# 3 ARQUITECTURAS RECONFIGURABLES

En esta sección vamos a tratar de manera general que es un Hardware Reconfigurable. Una arquitectura reconfigurable de manera general, es una arquitectura que normalmente contará de un procesador RISC y de una parte reconfigurable (un array o matriz reconfigurable por puntos físicos de control). La idea es usar este tipo de hardware para aplicaciones que realizan volúmenes de cálculos muy grandes, pero para los que se trabaja de forma muy parecida, por lo que es muy probable que se puedan obtener mejoras sustanciales aplicándolas en hardware que puede ejecutar muchos cálculos en paralelo. Esta sección se va a distribuir de la siguiente manera: primero hablaremos de modo general de Hardware Reconfigurable, sus principales características, para que puede venir bien usarlo y los métodos teóricos para pasar código a este tipo de arquitecturas, nombrando algunos de los sistemas más conocidos. Segundo, daremos unos breves ejemplos de algunas de estas arquitecturas, para tener una mayor idea de cómo son en realidad. Tercero, como nuestro proyecto es hacer un algoritmo de renderización, a parte de otras arquitecturas reconfigurables, vamos a hablar del hardware de las tarjetas gráficas, que son el mecanismo más usado y potente en la actualidad a la hora de renderizar. Por último, vamos a hablar de manera mucho más extendida de Morphosys, la arquitectura que usamos en nuestro proyecto. Sus características, su primera versión y una posterior mejorada, los cambios que haremos en ella para adaptarla a nuestro proyecto, como se programa y el uso del simulador de la arquitectura.

## 3.1 Introducción a las arquitecturas reconfigurables

Las arquitecturas reconfigurables han experimentado un importante crecimiento en los últimos años tanto a nivel académico como comercial, particularmente las arquitecturas reconfigurables de grano grueso. De modo general, éstas consisten en una matriz con un cierto número de unidades funcionales (FUs), las cuales son capaces de trabajar a nivel de palabra en lugar de al nivel de bit como encontramos comúnmente en las FPGAs [12]. Esta granularidad reduce ostensiblemente el área de la arquitectura, la potencia que se consume, el tiempo de retardos y el tiempo de configuración comparado con las FPGAs. Otras posibles mejoras de estas arquitecturas son que tengan topologías flexibles, un pequeño espacio de memoria para guardar configuraciones, etc.

El objetivo principal de estas arquitecturas en la actualidad va dirigido a telecomunicaciones y multimedia. Lo que se hace es gastar más tiempo en ejecutar pequeñas partes de código con características bien definidas (kernels), que son partes críticas en lo que se refiere a su tiempo de ejecución. Las partes criticas más comunes son los bucles, difíciles de llevarlos (de mapearlos) a arquitecturas reconfigurables o grandes partes de código muy irregular, es decir con partes muy distintas las unas de las otras, por lo que es difícil plasmarlo en la arquitectura.

El cómputo reconfigurable es un concepto bastante reciente hoy en día, sin embargo, muchas de sus ideas ya las usábamos en procesadores de propósito general, como puede ser usar distintos componentes simultáneamente para procesar partes independientes de código, etc. Sin embargo, cuando hablamos de un sistema reconfigurable, vamos más

allá y nos referimos a sistemas que incorporan algún tipo de hardware programable, normalmente por distintos puntos físicos de control. Estos puntos pueden ir cambiando periódicamente para poder ejecutar distintas aplicaciones en el mismo hardware. De esta manera, y con este tipo de hardware podremos acelerar la ejecución de distintos algoritmos mapeando aquellas partes que conllevan un cómputo intensivo de cálculos.

Para comprender un determinado sistema reconfigurable, debemos atender a varios puntos. Lo primero que hay que examinar es la tecnología sobre la que se va a trabajar, pues los distintos sistemas reconfigurables usan hardware muy distinto, en los que sus bloques y unidades de computación varían mucho de unos a otros. Tan importante como el hardware, es cómo se efectúa la comunicación entre las partes de éste, ya que estos buses contribuyen de manera directa al tamaño global del hardware reconfigurable y a su coste. Una vez se tiene conocimiento de esta parte física, hay que mirar el software requerido para poder compilar algoritmos en estos sistemas y ver si es posible el mapeado automático o si hay que hacerlo manualmente.

Muchas veces, el hardware reconfigurable no es suficiente o no es capaz para llevar acabo la ejecución de aplicaciones enteras, por lo que la mayoría de arquitecturas de grano grueso están compuestas por una matriz reconfigurable y por un procesador, normalmente RISC. Lo que se hace es que algunas partes importantes del programa son llevadas a la matriz para que se ejecuten, mientras que el resto del código, que no puede ser eficientemente acelerado por la parte reconfigurable, es ejecutado por el procesador. Sin embargo aun no se le presta suficiente apoyo y atención a la integración de estas dos partes (matriz reconfigurable y procesador RISC), debido a que la unión entre el procesador y la matriz reconfigurable es a menudo muy débil y difícil, pues consiste esencialmente en las dos partes separadas conectadas por un canal de comunicación. El resultado de esto es una difícil programación de aplicaciones en estas arquitecturas y el exceso de comunicaciones que se puede dar teniendo solo un canal de comunicación.

No solo cambia el diseño del hardware de unos sistemas a otros, sino también el modelo de ejecución de los distintos entornos reconfigurables, por ejemplo el sistema NAPA **[13]** por defecto para la ejecución del procesador principal mientras se hace la ejecución en el hardware reconfigurable, pero permite que se den computaciones simultaneas usando primitivas “fork” y “join”, como en sistemas multiprocesador. Podemos hacer un pequeño resumen de los modelos principales de cómo en los sistemas compuestos por un procesador de propósito general y lógica reconfigurable se comportan cada una de estas partes.

- El hardware reconfigurable puede ser usado únicamente para aportar unidades funcionales reconfigurables al procesador principal [14]. Esto es permitir a la programación tradicional añadir instrucciones que pueden cambiar cada ciertos periodos de tiempo. Para el correcto funcionamiento de las unidades funcionales el procesador principal tiene que estar cada cortos periodos de tiempo indicando que instrucción será ejecutada.
- La unidad reconfigurable se usa como coprocesador [15], de este modo sería más grande que simplemente una unidad funcional reconfigurable en si, pero permite realizar computaciones sin la constante supervisión del procesador principal. Lo que hace éste es proveer los datos o la información necesaria para que el hardware reconfigurable encuentre los datos, así el sistema reconfigurable llevaría acabo sus operaciones al margen del procesador principal, devolviendo

los datos cuando finalice su ejecución. Esto hace que el procesador principal y el hardware reconfigurable puedan estar trabajando simultáneamente.
- La unidad de proceso reconfigurable se comporta como si fuese un procesador adicional en un sistema multiprocesador **[16]**, sin embargo la caché de datos del procesador principal no es visible para la unidad reconfigurable, lo que penaliza la comunicación entre ésta y el procesador principal.
- Por último, nos encontramos con sistemas en los que el hardware reconfigurable es una única unidad de procesamiento externa **[17]**. En este modelo, el hardware reconfigurable se comunica muy infrecuentemente con el procesador principal, cuando éste existe en el sistema.

Para poder pasar una aplicación a una arquitectura hay que seguir unos procedimientos más o menos comunes a todas las arquitecturas reconfigurables. En general, lo primero que necesitaremos es tener la aplicación en un determinado lenguaje de programación (en nuestro caso, y para el proyecto, desarrollaremos nuestro algoritmo de renderización en C++). Una vez teniendo el código, hay hacer una partición del programa o aplicación con la que estemos trabajando en partes, para ser implementadas, unas en el hardware reconfigurable correspondiente y otras que serán implementadas en el procesador principal, de tal manera que la o las secciones que se llevará al hardware reconfigurable debe ser sintetizada y descrita por un circuito de diseño a nivel de transferencia de registros. En este punto, dependiendo del hardware se seguirán distintas estrategias, dependiendo si se disponen herramientas especializadas para compilar, si hay que hacerlo todo manualmente, o una mezcla de ambas. Aquí es cuando nos encontramos con uno de los problemas más comunes que se dan para el desarrollo de esto tipo de arquitecturas, y es que aunque como venimos diciendo, se pueden conseguir buenos beneficios para cierto tipo de aplicaciones, las arquitecturas reconfigurables son a menudo ignoradas por los programadores, a no ser que puedan ser incorporadas de manera sencilla a sus sistemas. Para esto se requieren herramientas de diseño de software para la creación de configuraciones sobre el hardware reconfigurable. Podemos incluir desde un software que simplemente ayude a la creación manual de circuitos o hasta sistemas de diseño que realizan de manera totalmente automática el circuito. El método más potente es el diseño manual, pero en contra requiere un gran conocimiento de la arquitectura particular que se quiera usar. Por otro lado un sistema de compilación automática puede crear rápidamente un circuito para el sistema reconfigurable y lo hacen más accesible a los programadores, aunque tiene la desventaja que los circuitos que crea son menos eficientes que aquellos hechos manualmente. Es muy normal que se de una mezcla de ambas soluciones, por un lado para que no sea tan complicado mapear programas a un hardware reconfigurable y por otro para que no se produzca una caída de potencia como ocurre con el diseño totalmente automático. Para realizar el paso de un programa a una arquitectura hay que seguir un flujo de diseño, que será diferente para cada arquitectura, pero que en general tienen características comunes de unos sistemas a otros. Mostramos a continuación (Fig. 33) como serían tres tipos de flujo de diseño [18] según si se hace de manera automática, manual o con una mezcla de las dos.

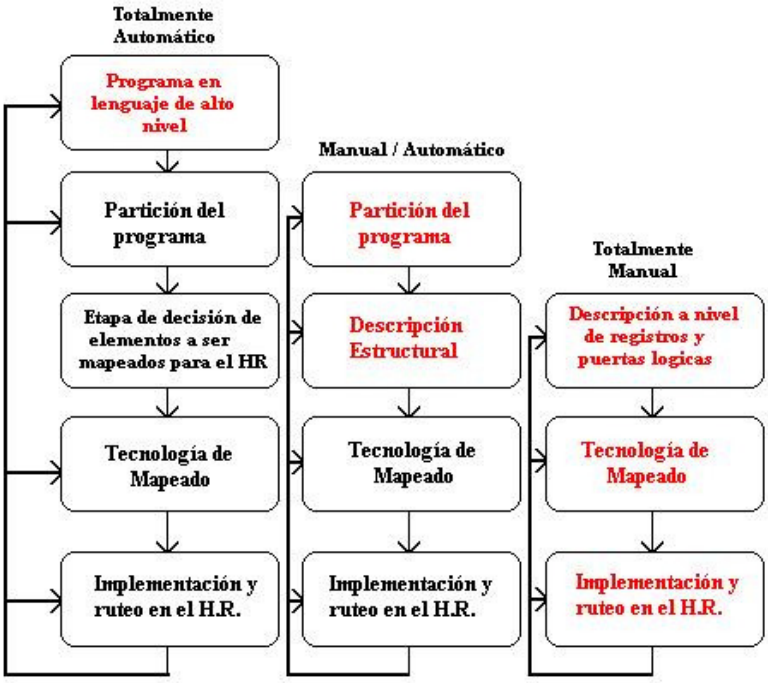


**Figura 33. Flujos de diseño según tipo de mapeado. En rojo fases manuales.**

En todo este tiempo se han propuesto diversas arquitecturas, sin embargo para muy pocas se han diseñado herramientas y metodologías con las cuales se pueda explotar el alto paralelismo. Hay una gran falta de herramientas eficientes para llevar estas partes críticas de código (para *mapear*) a matrices reconfigurables, mientras que algunas no tienen un buen soporte para poder ser diseñadas.

Podemos hablar de distintas arquitecturas reconfigurables. Morphosys (que es la que usamos en nuestro proyecto y que describiremos con más detalle posteriormente) que usa ensamblador para *mapear* manualmente estas partes críticas de código (*kernels*). Aquí el diseñador tiene que identificar y traducir las transferencias de datos entre el RISC y el array de celdas reconfigurables, lo que implica mucho movimiento de datos. RAPID, que soporta C como lenguaje de programación de los *kernels*. RAPID-C está especializado en pipeline, pero requiere muchos conocimientos de diseño acerca de la arquitectura y no es fácil para integrar con ANSI-C para completar le diseño de la aplicación. PACT, que usa un lenguaje NML, que es esencialmente un lenguaje ensamblador, para hacer los *kernels*. Existen herramientas para colocar y enrutar de manera automática a la hora de *mapear los kernels* a PACT XXP. Recientemente PACT ha comenzado a construir sistemas incluyendo un procesador ARM7 y usando un bus AMBA como canal de comunicación, pero aun no tienen establecido un flujo de diseño para una aplicación completa.

Todo esto se completa con otra arquitectura programable con procesadores VLIW (*very long instruction word*), que ya cuentan con muchos mas soportes y tienen muchas herramientas y metodologías de diseño desarrolladas, y dan buenas mejoras. Dentro de estas arquitecturas (procesador VLIW en lugar de un procesador RISC, más una Matriz reconfigurable,) entra ADRES, que veremos a continuación, nombrando sus ventajas sobre sistemas reconfigurables más comunes, como algunos de los antes mencionados.

Para concluir, podemos decir que estas arquitecturas reconfigurables de grano grueso han aventajado en los últimos tiempos a las tradicionales FPGAs. Una de las principales aplicaciones de estas arquitecturas reconfigurables no es solo poder *mapear kernels*, sino también aplicaciones enteras.

## 3.2 Tipos de arquitecturas reconfigurables

En esta sección encontraremos las arquitecturas reconfigurables más conocidas, explicando brevemente su funcionamiento y uso.

### 3.2.1 Arquitectura ADRES

El sistema reconfigurable ADRES (Architecture for Dynamically Reconfigurable Embedded System) [13] ha sido diseñado por miembros de distintas universidades Belgas (Katholieke Universiteit y Vrije Universiteit), junto con algunas herramientas y metodologías de diseño para el mismo.

Aquí vemos el sistema ADRES en general (Fig. 34 A), y el núcleo de este sistema (Fig. 34 B) compuesto por componentes básicos como son Unidades Funcionales (FUs) que pueden almacenar datos intermedios y que con capaces de trabajar a nivel de palabra (en lugar de a nivel de bit) seleccionada por una señal de control y Archivos de Registro (RF) conectados en la topología que nos muestra la imagen.

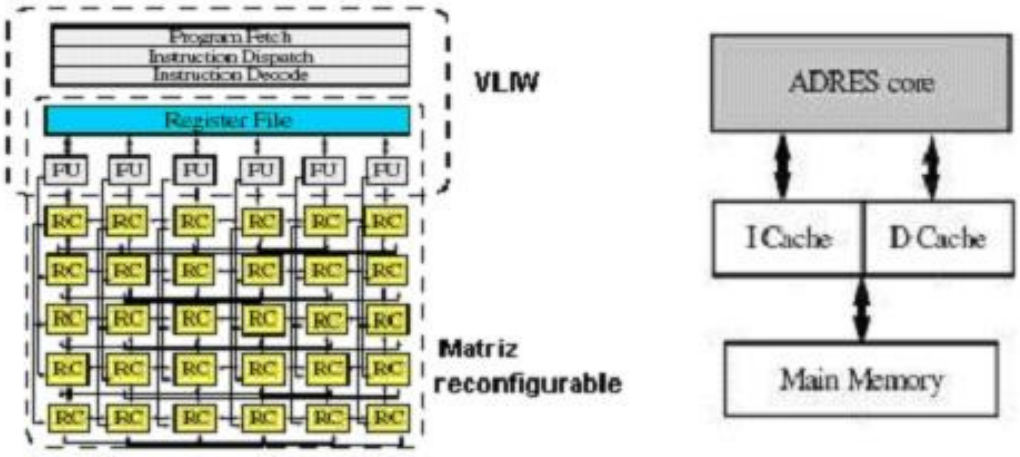


**Figura 34.**

**A: Sistema Adres** **B: Núcleo del sistema**

Como podemos observar, tenemos dos partes funcionales claramente diferenciadas, la matriz reconfigurable y el procesador VLIW. La primera se usa para acelerar el flujo de datos del *kernel* (código crítico) con un alto paralelismo, mientras que el procesador VLIW ejecuta el resto del código tratando de explotar el paralelismo a nivel de instrucción (ILP, *instruction level parallelism*). Estas dos partes funcionales ahorran

algunos recursos, ya que comparten algunas Fus y un RF, sin embargo esto no crea ningún conflicto entre ellas gracias al modelo procesador/co-procesador.

Por la parte del procesador VLIW, encontramos una file de FUs, conectadas todas juntas con un único registro (RF) multipuesto, típico de las arquitecturas VLIW. Estas FUs estarán también conectadas a la memoria. Las FUs del procesador VLIW, comparadas con las partes de la matriz reconfigurables, son mejores en término de velocidad y funcionalidad. Para la matriz reconfigurables, a parte de las FUs y el RF que tienen ambas partes en común, cuenta con muchas celdas reconfigurables.

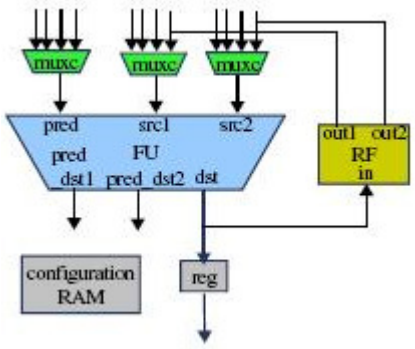


**Figura 35. Celdas reconfigurables**

Básicamente están compuestas por FUs y FRs también. Las FUs pueden soportar distintos conjuntos de operaciones, incluyendo algunas específicas para admitir bucles. Cada celda contara un unos multiplexores, encargados de dirigir los datos desde las diferentes fuentes que le lleguen. Hay una memoria RAM, para guardar contextos locales. La comunicación entre las dos partes funcionales de ADRES se realiza a través del RF que tienen en común ambas.

Gracias a todo esto, ADRES tiene muchas ventajas. Por un lado al tener un sistema VLIW en vez de un sistema RISC, como otros sistemas reconfigurables, con el que acelera el código no crítico. También reduce la complejidad a la hora de programar por el RF compartido y por los accesos a memoria entre VLIW y la matriz reconfigurable, por último, los recursos compartidos reducen notablemente el coste.

### Diseño de flujo C-Based (basado en C)

Para guiar esta breve explicación, veamos como sería el flujo de una arquitectura ADRES.

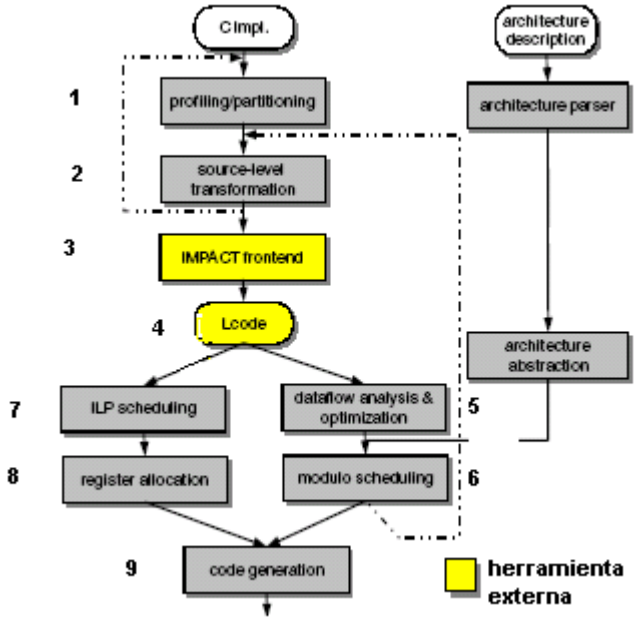


**Figura 36. Etapas del diseño de flujo**

El diseño lo realizamos a partir de una implementación de la aplicación en un lenguaje de alto nivel, en este caso el lenguaje usado es C.

1. Aquí se identifican aquellos bucles candidatos a ser hechos en la matriz reconfigurable, basándose en el tiempo de ejecución y en la posible ganancia que se podría llegar a obtener.
2. En esta etapa se intenta rescribir el código crítico (kernel) para poder aplicar pipeline y maximizar su rendimiento.
3. Este compilador (IMPACT), fue diseñado por los mismos miembros de las universidades belgas antes citadas. Lo hicieron antes que esta arquitectura y posteriormente lo adaptador a ella para poder explotarla mejor. IMPACT se encarga de parsear el código C, analizándolo y optimizándolo, creando un código intermedio (Lcode), que se usa como entrada en la siguiente etapa.
4. Lcode
5. En esta etapa, lo que se hace es describir la arquitectura objetivo (ADRES) mediante un lenguaje XML, y nos queda transformada internamente a un gráfico.
6. En la siguiente fase se toma el código intermedio (Lcode) y el gráfico creado anteriormente como entradas del planificador, y se aplica el algoritmo de planificación que haya para conseguir un gran paralelismo para los kernels.
7. Por este otro lado lo que se hace es aplicar técnicas cotidianas de planificación ILP para obtener un pequeño paralelismo para el código no crítico.
8. Aquí simplemente se asignan que registros se usaran en VLIW para realizar el código no crítico.
9. Y por último se crea un único código planificado, tanto para la matriz reconfigurable como para el procesador VLIW que podrá ser ejecutado en un simulador de nuestra arquitectura.

La mayoría de las etapas se realizan de manera automática con las herramientas que se han creado para el diseño del flujo de ADRES, pero aun hay etapas que

necesitan que el propio diseñador introduzca datos para una mejora más considerable. En particular hablamos de la primera fase, a la hora de realizar el particionamiento. Las decisiones acerca de esto tienen que tomarse en la fase más temprana, principalmente porque todos los requerimientos de optimización son muy diferentes de la matriz reconfigurable al procesador VLIW.

Con la visión de esta arquitectura, y sobre todo del diseño del flujo para una aplicación, y a pesar de las diferencias entre ADRES y Morphosys, al menos conseguimos tener una idea de cómo encaminar nuestro trabajo, distinguiendo distintas fases, y entendiendo mejor a que nos estamos enfrentando. La idea a la hora de plantearnos el desarrollo de nuestro renderizador, hecho en C++, a Morphosys, no será intentar escribir todo el algoritmo con lo que eso conlleva sobre Morphosys, sino estudiarlo, ver distintas estrategias sobre el código escrito en C++ (como podría dividirse, partes comunes, bucles, etc.) y encontrar partes críticas en el código que nos puedan causar problemas y eso llevarlo a Morphosys, planteando las distintas estrategias, y buscando la mejor que encontramos para este hardware reconfigurable.

### 3.2.2 Arquitectura XXP (Extreme Processing Platform)

Esta nueva arquitectura [19] nos presenta una nueva tecnología de procesamiento reconfigurable de datos en tiempo de ejecución, reemplazando el concepto de instrucción secuencial por configuración secuencial y dando estrategias de reconfiguración muy eficientes.

Lo que hay que pensar para hacer buen uso de dispositivos de computación reconfigurable es la manera de separar la "computación" y el proceso de configuración. La estrategia de configuración de XXP nos da un modo revolucionario para solventar este problema. La idea principal del desarrollo de XXP se basa en tres puntos:

**Procesamiento de data stream**

Un data stream es una secuencia de paquetes de datos individuales que viajan a través del grafo de flujo de datos que define el algoritmo. Un paquete de datos en XXP es una palabra máquina de 24 bits, de manera que solo numero y orden de los paquetes que van a través del grafo es importante. Estos streams pueden venir de fuentes naturales como transformadores A/D. Una vez que los datos están en memoria RAM, XXP puede generar direcciones, para producir un data stream. Igualmente, datos calculados pueden ser enviados a distintos destinos como a convertidores D/C o memorias.

**Configuraciones**

Las configuraciones en XXP son módulos de cálculo básico en paralelo, los cuales derivan del flujo de datos del grafo del algoritmo. Los nodos del grafo son mapeados a operaciones máquina básicas, como sumas, multiplicaciones, etc. Los arcos del grafo son las conexiones entre los nodos. Como dijimos antes, se debe separar el procesamiento de datos con la configuración. Para esto, el concepto básico es en

reemplazar los streams de instrucciones Von-Neumann por un stream de configuración y el proceso de streams de los datos.

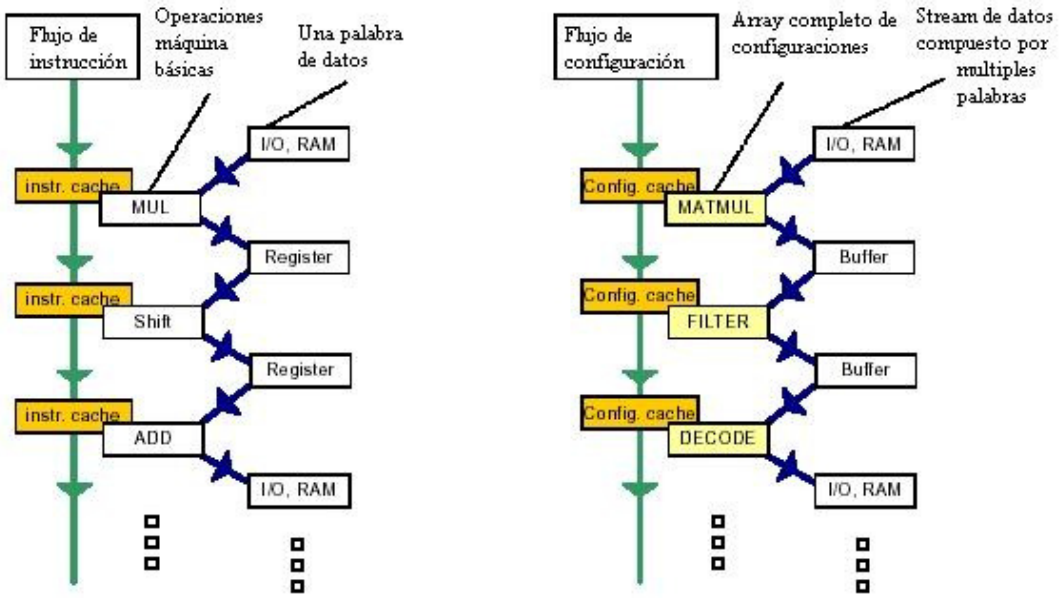


**Figura 37. Paso del flujo de instrucción al flujo de configuración**

Con este modelo de programación, la aplicación se separa en pequeñas secuencias las cuales pueden ser procesadas en paralelo. Conseguimos que una sola configuración, procese un stream de datos.

### Desarrollo de aplicaciones

A parte de la arquitectura, también existe software específico para ella, "XXP development suite" es una herramienta que nos ayuda al desarrollo de programas así como a su corrección. Gracias a su regularidad y simplicidad, un compilador de alto nivel puede extraer instrucciones a nivel de paralelismo y pipelining que esta implícito en los algoritmos.

Todo esto hace que XXP sea una buena elección para el campo de aplicaciones donde grandes streams de datos deben ser procesados. XXP esta bien situado en muchos campos, como procesamiento de imagen y video, visualización en tiempo real…

### Componentes físicos y estructuras XXP

Una arquitectura XXP (Fig. 38), está compuesta básicamente de cuatro componentes. Array de proceso de elementos, organizamos como arrays de procesos (Processing arrays, PAs). Un paquete dirigido a la comunicación interna. Un árbol Manager de Configuración (CM). Y un conjunto de módulos de entrada salida (I/O). Con esto se soporta la ejecución de múltiples flujos de datos funcionando en paralelo.

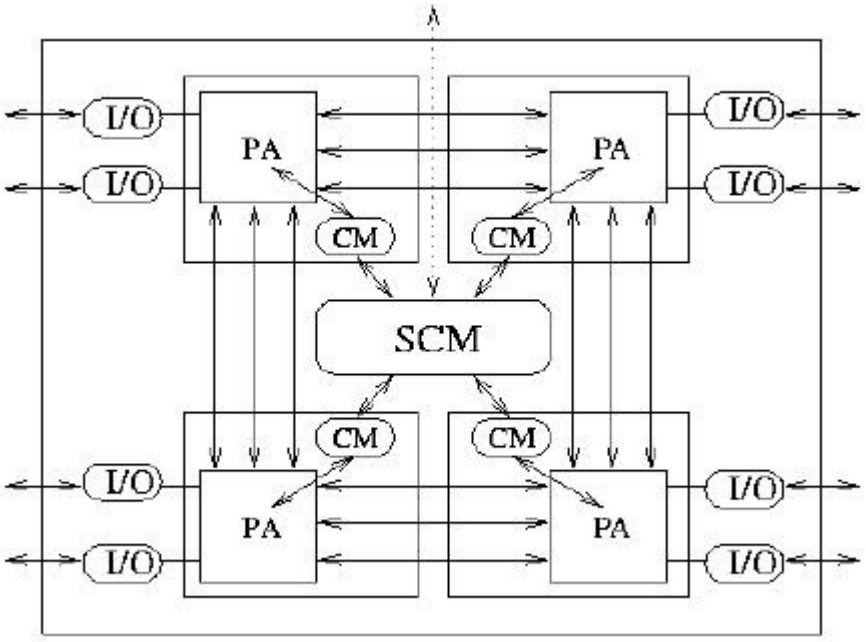


**Figura 38. Arquitectura XXP con cuatro PAs.**

Un array de procesos (PA) y un CM de bajo nivel se agrupa en una estructura llamada PAC (Cluster de array de procesos). El CM de bajo nivel es el encargado de escribir la configuración de datos en los objetos configurables del PA. Lo normal es que haya varios PACs para la construcción de un dispositivo XXP, de manera que según aumenta su número en el diseño, más CMs son añadidos, te manera que forman una estructura en árbol. Al CM que sea el root de dicho árbol se le llama CM Supervisor (SCM), y normalmente va conectado a una memoria RAM externa. En algunos casos el SCM puede actuar como un simple CM, lo que hace que se puedan soportar arquitecturas con varios dispositivos XXP (Fig. 39).

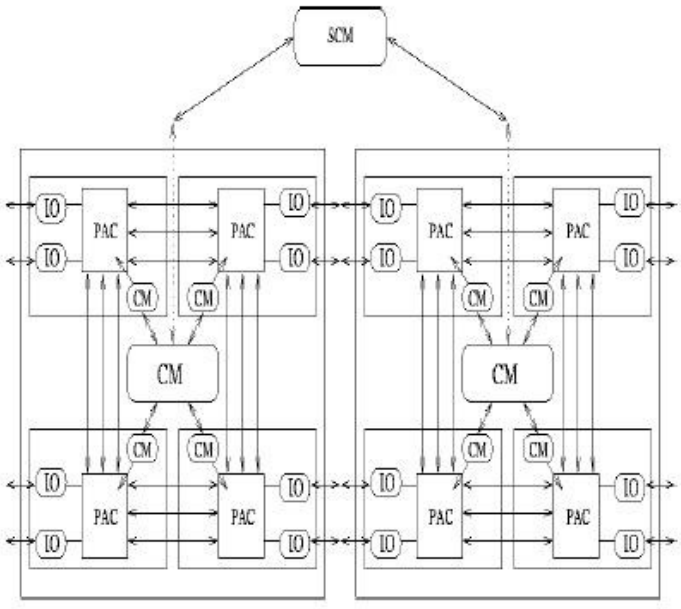


**Figura 39. Múltiples dispositivos XXP unidos por un CM**

Para estos casos de aplicaciones de múltiples dispositivos, el SCM de cada dispositivo XXP actúa como un simple CM, y se añade un SCM adicional que se usa como el root del árbol CM.

Por último presentamos con más detalle la arquitectura y los objetos de un dispositivo XXP (Fig. 40).

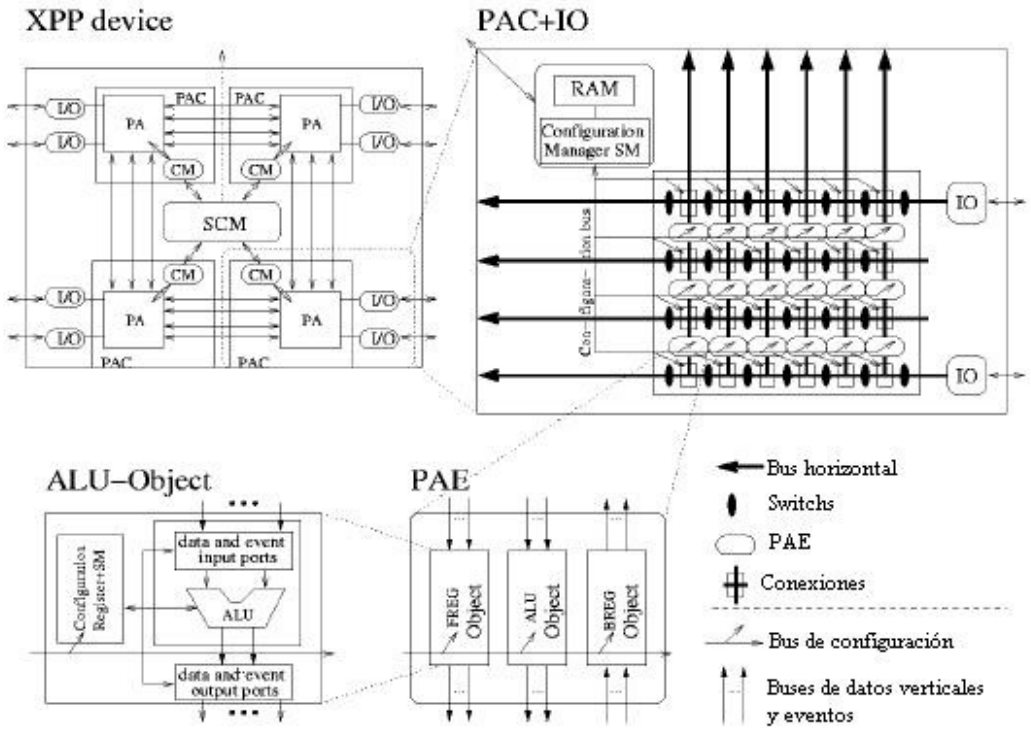


**Figura 40. Estructura y objetos de un dispositivo XXP**

Los buses horizontales son usados para conectar objetos sin un PAE en una fila. Verticalmente, cada objeto puede conectarse a los buses horizontales. Los switches configurables son usados para segmentar las líneas de comunicación horizontalmente. El ruteo vertical se lleva a cabo mediante Registros integrados en el PAE. Los objetos de entrada salida (I/O) son usados para interactuar con variados dispositivos externos, usando un protocolo handshake. Todos estos módulos están conectados por los switches al bus horizontal. El PAE contiene distintos objetos, el registro BREG (backward register), que da facilidades de ruteo de abajo a arriba y funciones aritméticas, el registro FREG (forward register), que proporciona recursos de ruteo en la dirección contraria que el anterior, y funciones para el control del flujo de datos, y normalmente también cuenta con un objeto ALU, que se basa en la ALU en si misma, con sus puertos de entrada y salida, una máquina de estado (SM) para el control de la ejecución y un CM para controlar la conexión.

Los puertos de entrada y salida pueden tanto recibir como transmitir paquetes de datos y eventos. Los paquetes de datos suelen ser procesados por las ALUs, mientras que los paquetes de evento son procesados por la máquina de estados que controla la ejecución.

Como dijimos en la introducción de hardware reconfigurable, para especificar algoritmos para XXP, usamos un lenguaje específico, NML (Native Mapping

Language). La herramienta "XXP development suite" (XDS) diseña el flujo para la programación sobre XXP según muestra la siguiente figura (Fig. 41)

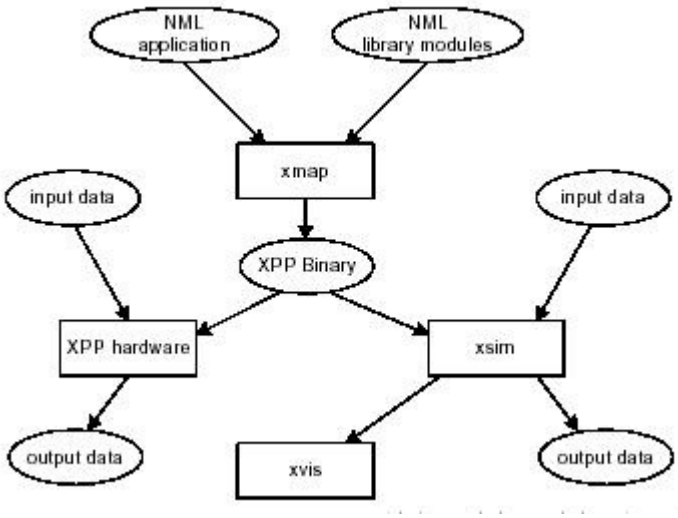


**Figura 41. Diseño de Flujo XDS**

Esta herramienta hace el diseño tanto para la arquitectura, como para el simulador que existe de la misma (XSIM). El algoritmo que obtenemos en XXP binario, puede ser llevado directamente a su ejecución sobre el hardware o sobre el simulador.

### 3.2.3 Procesador SMeXXP Media

Como dijimos en la introducción, últimamente PACT ha desarrollado sistemas XXP incluyendo además un procesador ARM7 [20], memorias RAMS, y distintos periféricos. Está desarrollado con orientación a aplicaciones multimedia (como por ejemplo un encoder MPEG4 [20]), Veamos su estructura con la siguiente figura (Fig. 42)

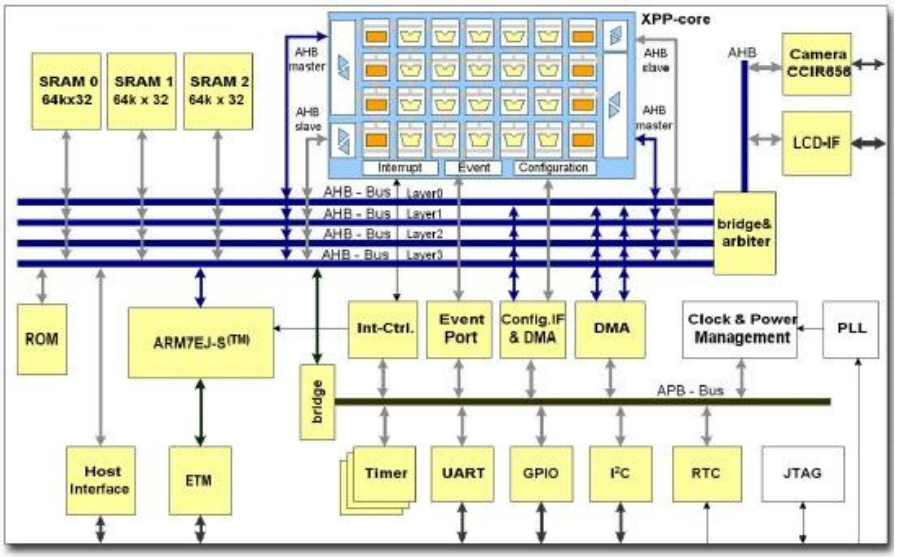


**Figura 42. Procesador SMeXXP**

El corazón de esta estructura son los buses AMBA, que conforman un bus AHB que conecta todos los componentes como se muestra en la figura anterior. Veamos de manera resumida los componentes.

### ARM7

Es un poderoso procesador RISC de 32 bits. Este procesador funciona de manera excelente para todas las tareas y control de flujos orientados a tareas de aplicaciones de video. Su rango de reloj de frecuencia oscila entre los 13 y los 104 MHz.

### Memoria

Hay tres bancos de memorias RAMS disponibles tanto para el procesador ARM7 como para el núcleo XPP. Los bancos proveen accesos alineados e independientes de 32 bits desde los buses AHB.

### Controlador DMA

Hay dos canales DMA para modos de transferencias a través de los buses AHB. El DMA genera solo secuencias de direcciones lineares. Para casos de direcciones más complejas, la mejor opción es usar el núcleo XXP.

### Periféricos

Los periféricos típicos requeridos para ayudar al tipo de aplicaciones hacia los que va dirigida esta arquitectura están integrados en el chip. El bus-APB (bus de periféricos) está conectado por un puente al bus-AHB. Algunos de estos periféricos son el Timer (temporizadores), RTC (tiempo real de reloj), puertos de configuración XXP, etc.

### Núcleo XXP

Es un núcleo XXP parecido al descrito anteriormente, de 16 bits. Está formado por un array de 5 x  4 PAEs, 2 x 4 RAM PAEs, dos interfaces maestros de entrada salida AHB, y otros dos interfaces esclavos. (Fig 43)


**Figura 43. Núcleo XPP del procesador SMeXXP**

Donde los módulos PAEs ya los vimos descritos anteriormente y los RAM- PAEs, son iguales, pero en vez de tener una ALU como elemento principal, tienen una memoria de 512 x 16 bits, aparte del registro FREG y BREG.

### 3.2.4 Arquitectura RaPiD

El proyecto RaPiD [21] ha estado explorando una arquitectura de computo configurable denominada RaPiD (Reconfigurable Pipelined Datapath), que esta optimizado para el dominio de procesamiento de señales e imágenes. El objetivo de esta arquitectura es juntar una alta mejora para las características de estos dominios unido a un bajo consumo tanto económico como energético.

RaPiD está optimizado para operaciones aritméticas sobre valores de datos de muchos bits. Hay un ahorro muy grande si se utilizan unidades específicas de computación optimizadas para palabras de la longitud usada en estos cómputos. El camino de datos de RaPiD soporta un alto nivel de paralelismo. En particular los algoritmos "sistólicos" están muy bien situados en esta arquitectura. El camino de datos de RaPiD implementa la entrada/salida usando múltiples stream de datos independientes. Estos streams van conectados directamente a dispositivos de streams externos o a un sistema stream de memoria. RaPiD coge ventaja de la computación tan repetitiva y regular que se da en estos dominios para general las señales de control que controlen el camino de datos por medio de una ruta de datos muy eficiente.

Según los estudios realizados por miembros de la universidad de Washington, RaPiD puede conseguir hasta un rendimiento de 1.5 billones de operaciones de acumulación o multiplicación por segundo para un buen número de importantes aplicaciones usando un array de un tamaño moderado ejecutándose a 100 MHz.

La arquitectura RaPiD normalmente esta formada por cientos de unidades funcionales, desde simples registros, hasta multiplexores, desplazadores, ALUs y memorias. Estas unidades suelen estar colocadas de manera lineal (Fig. 44) y conectadas usando interconexiones configurables basadas en buses segmentados. Tanto los buses, como las unidades funcionales de esta arquitectura configurable estás preparadas para trabajar a nivel de palabras, en lugar de bits. Esta alineación de las unidades funcionales de la arquitectura puede parecer algo limitada. La clave está en como mapear algoritmos multidimensionales a arrays lineares [22].

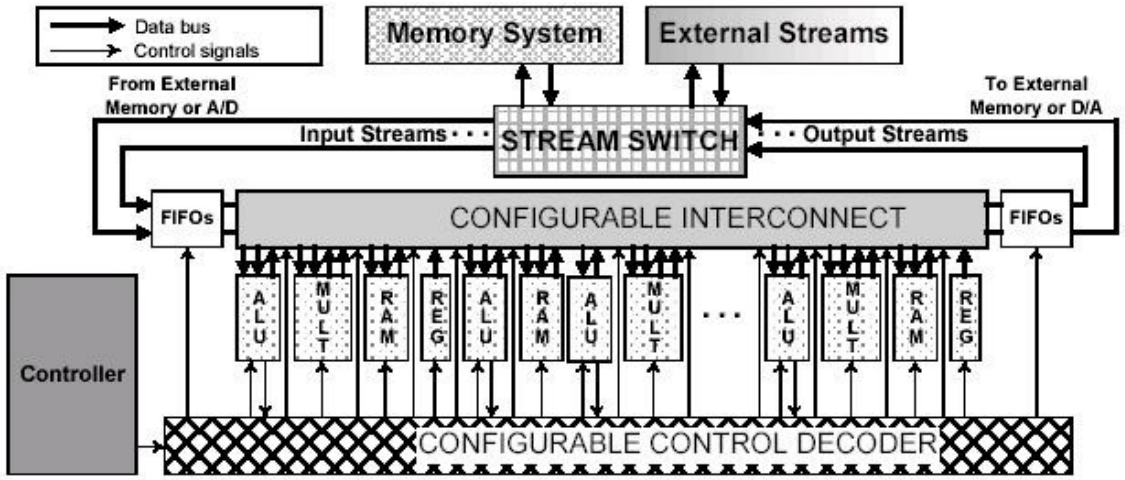


**Figura 44. Estructura de RaPiD**

En RaPiD, cada unidad funcional recibe los valores de los datos de los buses seleccionados en la interconexión configurable, obteniendo una computación específica por un conjunto de señales de control y resultados de salida en la forma de valores de datos.

A parte de las unidades funcionales antes mencionadas, otras más específicas pueden ser añadidas a la arquitectura RaPiD. Lo más normal, para dominios específicos es añadir una unidad funcional de propósito general, que ayude a mejorar funciones individuales que no tienen entradas de control.

La interconexión configurable consiste en un conjunto de buses segmentados colocados al lo largo de toda la arquitectura. Algunos de estos buses segmentados están unidos por conectores que pueden ser configurados para conectarse con segmentos adyacentes creando segmentos más largos, con la posibilidad de llevar a cabo pipelining. Todos los buses tienen el mismo ancho.

Cada entrada a una unidad funcional selecciona uno de los buses de la interconexión mediante un multiplexor, que opera según un conjunto de señales de control. Los datos de salida son conducidos por buses seleccionados también por señales de control.

La operación que realizará RaPiD en cada momento vendrá dado por las señales de control que determinarán que operaciones hace cada unidad funcional y como los datos serán conducidos por la interconexión configurable a través de las distintas unidades funcionales. Un conjunto entero de señales de control dentro de un ciclo de reloj es llamado "datapath intruction" y un programa en RaPiD es una secuencia de estas instrucciones que hacen la computación deseada. Una arquitectura RaPiD típica, puede contener de orden de unas 100 unidades funcionales, requiriendo para poder controlar todas más de 5000 señales de control.

Seguramente, el problema más común en estas arquitecturas reconfigurables es la generación de las señales de control que controlen la ejecución sobre la arquitectura, ya que el alto nivel de paralelismo unido a la reconfigurabilidad genera un gran número de potenciales señales de control. Generar y conducir estas señales según los ciclos de reloj

por la ruta de datos es muy costoso tanto en superficie para la arquitectura como en consumo. RaPiD solventa este problema dividiendo las señales de control en dos tipos, duras (traducido literalmente, hard) y suaves (soft). Las primeras, comúnmente son señales de configuración que distribuye la memoria estática que cambia muy pocas veces cuando la arquitectura es reconfigurada para diferentes aplicaciones. Las segundas, cambian cada ciclo de reloj dirigido por un programa de control.

RaPiD es programado usando un lenguaje llamado RaPiD-C que está especializado en algoritmos en paralelo. Los programas en RaPiD-C son compilados en archivos de configuración y programas que los controlan por el proceso descrito a continuación (Fig. 45).

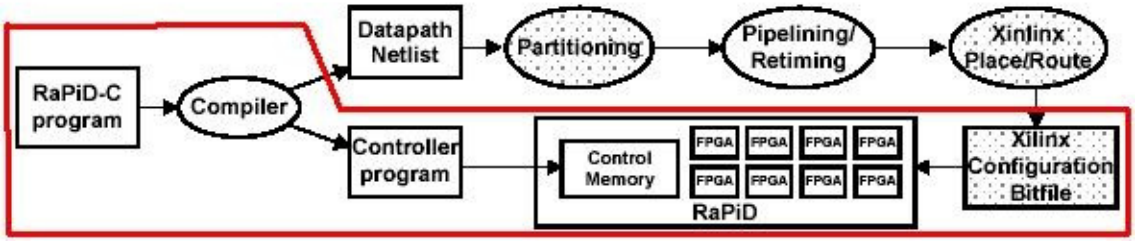


**Figura 45. Herramienta de flujo CAD para compilación**

Aquí, el compilador analiza en programa en RaPiD-C, una vez analizado lleva a cabo una partición de programa, para cada una de las unidades funcionales, para realizar de manera eficiente las operaciones de datos y el programa de control, el cual genera las señales de control que operarán la arquitectura. Todo esto se hace según la parte señalada dentro del cuadro de la imagen anterior (Fig. 45). El otro flujo que se muestra, es el flujo aumentado para soportar también la compilación de RaPiD-C para el simulador de esta arquitectura, donde será el programador en este caso, el que realice la partición, indicando que operaciones se harán en cada chip.