Softcores, System On Chip y Diseño

David Alejandro González Márquez

Programación de softcores en FPGAs
Programa de Profesoras/es Visitantes
Departamento de computación
Universidad de Buenos Aires

Clase disponible en: `https://github.com/fokerman/fpgaSoftcoreProgrammingCourse`

Softcores, System On Chip y Diseño

Los procesadores diseñados o configurados especialmente para implementarse sobre un FPGA se denominan **Softcores**.

Estos requieren soporte de acceso a memoria, memoria cache, periféricos y de sistemas de interconexión, construyendo lo que denominamos un **System On Chip**.

Si el soporte es muy limitado, diseñado con un propósito específico y completamente embebido lo llamamos **Microcontrolador**.

La interconexión por su parte requiere buses y control de estos, mediante alguna **interfaz de interconexión**.

Adicionalmente no todo diseño es útil para cualquier problema.
Es fundamental entender como combinar las capacidades del software con el hardware flexible para lograr un **Codiseño Hardware/Software** eficiente.

Softcores

Existen diferentes implementaciones de *Softcores* con distintas características y propiedades.
Estos pueden ser completamente configurables, o no, ser parte de una solución integrada a un SoC, ser completamente independientes o incluso limitarse a comportarse como un Microcontrolador.

Algunos ejemplos,

- PicoBlaze
- MicroBlaze
- LEONN3
- Nios II
- PULPino
- Rocket
- BOOM

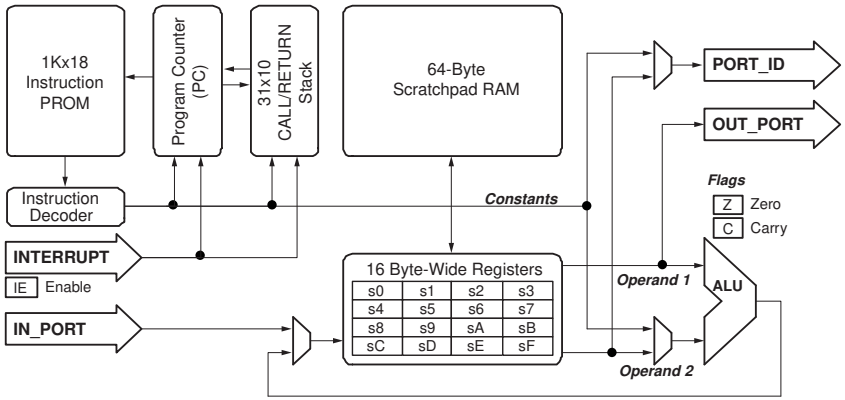
Vamos a recorrer las características de agunos procesadores, SoC o MCU seleccionados.

Softcores - PicoBlaze

Microcontrolador de 16 bits desarrollado por Xilinx, con 1KB de memoria de instrucciones, 64 bytes de RAM, 256 puertos de entrada salida y pila de 31 posiciones. Bajo licencia BSD.

Todas las instrucciones demoran 2 ciclos (predecible), e interrupciones de respuesta rápida en 5 ciclos.

Optimizado para FPGAs, requiriendo solo 96 celdas según documentación.

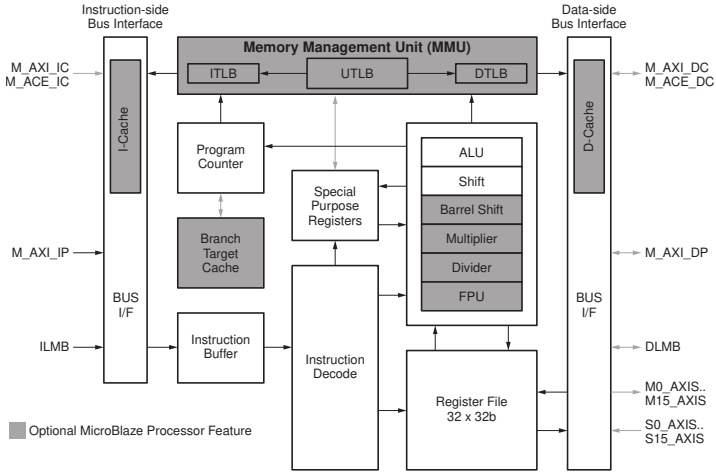


Softcores - MicroBlaze

Procesador de 32 bits completamente configurable, con soporte para todo tipo de conexiones e interfaces.

Integrado a las herramientas de Xilinx para su uso dentro de FPGAs del fabricante.

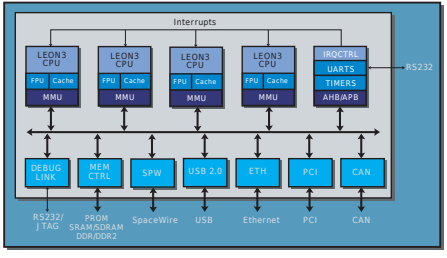
Soporta gran cantidad de bibliotecas de código. Es de código cerrado.



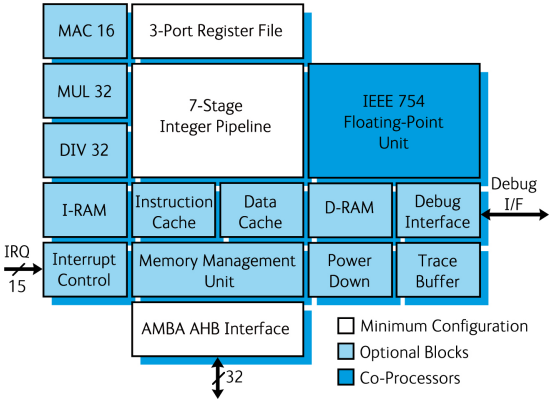
Softcores - LEON3

Procesador de 32 bits basado en la arquitectura SPARC V8 desarrollado por Aeroflex Gaisler.

Completamente configurable, soportando hasta 16 cores implementados como *asymmetric multiprocessing* (AMP) o *synchronous multiprocessing* (SMP).



Todo el código disponible bajo licencia GNU GPL para evaluación, investigación o fines educativos.



https://www.gaisler.com/index.php/products/processors/leon3

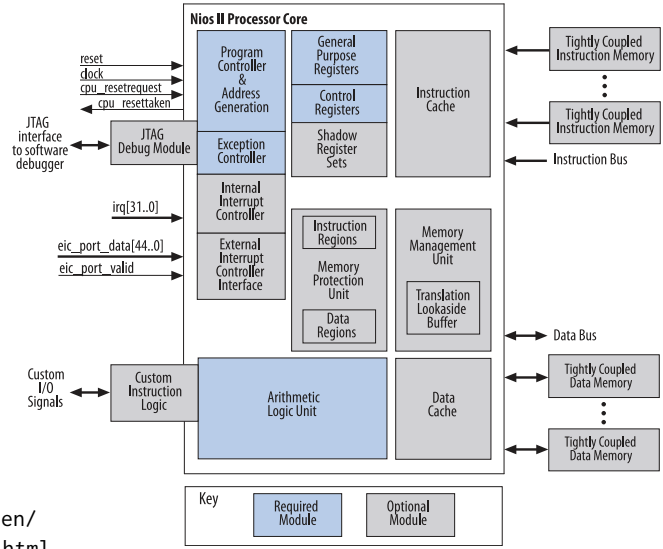
Softcores - Nios II

Procesador de 32 bits desarrollado por Altera, actualmente Intel.

Diseñado como procesador embebido de propósito general, posee un pipeline de 5 etapas In-Order y un predictor de saltos estático.

Soportado dentro de las herramientas de Altera, con diferentes configuraciones.

Es de código cerrado.



https://www.intel.com/content/www/us/en/products/details/fpga/nios-processor.html

Softcores - PULPino

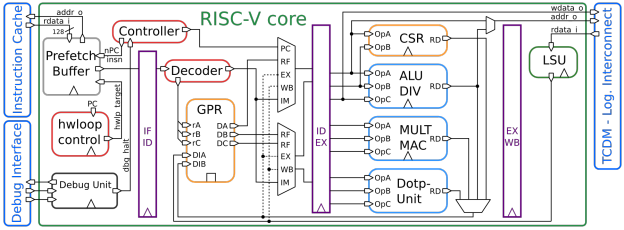
Procesador de 32 bits basado en RISC-V desarrollado por ETH Zurich.

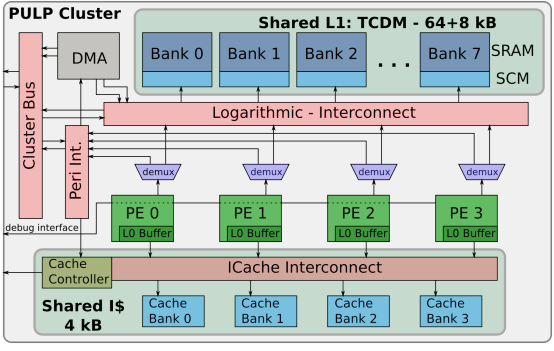
Su implementación es In-Orden single-issue con un pipeline de 4 etapas.

Configurable y diseñado para muy bajo consumo y muy poca área.
Soporta extensiones para punto flotante de precisión simple.

Distribuido bajo licencia Apache 2.0. y codificado en System Verilog.

`https://github.com/pulp-platform/pulpino`

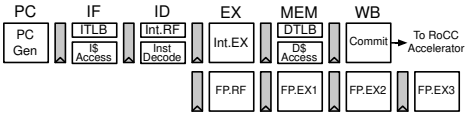




Softcores - Rocket

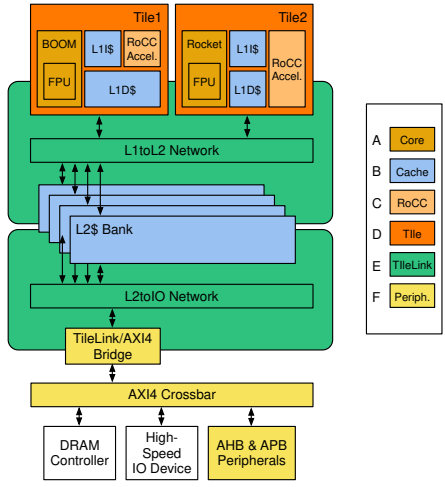
Procesador desarrollado por Berkeley, actualmente mantenido por SiFive. Empresa privada que comercializa soluciones basadas en este procesador.

Es un procesador In-Order single-issue con un pipeline de 5 etapas.



Junto con memorias cache para datos y código se construye un *tile* que se utilza como procesador del Rocket Chip SoC. Completamente codificado en Chisel.

`https://github.com/chipsalliance/rocket-chip`



Softcores - Boom

Procesador desarrollado por Berkeley basado en RISC-V. Soporta hasta las extensiones RV64GC.

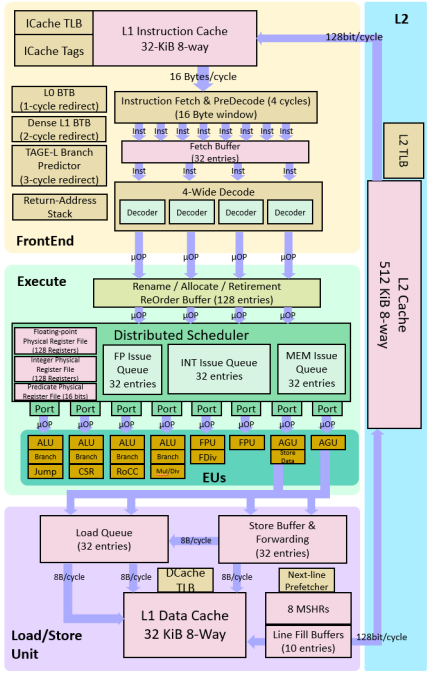
Su microarquitectura es Out-Of-Order, parametrizable y diseñada para alto rendimiento.

Creado el grupo de Arquitectura de Procesadores con objetivos de académicos y de investigación.

La última versión BOOMv3 alcanza un rendimiento similar a procesadores comerciales equivalentes.

Esta codificado en Chisel y distribuido bajo licencias BSD.

`https://github.com/riscv-boom/riscv-boom`



Interconexionado

Existen múltiples soluciones propietarias de buses y sistemas para la interconexión de dispositivos, ya sea on-chip o off-chip. Sin embargo el más utilizado es AMBA (Advanced Microcontroller Bus Architecture), un protocolo abierto desarrollado por ARM.

Este protocolo especifica el mecanismo la interconexión on-chip para la comunicación y administración de bloques funcionales en un SoC.

1996 Primera versión. Especificación de Advanced System Bus (ASB) y Advanced Peripheral Bus (APB).

1999 AMBA 2, se agrega High-performance Bus (AHB) que implementa transferencias en dos etapas.

2003 AMBA 3, incluye Advanced eXtensible Interface (AXI) para soluciones de mayor rendimiento y Advanced Trace Bus (ATB) como solución de debug on-chip.

2010 AMBA 4, se introducen las especificaciones de AXI4.

2010 Extensiones para coherencia en todo el sistema usando AMBA 4 AXI Coherency Extensions (ACE).

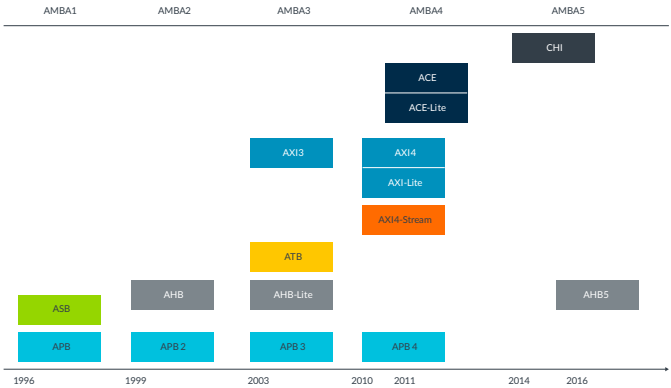
2013 AMBA 5, introducen el Coherent Hub Interface (CHI). Mejora los mecanismos de coherencia.

`https://www.arm.com/architecture/system-architectures/amba/amba-specifications`

Interconexionado - AMBA

Estos protocolos se comercializan mediante una familia de dispositivos sintetizables de propiedad intelectual (*synthesizable intellectual property (IP) cores*).

Productos licenciados por ARM que implementan buses digitales on-chip para mover y almacenar eficientemente información bajo la especificación de los protocolos AMBA.



Muchos fabricantes utilizan los buses AMBA en diseños no ARM.

Xilinx implementa sus propios IP para utilizar protocolos AMBA dentro de sus FPGA.

Actualmente es un estandar de facto

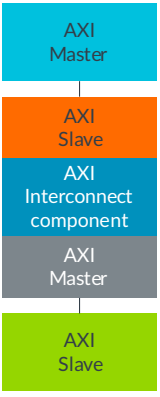
Interconexión - Protocolo AXI

La especificación AXI define la interfaz que los bloques IP deben cumplir para la interconexión.

En AXI, solo hay dos tipos de interfaces: **master** y **slave**. Ambas interfaces son simétricas.

Todas las conexiones AXI se realizan entre interfaces maestras e interfaces esclavas.

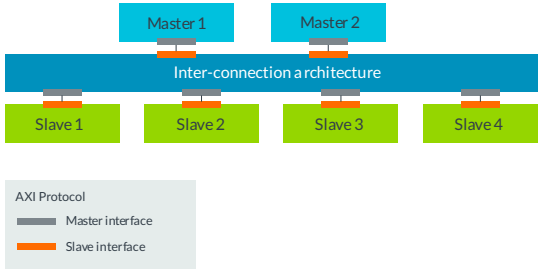
Las interfaces de interconexión AXI contienen las mismas señales, lo que hace que la integración de diferentes IP sea relativamente simple.



Interconexión - Protocolo AXI - Inter-connection architecture

Ejemplo simplificado de un sistema SoC, que se compone de maestros, esclavos y la interconexión que conecta todo.

Un procesador podría ser un ejemplo de maestro, y un controlador de memoria podría ser un ejemplo de esclavo.



AXI define las señales y la temporización de las conexiones punto a punto entre maestros y esclavos. Cuando están involucrados varios maestros y esclavos, se requiere una estructura de interconexión.

El *interconnect fabric* también implementa interfaces maestras y esclavas, donde se implementa el protocolo AXI.

Interconexión - Protocolo AXI - Channels

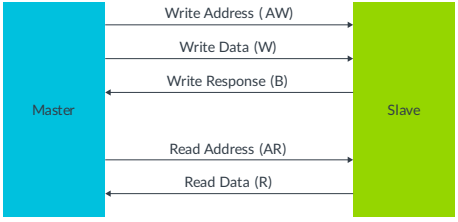
AXI define canales para la comunicación.

Operaciones de escritura:

- El maestro envía una dirección en el canal **Write Address (AW)** y transfiere datos en el canal **Write Data (W)** al esclavo.
- El esclavo escribe los datos recibidos en la dirección especificada. Una vez que el esclavo ha completado la operación de escritura, responde con un mensaje al maestro en el canal **Write Response (B)**.

Operaciones de lectura:

- El maestro envía la dirección que desea leer en el canal **Read Address (AR)**.
- El esclavo envía los datos desde la dirección solicitada al maestro en el canal **Read Data (R)**.



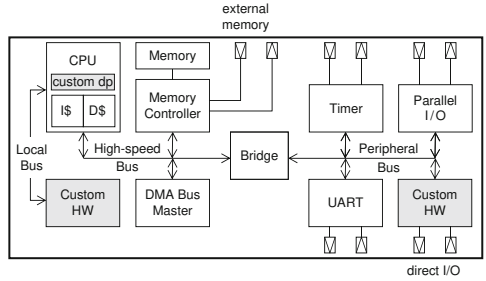
El esclavo también puede devolver un mensaje de error en el canal **Read Data (R)**.
Se produce un error si, por ejemplo, la dirección no es válida, o los datos están dañados, o el acceso no tiene el derecho permiso de seguridad.

Cada canal es **unidireccional**, por lo que se necesita un canal de respuesta de escritura independiente para pasar las respuestas de vuelta al maestro.

System On Chip

Un SoC combina componentes bajo un mismo bus. El procesador (típicamente RISC) que juega el rol de conductor del SoC.

Otros componentes también son incluidos, memorias on-chip, interfaces de memoria off-chip, dispositivos dedicados, co-procesadores, e infraestructura de comunicación entre componentes.



El **dominio de la aplicación** afecta sustancialmente el tipo dispositivos, tamaño de la memoria y la naturaleza de las comunicaciones on-chip.

Una configuración particular de todos estos elementos se denomina **plataforma**, especializada para un dominio de computo particular.

System On Chip

La especialización de un SoC tiene las siguientes ventajas:

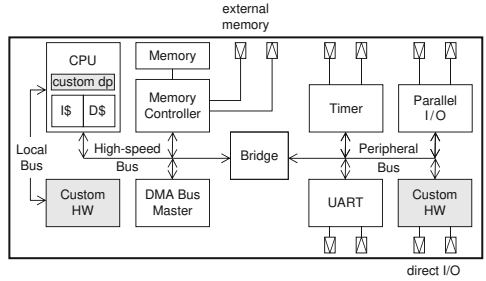
- La **especialización** asegura que su eficiencia de procesamiento sea mayor en comparación con las soluciones de propósito general.
  Una mayor eficiencia de procesamiento significa un menor consumo de energía (mayor duración de la batería) o un mayor rendimiento absoluto.

- La **flexibilidad** asegura que sea una solución reutilizable que funcione en múltiples aplicaciones.
  Como resultado, el costo por aplicación del diseño disminuye, las aplicaciones se pueden desarrollar más rápido y el SoC en sí mismo se vuelve más barato porque se puede fabricar para un mercado más grande.

System On Chip

La arquitectura de un SoC se puede analizar en cuatro aspectos ortogonales:

1. Control
2. Comunicación
3. Computo
4. Almacenamiento



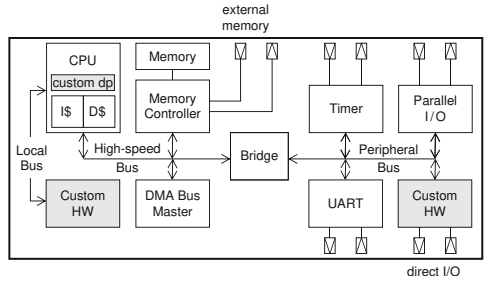
El rol de **control** lo toma el procesador, responsable de señales de control y recopilar el estado de los componentes. Puede contar o no con memoria de instrucciones local (o bien una cache).

La **comunicación** se implementa usando buses a lo largo de todo el sistema. Donde cada componente responde a un determinado rango de memoria. Inclusive contando con DMAs.

Es común dividir los buses en segmentos. Limitando la cantidad de componentes conectados a cada segmento de acuerdo con la necesidad de comunicación.

Interfaces con Hardware dedicado

En un SoC podemos integrar módulos de hardware personalizado. Esto quiere decir una máquina digital dedicada descrita como un FSM o como una máquina microprogramada.



Se pueden distinguir tres enfoques:

- Integrar un módulo de hardware personalizado como un **periférico** estándar en un bus del sistema. El microprocesador se comunica con el módulo de hardware personalizado mediante accesos a memoria de lectura/escritura.
- Conectar hardware personalizado a través de un **bus local** o una interfaz de coprocesador proporcionada por el microprocesador. La comunicación entre el módulo y el microprocesador seguirá un protocolo dedicado, definido por el bus local o la interfaz del coprocesador.
- Los microprocesadores también pueden proporcionar un medio para **integrar hardware personalizado** dentro del microprocesador. Luego, el conjunto de instrucciones del microprocesador se amplía con nuevas instrucciones adicionales para controlar este hardware personalizado.

Principios de diseño de Arquitecturas SoC

Si bien un SoC es un dispositivo muy específico para un dominio de aplicación.
Existen algunas máximas que se deben considerar en el diseño.

A continuación se presentan cuatro principios de diseño que rigen la mayoría de las arquitecturas SoC.

1. Procesamiento de datos heterogéneo y distribuido.
2. Comunicaciones heterogéneas y distribuidas.
3. Almacenamiento heterogéneo y distribuido.
4. Control jerárquico.

Vamos a ir describiendo cada una de las maximas.

Procesamiento de datos heterogéneo y distribuido

Un SoC puede contener múltiples unidades computacionales independientes (distribuidas). Estas unidades pueden ser heterogéneas e incluir FSM, máquinas microprogramados o microprocesadores.

Se pueden distinguir tres formas de paralelismo de procesamiento de datos.

1. Paralelismo a **nivel de palabra**
   Permite el procesamiento paralelo de múltiples bits en una palabra.
2. Paralelismo a **nivel de operación**
   Permite ejecutar múltiples instrucciones simultáneamente.
3. Paralelismo a **nivel de tareas**
   Permite que múltiples hilos de control independientes se ejecuten de forma independiente.

Solo un SoC admite el paralelismo a nivel de tarea.

Comunicaciones heterogéneas y distribuidas

El bus central es un recurso crítico compartido por muchos componentes en un SoC.

Para evitar que este recurso se convierta en un **cuello de botella** se divide el bus en varios **segmentos** utilizando *bus bridges*.
El *bus bridge* es un mecanismo para distribuir la comunicación en el chip.

Los requisitos de comunicación en el chip suelen ser variados, por lo tanto los mecanismos de interconexión de SoC también deberían ser heterogéneos.

Puede haber buses compartidos, conexiones punto a punto, conexiones en serie y conexiones en paralelo.

Las comunicaciones SoC heterogéneas y distribuidas permiten a un diseñador **explotar el ancho de banda de comunicación en el chip**.

Almacenamiento heterogéneo y distribuido

En lugar de una sola memoria central, un SoC utilizará una colección de memorias dedicadas.

- Los procesadores pueden contener memorias de instrucciones locales o utilizar memorias caché para mantener copias locales de datos e instrucciones.
- Los coprocesadores y otros componentes pueden utilizar bancos de registros locales.
- Los aceleradores especializados pueden usar memorias dedicadas para aplicaciones específicas, como almacenamiento en buffers de bloques o un *local scratchpad*.

Este almacenamiento se implementa con una colección de diferentes tecnologías de memoria.

Existen cinco categorías básicas almacenamiento basado en silicio.

Almacenamiento heterogéneo y distribuido

- Los **registros** son el tipo de memoria más rápido disponible. Residen más cerca de los elementos de computación de una arquitectura.
- La **memoria dinámica de acceso aleatorio** (DRAM) proporciona almacenamiento económico a densidades muy altas. A diferencia de los registros, las celdas DRAM utilizan una tecnología diferente y por lo tanto, son más complejas de interar en un solo chip.
- La **memoria estática de acceso aleatorio** (SRAM) se utiliza cuando se requiere un almacenamiento rápido de lectura y escritura. SRAM tiene menor densidad y mayor consumo de energía que DRAM. Facil de integrar, utiliza la misma tecnología que los SoC.
- La **memoria no volátil de solo lectura** (NVROM) se usa para aplicaciones que solo requieren acceso de lectura en una memoria. Por ejemplo para almacenar las instrucciones de un programa. Las memorias no volátiles tienen una densidad más alta que la SRAM.
- La **memoria de acceso aleatorio no volátil** (NVRAM) se usa para aplicaciones que necesitan memorias de lectura y escritura que no pierden datos cuando se desconecta la alimentación. La velocidad de lectura y escritura en una NVRAM puede ser asimétrica.

Control jerárquico

Una jerarquía de control significa que todo el SoC opera como **una sola entidad lógica**.
Esto implica que todos los componentes deberán **sincronizarse** en algún momento.

Por ejemplo, considerar un programa que utiliza un coprocesador implementado como periférico. Este deberá enviar argumentos al coprocesador, esperar a que termine y finalmente recuperar el resultado.

En este caso:

- El coprocesador se puede implementar con una FSM o una máquina microprogramada.
- El procesador mantendrá el control del sistema y distribuirá los comandos al hardware.

El diseño de una buena jerarquía de control es un problema desafiante.

- Se debe **aprovechar** al máximo la naturaleza distribuida del SoC → ejecutar en paralelo.
- Se debe **minimizar** la cantidad de conflictos que surgen como resultado de ejecutar en paralelo.

Debido a la jerarquía de control, todos los componentes están lógicamente conectados entre sí y cada uno de ellos puede provocar un cuello de botella en el sistema.
El desafío es conocer la ubicación de dichos cuellos de botella del sistema y controlarlos.

Bibliografía

- Patrick R. Schaumont. **“A Practical Introduction to Hardware/Software Codesign”**, Sringer 2010
- Ross K. Snider. **“Advanced Digital System Design using SoC FPGAs.”**, Springer 2022
- Iouliia Skliarova, Valery Sklyarov. **“FPGA-BASED Hardware Accelerators”**, Sringer 2019. Lecture Notes in Electrical Engineering Volume 566
- **“AMBA AXI and ACE Protocol Specification. AXI3 , AXI4 , and AXI4-Lite, ACE and ACE-Lite”**, 2011 ARM. ARM IHI 0022D (ID102711)
- Krste Asanović et al. **“The Rocket Chip Generator”**
  Electrical Engineering and Computer Sciences, University of California at Berkeley
  Technical Report No. UCB/EECS-2016-17, April 15, 2016
- **“Nios II Processor Reference Guide”**, Intel 2020
  ID: 683836 NII-PRG Version: 2020.10.22
- **“PicoBlaze 8-bit Embedded Microcontroller”**, Xilinx 2011
  User Guide for Extended Spartan-3 and Virtex-5 FPGAs Introducing PicoBlaze for Spartan-6, Virtex-6, and 7 Series FPGAs
- M. Gautschi et al., **“Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices,”**
  IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 25, no. 10, pp. 2700-2713, Oct. 2017.

¡Gracias!