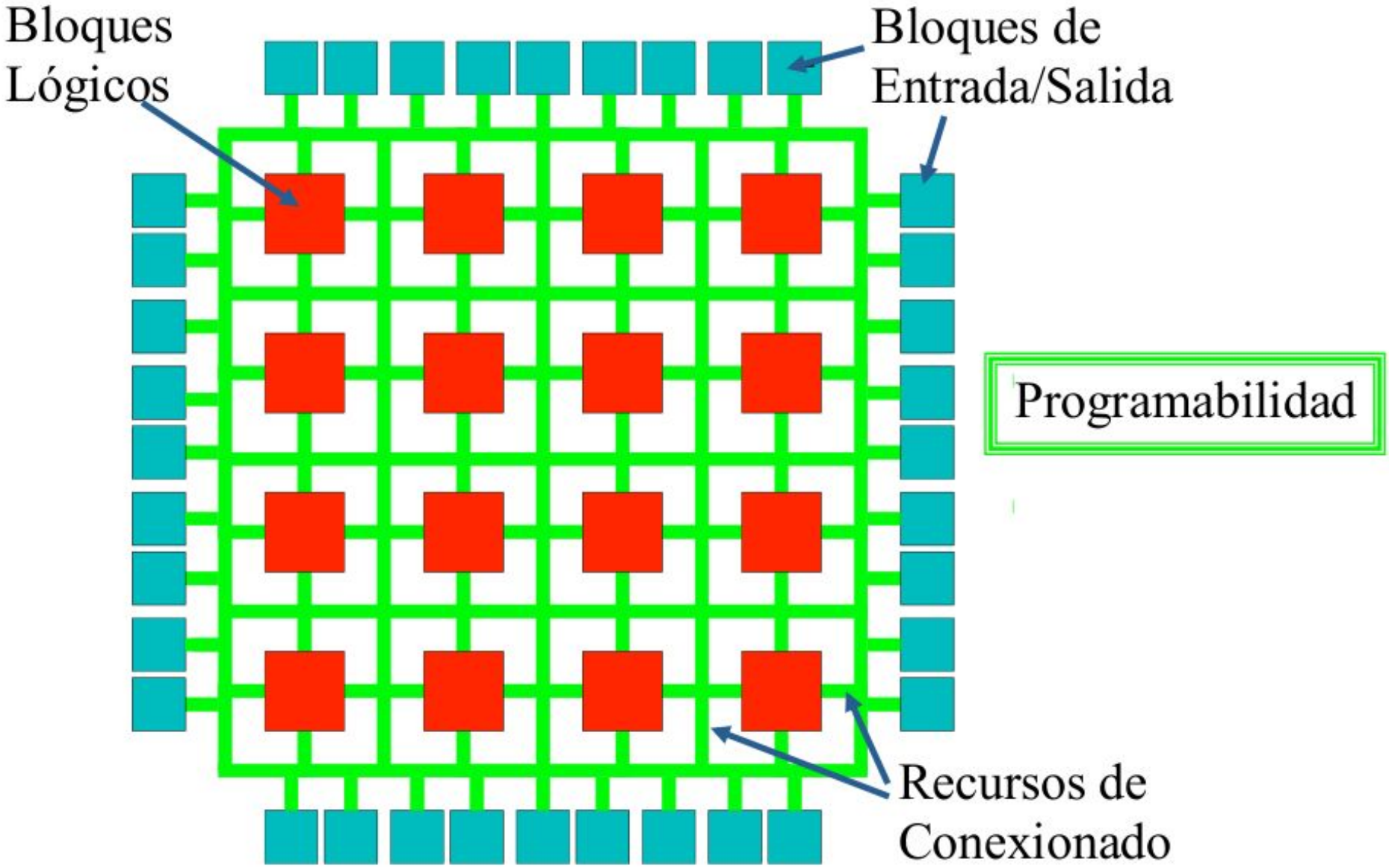
Contenido

- Arquitectura de una FPGA
- Bloques constituyentes
- El ‘Design Gap’
- FPGAs como System-on-Chip
- Procesadores soft-core y hard macro
- Diseño con IP cores
- Conclusiones
- Bibliografía

Arquitecura interna de una FPGA

- IOBs: In/Out Blocks (Bloques de Entrada/Salida)
- CLBs: Configurable Logic Blocks (Bloques Lógicos Configurables)
- Routing Resources (Recursos de Conexionado)
- (Re)Programabilidad

Arquitectura de una FPGA



Contenido

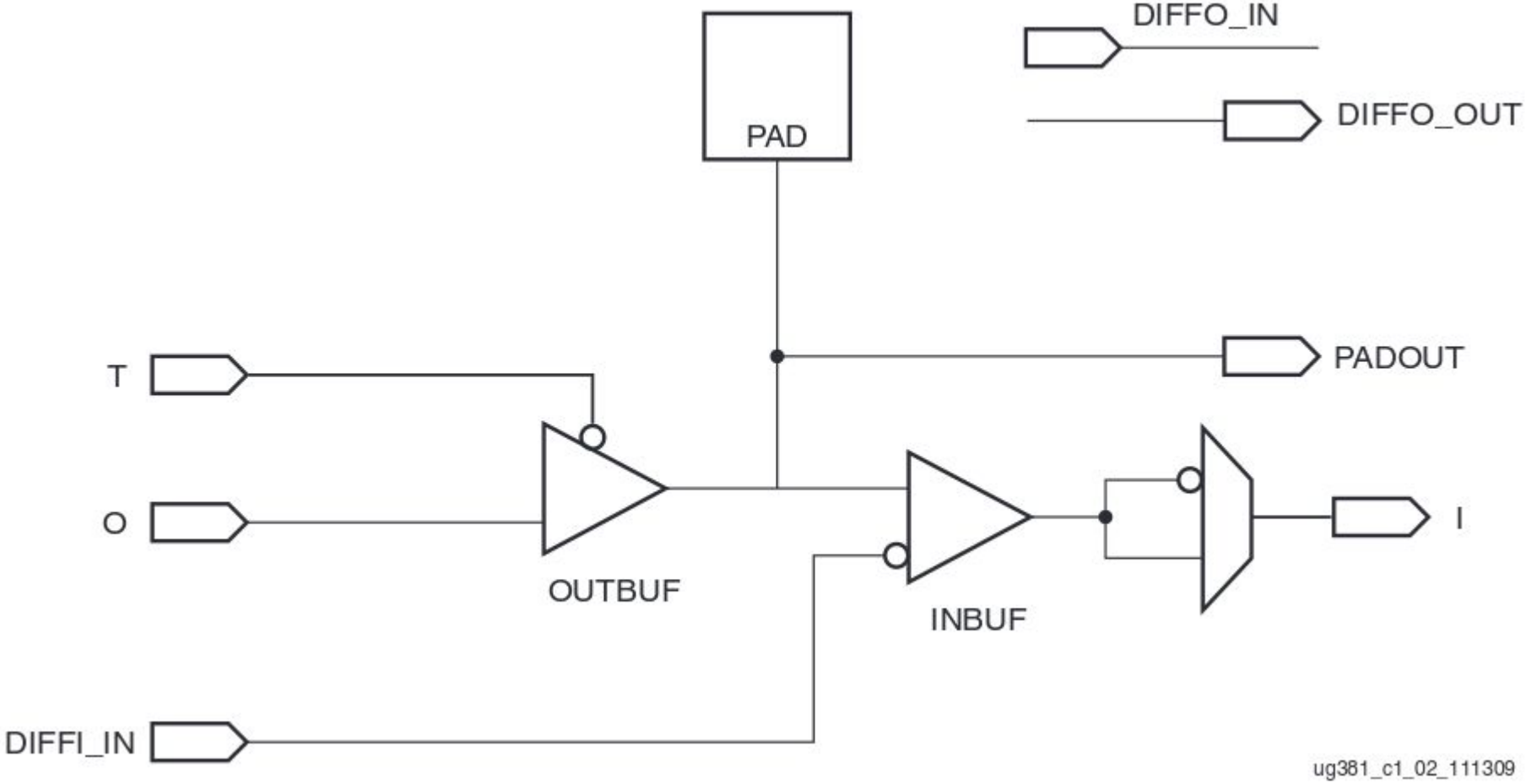
- Arquitectura de una FPGA
- Bloques constituyentes
- El ‘Design Gap’
- FPGAs como System-on-Chip
- Procesadores soft-core y hard macro
- Diseño con IP cores
- Conclusiones
- Bibliografía

In-Out Blocks (IOBs)

- PAD (conexión al exterior)
- Buffer de entrada
- Buffer de salida (triestado)
- Soporte para entradas/salidas diferenciales (dependiendo de la familia)

También llamados I/Os, dependiendo del fabricante

IOB Spartan-6



Configurable Logic Blocks (CLBs)

Compuestas por:

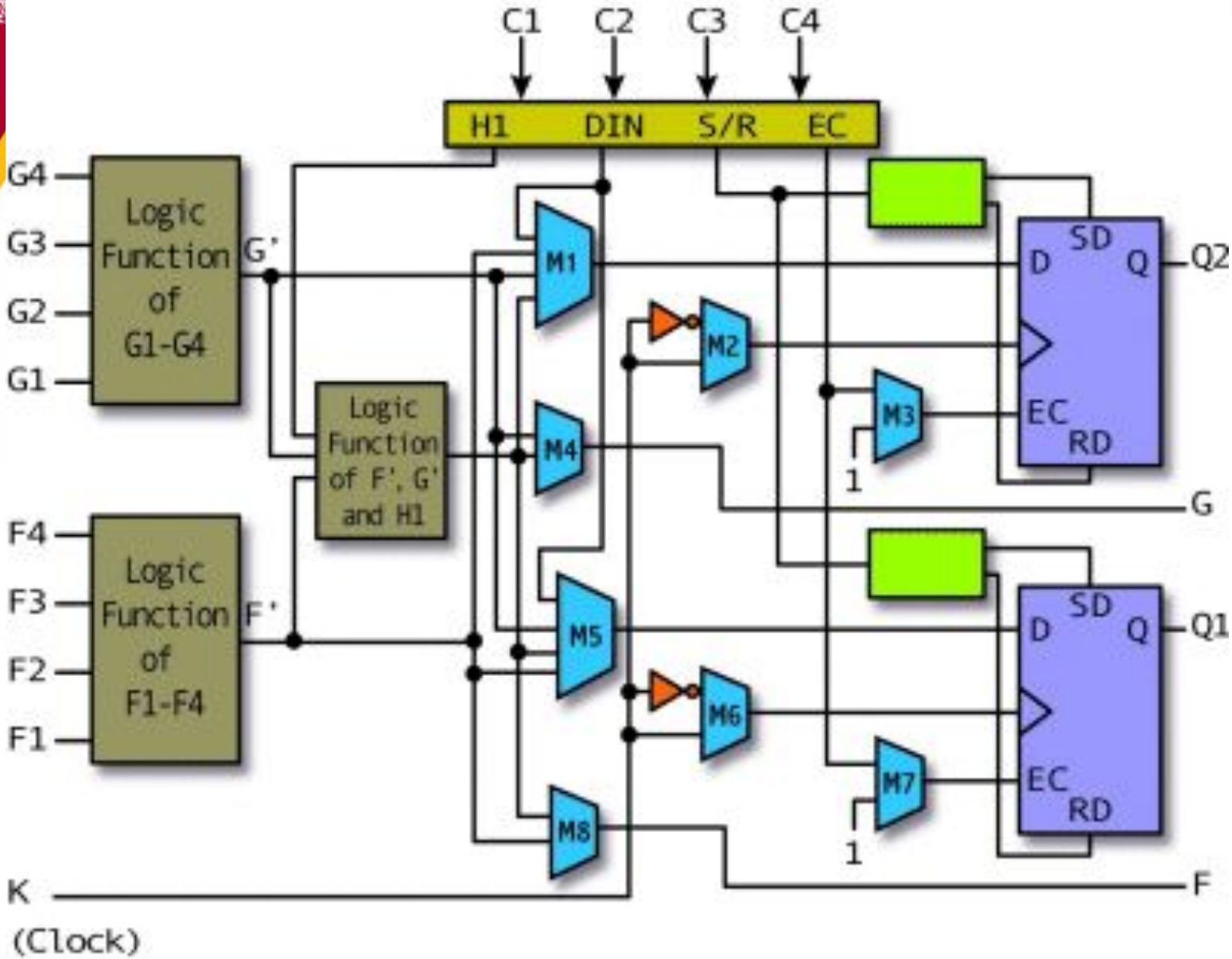
- K * N-input LUT (Look-Up Tables de N entradas)
- K * Flip-flops configurabes

K = 2 en tecnologías antiguas, 4+ en tecnologías modernas.

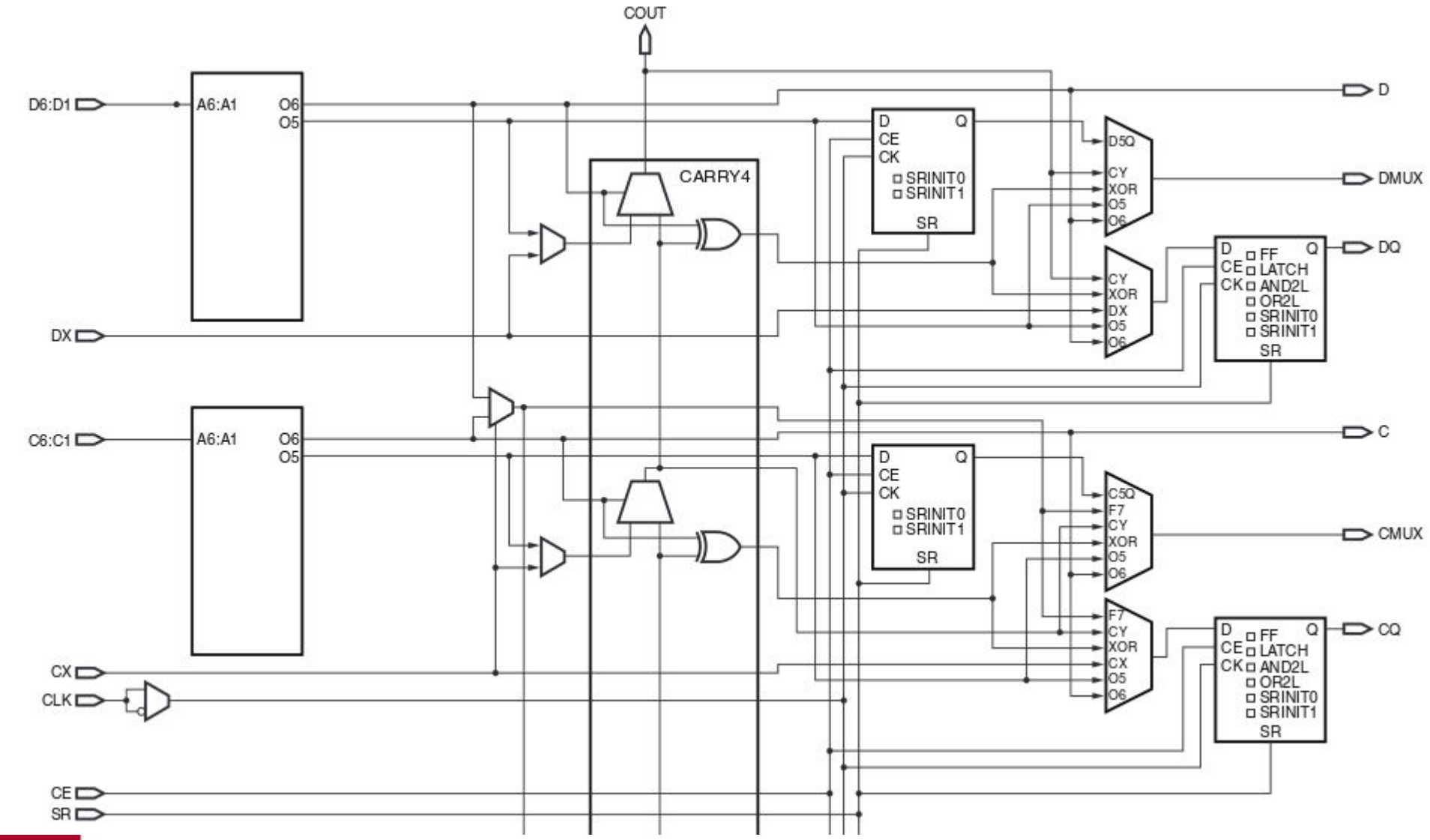
N crece también en tecnologías modernas (6 en Spartan-6)

Los CLB se pueden dividir en ‘Slices’ (Xilinx)

CLBs


C1
C2
C3
C4
H1
DIN
S/R
EC
G4
G3
G2
G1
Logic Function of G1-G4
G'
Logic Function of F', G' and H1
F4
F3
F2
F1
Logic Function of F1-F4
F'
M1
M2
M3
M4
M5
M6
M7
M8
SD
D
Q
EC
RD
Q2
Q1
1
G
F
K
(Clock)

½ Slice Spartan-6 (= ¼ CLB)

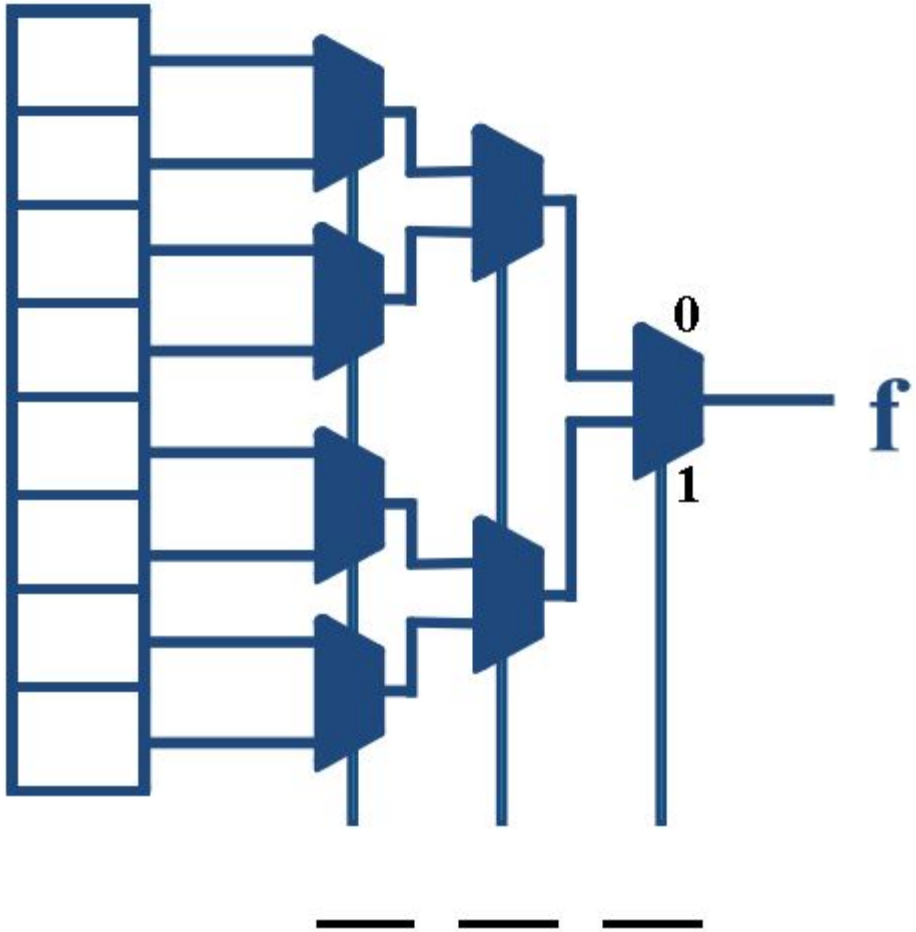


Look-Up Tables (LUTs)

En lugar de implementar las funciones lógicas con puertas lógicas, en FPGA se implementan con tablas de verdad

- Ej: Una LUT de 4 entradas (‘4-LUT’) puede implementar cualquier función lógica de 4 entradas

SRAM



Ejercicio

Configura la LUT de forma que implemente la función

$F = ABC + A\overline{B}\overline{C}$

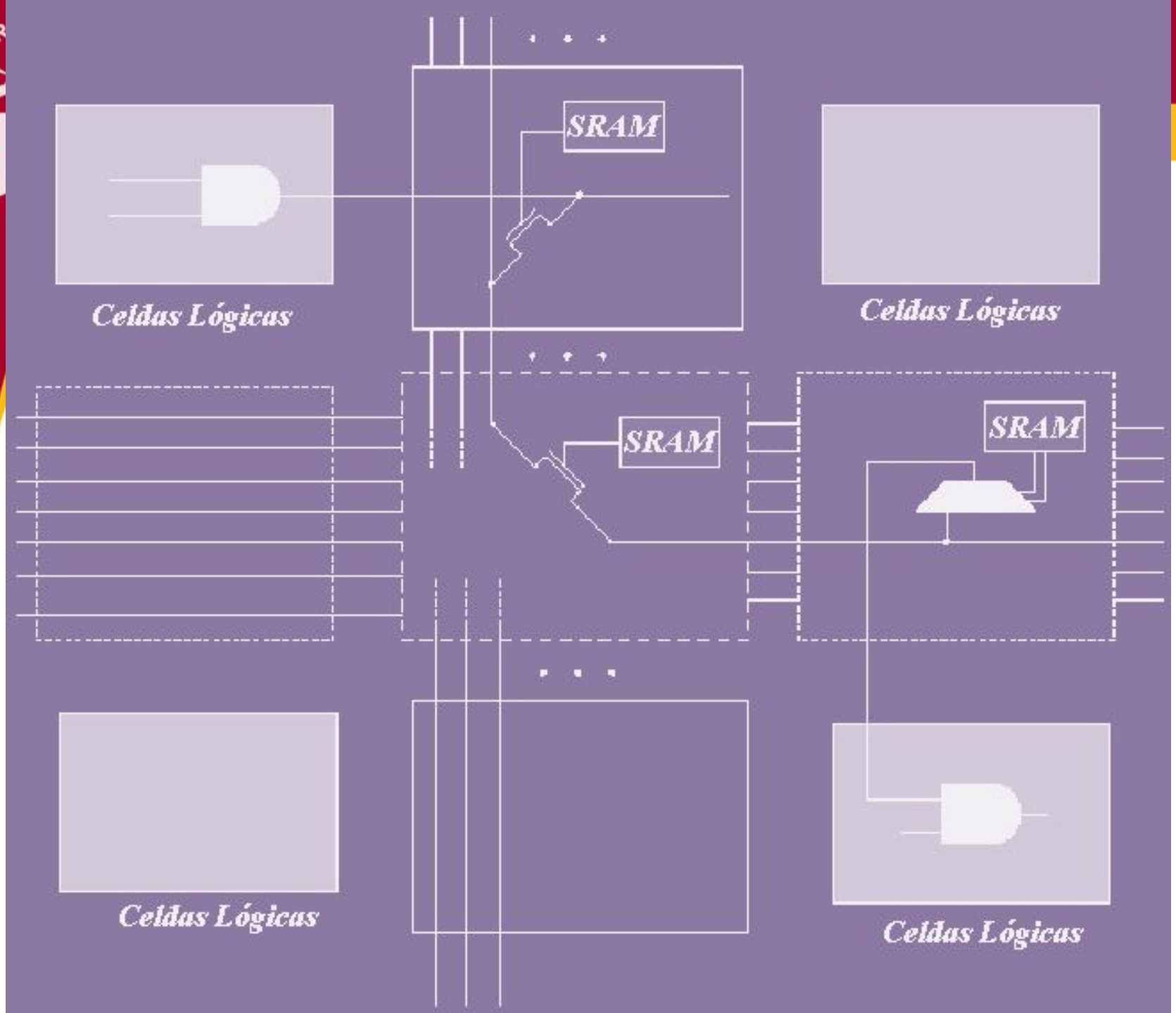
Otros bloques

En las FPGAs modernas estos bloques son cada vez más frecuentes:

- BRAM (Block RAM)
- DSP Blocks (procesado de señal)
- Transceptores de alta velocidad (comunicaciones)
- Microprocesadores

Recursos de Rutado

- PIP: Programmable Interconnection Points (Puntos de Interconexión Programable), también llamados Switching Matrix
- Líneas cortas y largas
- Recursos dedicados para relojes (ej: BUFG)


SRAM
Celdas Lógicas
Celdas Lógicas
SRAM
SRAM
Celdas Lógicas
Celdas Lógicas

Configurabilidad y Reconfigurabilidad

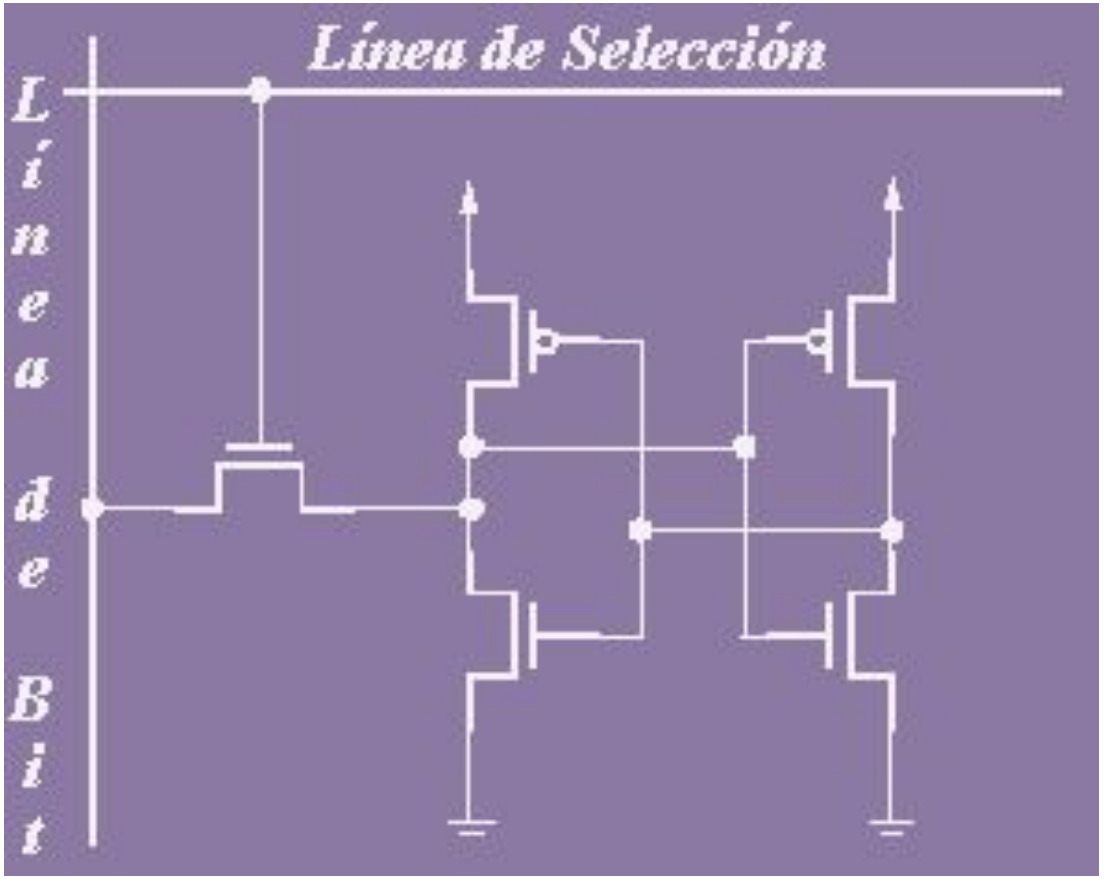
Existen varias tecnologías:

- SRAM: reconfigurable, volátil, muy extendida, aprovecha proceso CMOS estándar
- Flash: reconfigurable, no volátil, proceso no estándar
- Antifusible: no reconfigurable, proceso no estándar

Celda SRAM

Son dos inversores realimentados

4 transistores, pero CMOS estándar



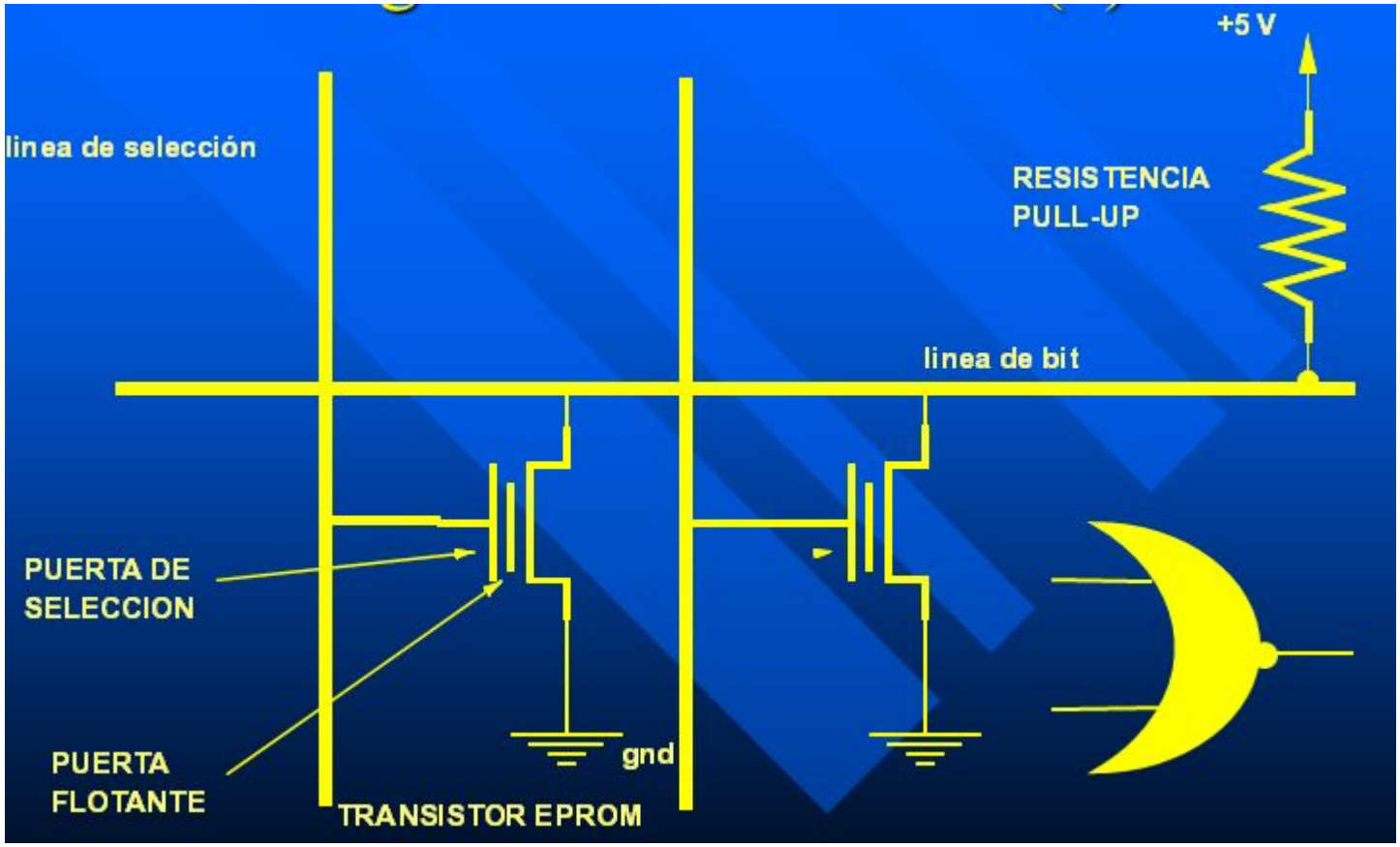
Tecnologías Flash

Se basan en el uso de FGMOS (Floating-Gate MOS)

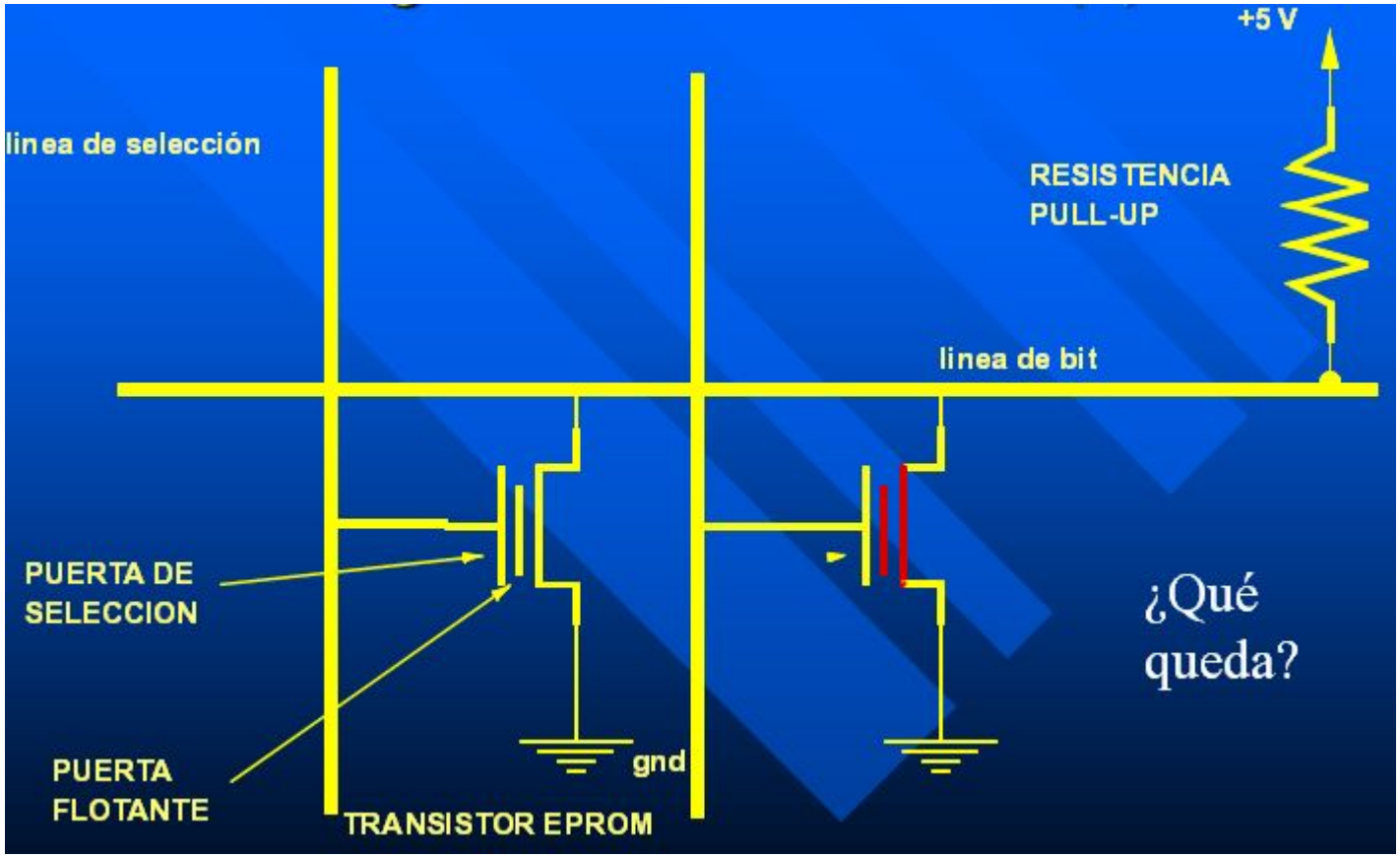
- Por lo que requieren de tecnologías con 2 niveles de polisilicio

Si la puerta flotante está cargada:
-> Incremento de la Vt
-> Transistor no puede encenderse ni con Vdd en la puerta

Ejemplo configurabilidad Flash

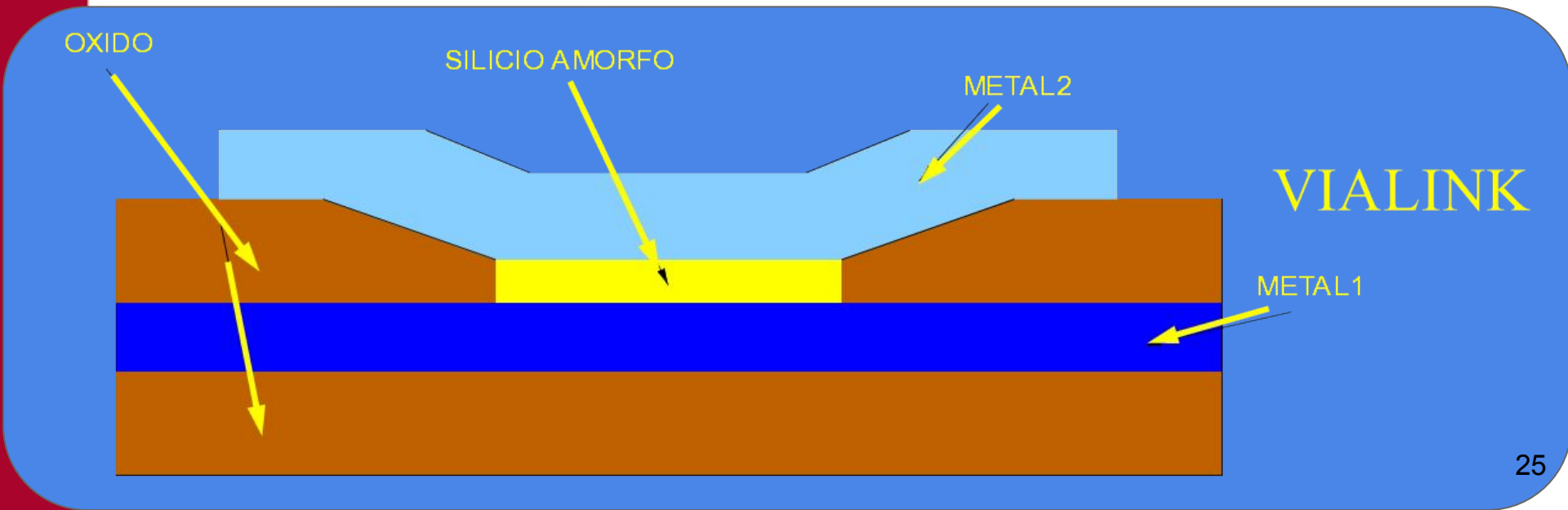


Ejemplo configurabilidad Flash



Tecnologías Antifusible

- OTP: One Time Programmable
- Requieren un proceso específico (no CMOS estándar)



Por tamaño

Para una misma tecnología, de menor a mayor área por bit de configuración:

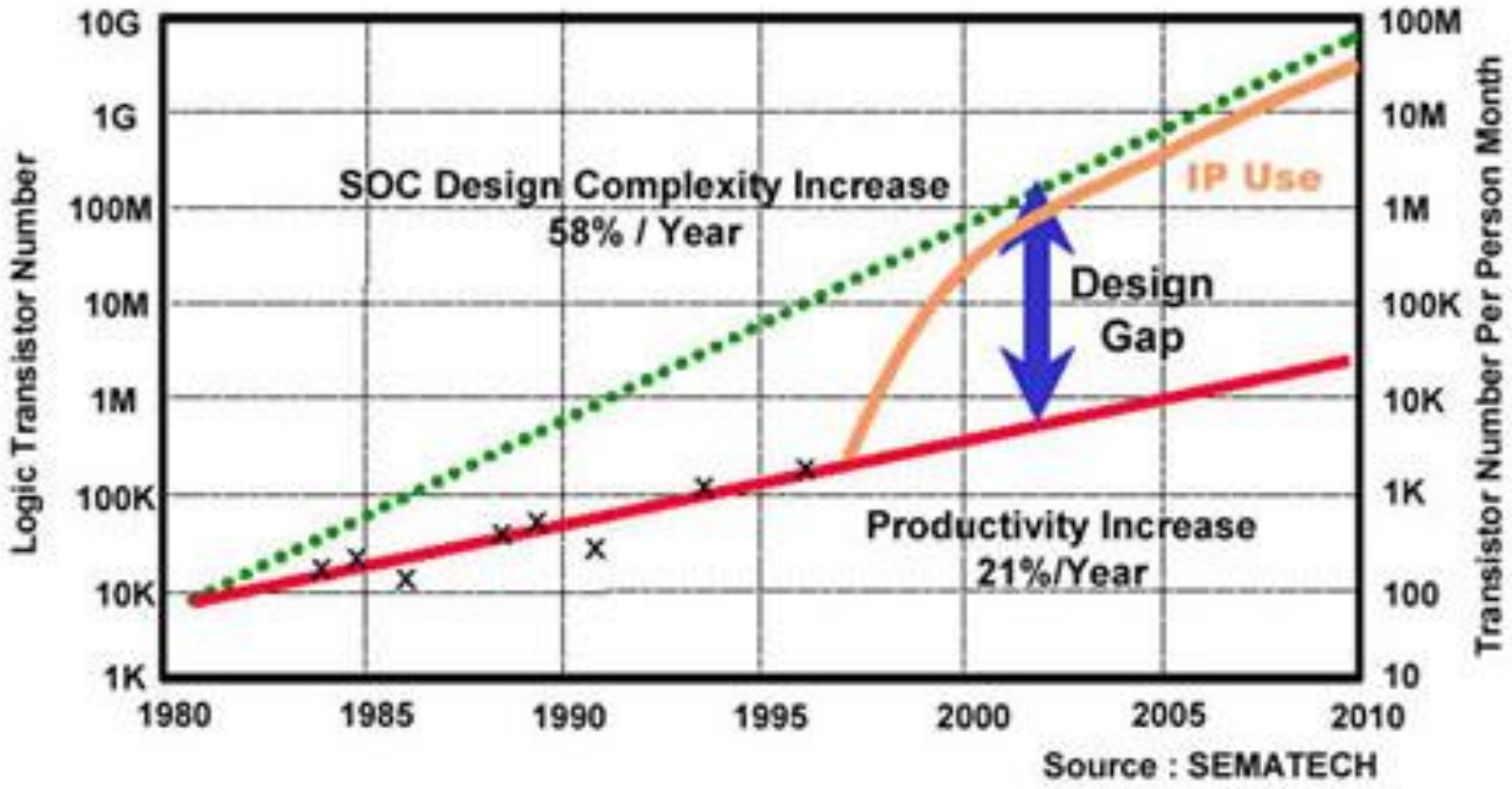
- Antifusible (<1 transistor)
- Flash (1 transistor)
- SRAM (4 transistores)

Pero SRAM, al ser CMOS estándar, escala perfectamente con la evolución de las tecnologías

Contenido

- Arquitectura de una FPGA
- Bloques constituyentes
- El ‘Design Gap’
- FPGAs como System-on-Chip
- Procesadores soft-core y hard macro
- Diseño con IP cores
- Conclusiones
- Bibliografía

El ‘Design Gap’

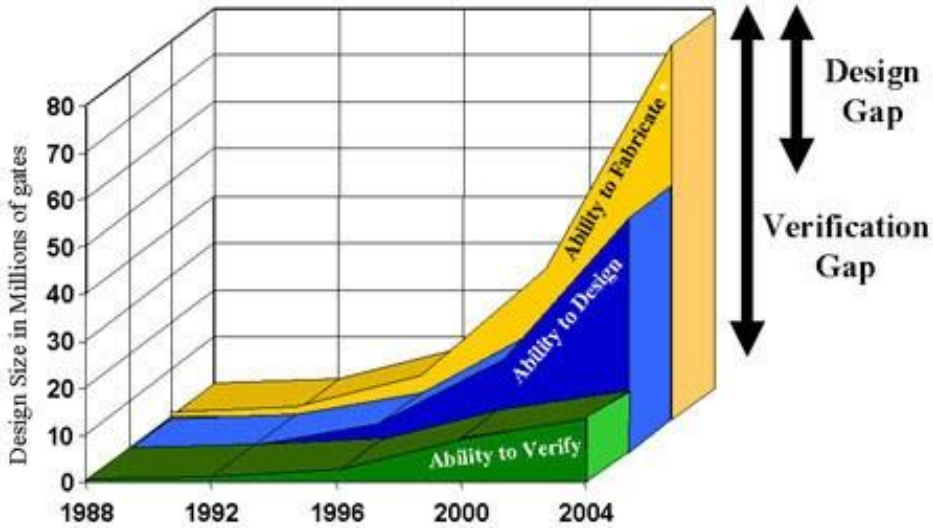


El ‘Design Gap’

- 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 ‘Design Gap’: soluciones

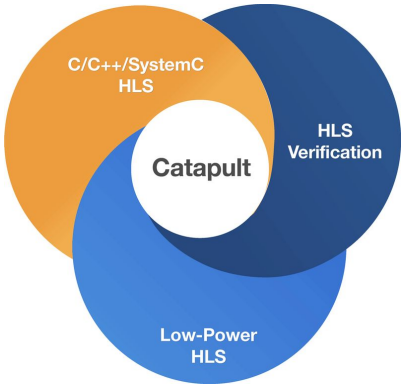
- Nuevas metodologías de diseño (síntesis de alto nivel)
- Uso de IP cores
- Uso de soft processors

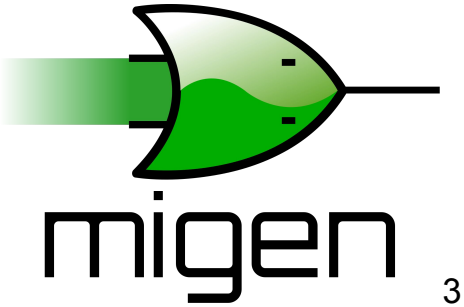


Ojo! Diseños más complejos implican mayor esfuerzo en **verificación**! También existe un ‘verification gap’

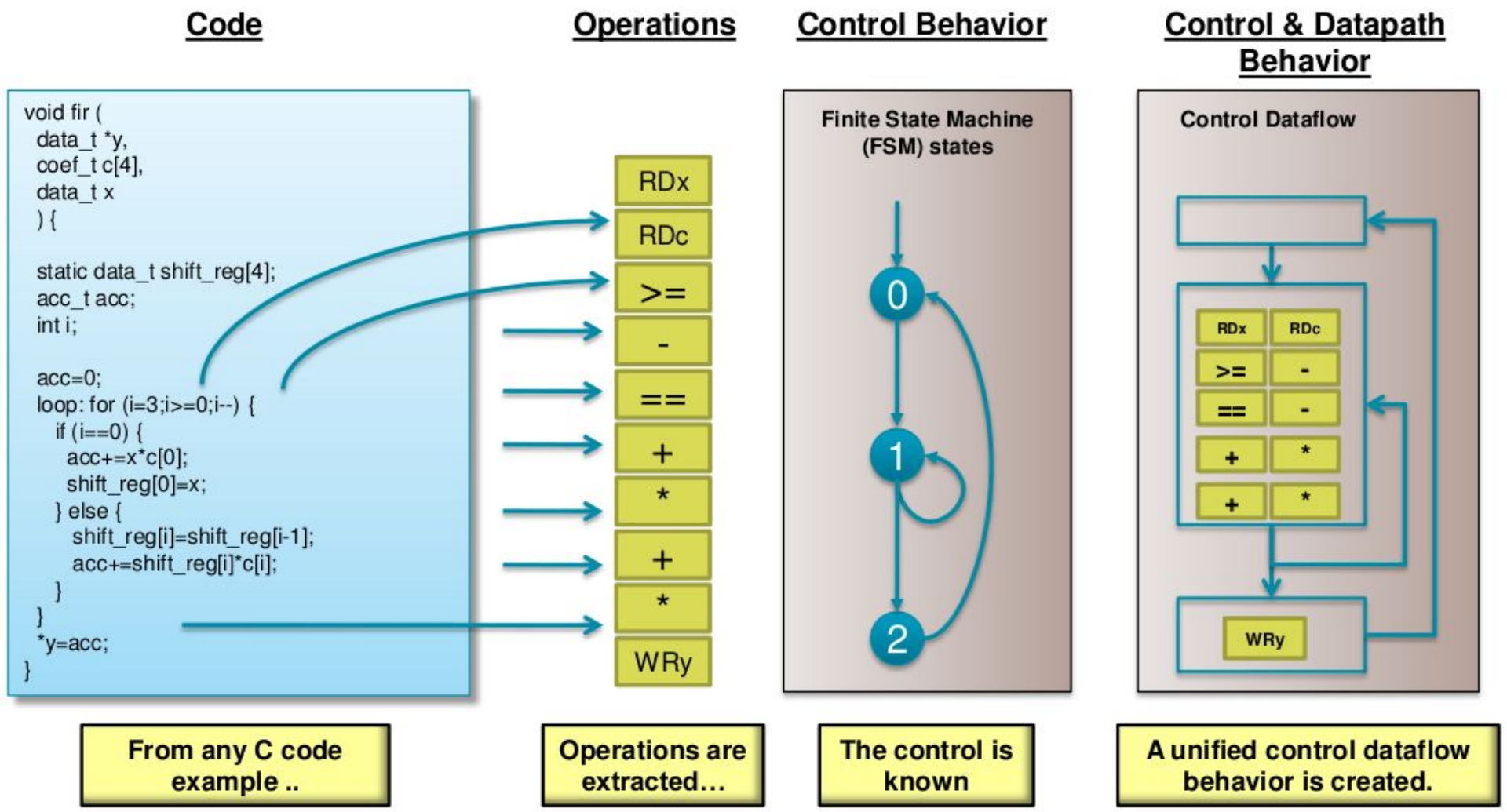
High(er) Level Synthesis (HLS)

- 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, Python) a hardware





Síntesis de alto nivel



Fuente: Xilinx

Síntesis de alto nivel

```
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;
}
```

Fuente: Xilinx

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

Síntesis de alto nivel

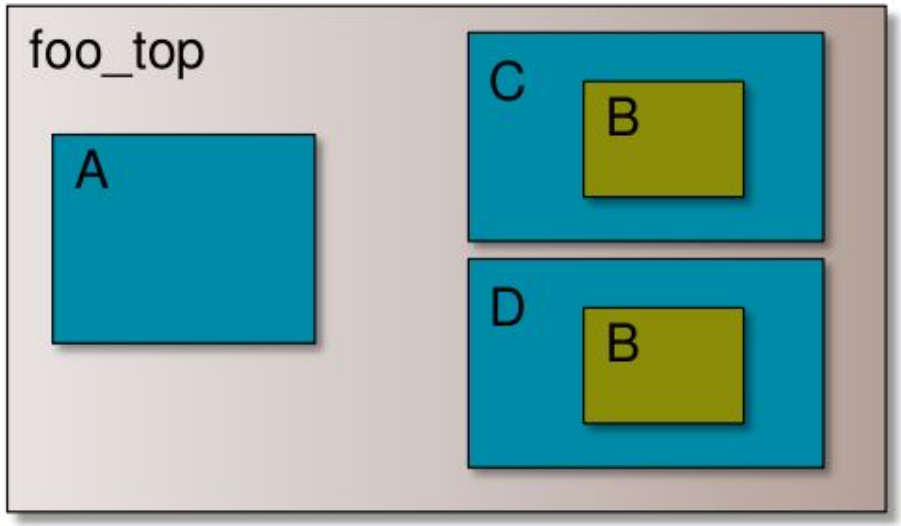
Source Code

```
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

Fuente: Xilinx

Consejos y conclusiones

- Se reduce el tiempo de desarrollo pero se incrementa el tiempo de procesado y recursos ocupados
- Existe una tendencia por parte de los fabricantes al uso y desarrollo de estas herramientas porque amplía su base de usuarios
- El principal riesgo es utilizar estas herramientas sin conocer en qué se traduce (perdiendo de vista la implementación hardware)
- Otro riesgo es el “vendor lockdown”: el fabricante “atrapa” tu diseño en su lenguaje + restricciones específico
- Hay que empezar por ejemplos pequeños, estudiando bien cómo se traducen a hardware

Contenido

- Arquitectura de una FPGA
- Bloques constituyentes
- El ‘Design Gap’
- FPGAs como System-on-Chip
- Procesadores soft-core y hard macro
- Diseño con IP cores
- Conclusiones
- Bibliografía

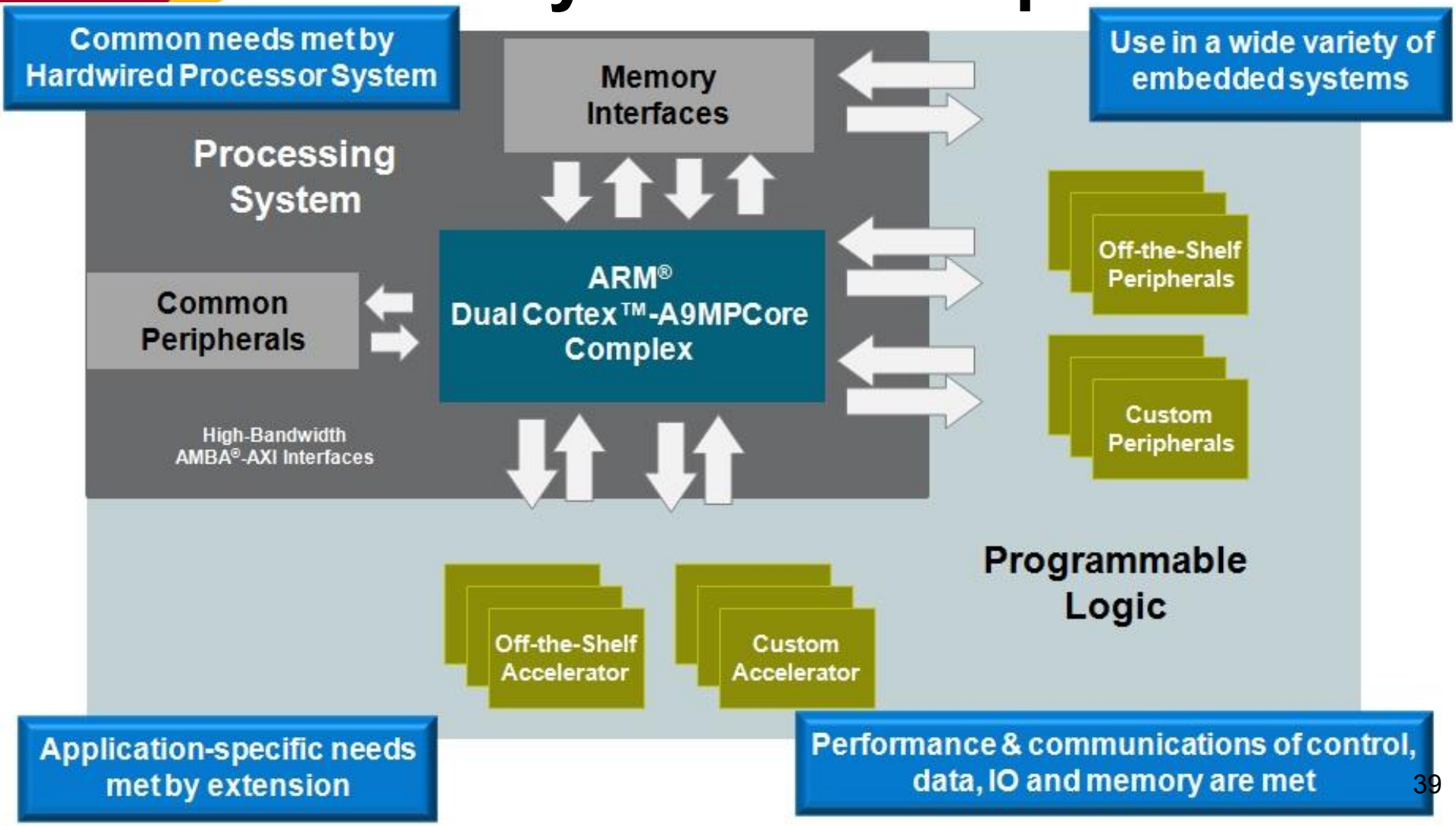
FPGAs como System-on-Chip

- Evolución de las tecnologías microelectrónicas (Ley de Moore)
- Nos lleva a una arquitectura de FPGA moderna con más tipos de bloques constituyentes...

Arquitectura de una FPGA moderna

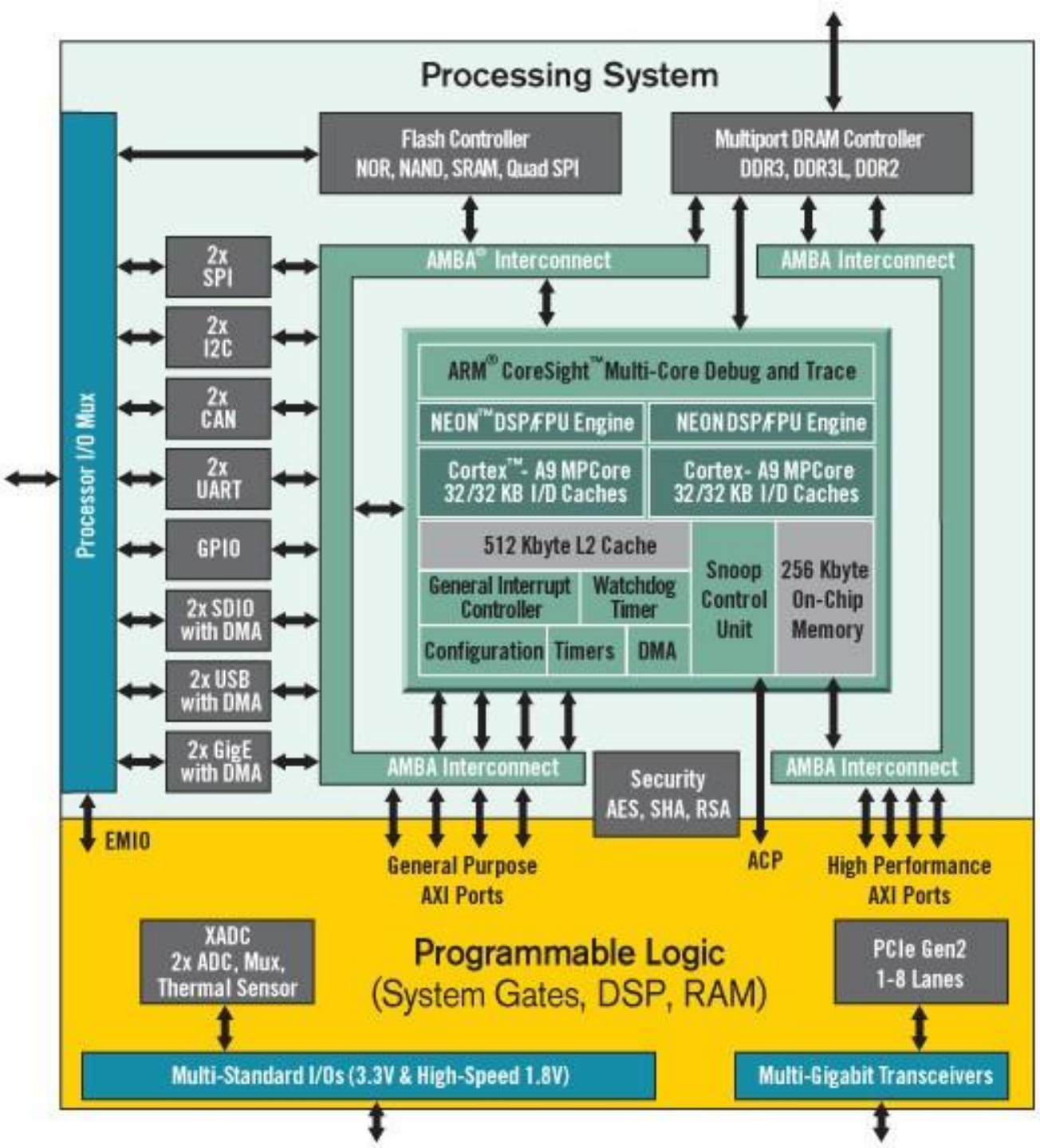
- Además de IOBs, CLBs y recursos de rutado:
- Memorias empotradas (Block RAMs)
- Bloques DSP (Digital Signal Processing)
- Conexiones de alta velocidad (Gigabit transceivers, PCIexpress, …)
- Microprocesadores!

System-on-Chip



Contenido

- Arquitectura de una FPGA
- Bloques constituyentes
- El ‘Design Gap’
- FPGAs como System-on-Chip
- Procesadores soft-core y hard macro
- Diseño con IP cores
- Conclusiones
- Bibliografía



Hard macro

Fabricado en silicio

Ejemplo Familia Zynq

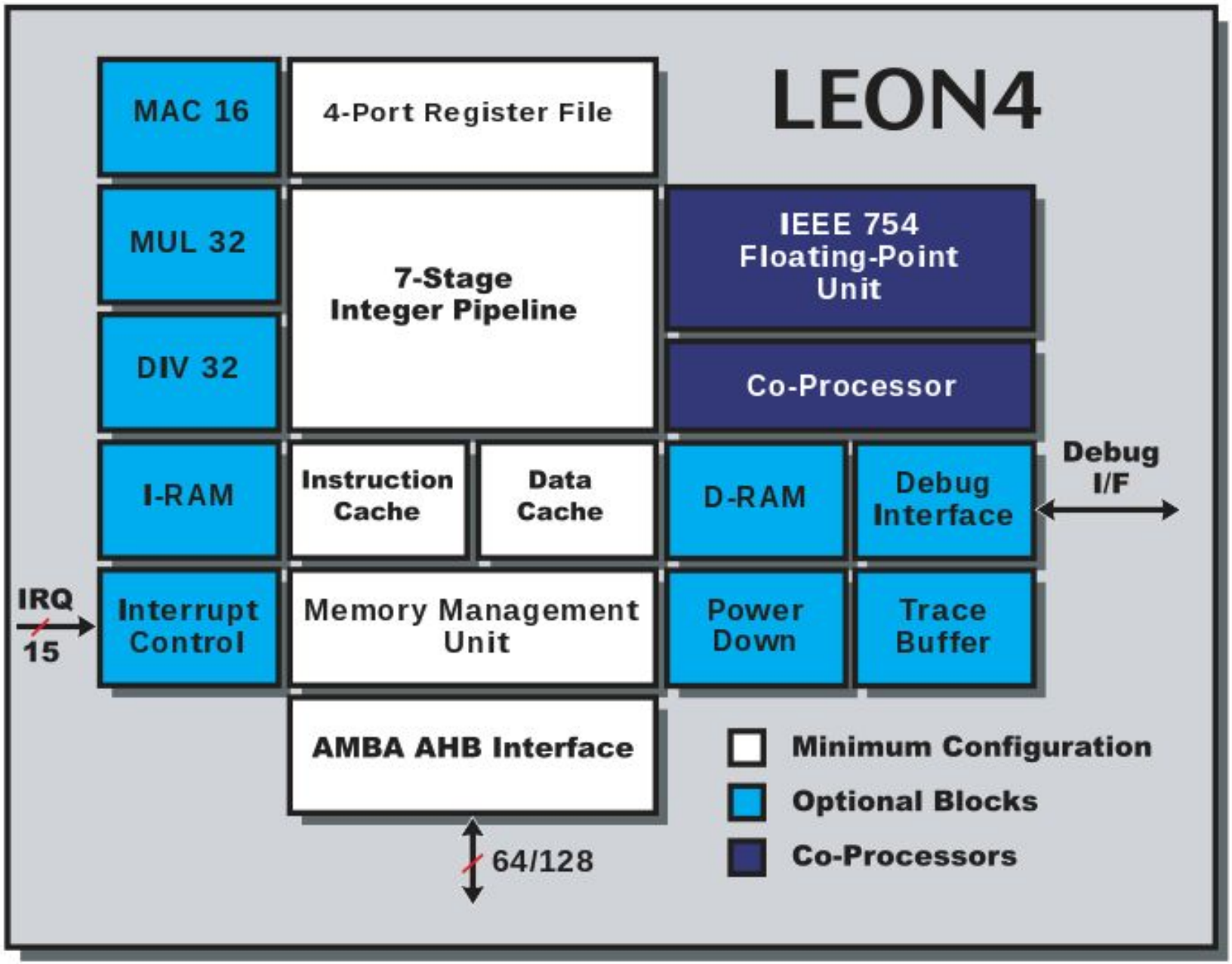
(ARM9 “hard macro”)

Soft processors

(Como contraposición a “hard macro”)

- 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 fabricado en silicio, 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
- RISC-V
- ...

Algunos problemas:

- Consumo de recursos de la FPGA
- 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:

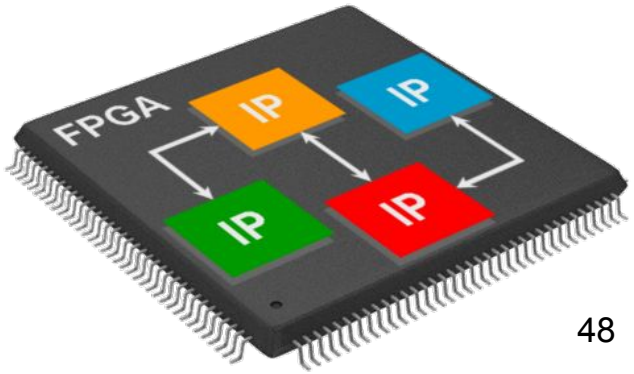
- Diseño óptimo: HW y SW se encargan cada uno de lo que les es más eficiente
- Simplicidad en las comunicaciones de tu diseño HDL con el exterior: USB, TCP/IP, …
- Una actualización no es sólo cambiar el programa: podemos añadir periféricos nuevos (por ejemplo un timer)

Contenido

- Arquitectura de una FPGA
- Bloques constituyentes
- El 'Design Gap'
- FPGAs como System-on-Chip
- Procesadores soft-core y hard macro
- Diseño con IP cores
- Conclusiones
- Bibliografía

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

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!!

Contenido

- Arquitectura de una FPGA
- Bloques constituyentes
- El ‘Design Gap’
- FPGAs como System-on-Chip
- Procesadores soft-core y hard macro
- Diseño con IP cores
- Conclusiones
- Bibliografía

Conclusiones

- FPGAs modernas son dispositivos de alta complejidad
- Conocer sus bloques constituyentes ayuda a optimizar el diseño
- Soluciones System-on-Chip: muy útiles cuando necesitamos procesado paralelo y secuencial de alta complejidad
- HLS, SoC, IP cores: ayudan a diseñar sistemas de alta complejidad, pero requieren conocimiento/esfuerzos específicos (no son soluciones “automágicas”)

Contenido

- Arquitectura de una FPGA
- Bloques constituyentes
- El ‘Design Gap’
- FPGAs como System-on-Chip
- Procesadores soft-core y hard macro
- Diseño con IP cores
- Conclusiones
- Bibliografía

Bibliografía

- Rodríguez Andina, J. J., de la Torre Arnanz, E., Valdés Peña, M. D., *FPGAs: Fundamentals, Advanced Features, and Applications in Industrial Electronics*. CRC Press, 2017
- Michael Keating, Pierre Bricaud, *Reuse Methodology Manual for System-on-a-Chip Designs*. Kluwer Academic, 2002

Resultados de aprendizaje

- ¿Qué bloques básicos componen una FPGA?
- Conocer y saber distinguir entre IOBs, CLB, y otros bloques constituyentes de una FPGA
- Conocer cómo se programa una LUT y saber identificar, a partir de su programación, la función que implementa
- Saber por qué es importante mitigar el “design gap” al diseñar sistemas de alta complejidad
- ¿Qué ventajas e inconvenientes tienen el diseño con High Level Synthesis, el diseño con IP cores y el uso de soft processors en FPGA?