VII Workshop IBERCHIP. IBERCHIP, Montevideo, 2001.

Diseño de microcontroladores empotrados mediante procesamiento serial: análisis usando FLEX10K para sintetizar un microcontrolador tipo COP8Sax.

Jaquenod, Guillermo Adolfo y De Giusti, Marisa Raquel.

Diseño de microcontroladores empotrados mediante procesamiento serial: análisis usando FLEX10K para sintetizar un microcontrolador tipo COP8SAx

Autores: Guillermo A. Jaquenod (*) y Marisa R. De Giusti (**)

Abstract:

When designing high performance digital systems it's usually needed to solve some slow tasks, such as serial asynchronous channels, keyboard scanning, or display multiplexing; today, embedded standard microcontrollers can be included as a macrofunction inside the FPL (Field Programmable Logic) core to solve these low speed tasks in a sequential way, with lower hardware requirements and gain on versatility. Since hardware is every day faster, new techniques can be used when designing these processors, and serial processing must be considered.

This paper describes some alternative design approaches for serial CPUs using ALTERA FLEX10K devices; serial designs require inherent low connectivity and they are easy to route using spare resources of the chip.

Resumen:

La incorporación de microcontroladores empotrados dentro de diseños complejos basados en lógica programable ha resultado ser una alternativa de uso habitual cuando, junto con una tarea de elevada performance, es necesario atender la interfase al usuario u otras aplicaciones de baja velocidad como teclados, displays, o bocas de comunicación asincrónicas. La elevada velocidad de operación de las nuevas familias de lógica programable permite considerar nuevas alternativas de diseño, y el uso de procesamiento serial es una de estas nuevas posibilidades.

Este artículo describe algunas alternativas de diseño de CPU seriales usando dispositivos de la familia FLEX10K de ALTERA; estos diseños consumen escasos recursos de conexionado, facilitando un ruteado del diseño que permite aprovechar la capacidad remanente dispersa por el chip.

(*) Profesor Titular, Facultad de Ingeniería, Universidad Nacional del Centro de la Provincia de Buenos Aires.
Dirección postal: A.Del Valle 5737 – (7400) OLAVARRÍA – ARGENTINA
Correo Electrónico: <*chipi@satlink.com*>

(**) Investigador Adjunto sin Director – Comisión de Investigaciones Científicas de la Provincia de Buenos Aires. Profesor Titular, Facultad de Ingeniería, Universidad Nacional de La Plata.
Dirección postal: Calle 24 # 709 – (1900) LA PLATA – ARGENTINA
Correo Electrónico: <*marisadg@volta.ing.unlp.edu.ar*>

Diseño de microcontroladores empotrados mediante procesamiento serial: análisis usando FLEX10K para sintetizar un microcontrolador tipo COP8SAx

Autores: Guillermo A. Jaquenod y Marisa R. De Giusti

(*) Universidad Nacional del Centro de la Provincia de Buenos Aires.
(**) Comisión de Investigaciones Científicas de la Provincia de Buenos Aires. Univ.Nac. de La Plata

1. Introducción

El uso de procesadores secuenciales empotrados dentro de diseños complejos es una alternativa de uso habitual cuando, junto con una tarea de elevada performance, es necesario atender aplicaciones de baja velocidad y/o complejidad algorítmica tal que hacen conveniente cierto tipo de procesamiento secuencial.

Las soluciones que se encuentran hoy día [10] se diferencian en dos grandes grupos:

- microcontroladores simples: del tipo 6805[8], 6502 u 8051[1], o similares, cuya ventaja comparativa más importante es el uso escaso de recursos, y cuya aplicación típica es la interfase al usuario u otras aplicaciones de baja velocidad como teclados, displays, o bocas de comunicación asincrónicas. En todos los casos son diseños que se compilan junto al diseño del usuario ("*soft-cores"*).
- procesadores de alta performance: como NIOS[4], ARM[2], MIPS [3], orientados a tareas de procesamiento complejo, de alta performance, que consumen importante cantidad de recursos lógicos. Mientras que el procesador NIOS es un *soft-core* tanto ARM como MIPS corresponden a soluciones de hardware que ofrecen un procesador predefinido ya cableado en un dado chip (*"hard-core"*).

El diseño serial se adapta especialmente al diseño de microcontroladores simples, pues disminuye en forma importante los requerimientos de ruteado (conexiones y llaves) a costa de requerir varios ciclos de reloj por cada byte que se transfiere; sin embargo, esta penalidad temporal es cada vez menos importante gracias a la elevada velocidad de reloj que toleran las nuevas familias de lógica programable. A su vez, la menor exigencia de recursos de conexionado posibilita distribuir un diseño a lo largo de un chip y aprovechar los huecos que el sistema de Place&Route ha dejado libres al rutear las etapas de alta performance.

El compromiso de diseño entre velocidad de operación y uso de recursos de conexionado no ocurre sólo dentro del área de la lógica programable: por ejemplo, la familia de microcontroladores de arquitectura Harvard modificada COP8SAx de National [11, 12, 13] opera con un reloj de 10MHz, a razón de un ciclo de instrucción cada 10 ciclos de reloj, y aunque su operación no presenta diferencias con la de otros dispositivos de 8 bits donde el procesamiento es en paralelo, internamente la operación es serial (en cada ciclo de instrucción se realiza la serialización de 8 bits). La disminución de la complejidad de ruteado y la menor cantidad de líneas que conmutan simultáneamente determina que estos procesadores sean más económicos y generen un nivel de interferencia electromagnética (EMI) hasta 20dB menor al de procesadores similares operando a igual número de MIPs. Otro ejemplo interesante es el de los Transputers, donde el uso de conexiones serie (*Serial Links*) entre procesadores facilita la conectividad [15].

Este artículo presenta ciertas ideas generales de diseño de procesadores seriales [6] y analiza las ventajas arquitectónicas de la familia FLEX10K[5] de ALTERA para este tipo de diseños, tomando como caso particular para los ejemplos ciertos módulos funcionales de un procesador tipo COP8SAx. Además, busca mostrar cómo a través de un diseño cuidadoso que considere las características de la familia de FPL a emplear, es posible lograr una gran eficiencia de uso de recursos.

2. Consideraciones generales para el diseño de procesadores

Al diseñar un dado procesador, para intentar llegar a un diseño óptimo se hace necesario analizar tanto los distintos actores de la CPU (registros, memorias, unidades de cálculo, unidades de direccionamiento, módulos de decodificación) como el flujo de información (de instrucciones, datos y control) requerido entre dichos actores durante la ejecución de las posibles instrucciones [7]. Hecho este análisis, deben considerarse las ventajas arquitectónicas que ofrece la tecnología a emplear, teniendo en cuenta que ciertas soluciones se acomodan mejor que otras a cierto tipo de arquitecturas

En un procesador tipo HARVARD como el COP8SAx pueden identificarse como actores a los registros A, B, X y SP, los dispositivos de I/O, la memoria de datos (DM) y la memoria de programa (PM), y ciertos bloques funcionales necesarios para decodificación de instrucciones (ID) y generación de direcciones (AG). A diferencia del 6805, en un COP8SAx los registros B, X, SP y la I/O son también direccionables en el espacio de datos.

Para la interconexión de estos actores existen varios buses, cada uno de los cuales presenta características particulares:

- El *bus de direcciones de programa (PAB)* tiene un único origen (AG) y un único destino (PM), y puede ser de 10 a 12 bits. En una aplicación de FPL conviene que el ancho de este bus sea parametrizable según los requerimientos de dicha aplicación.
- El *bus de contenidos de programa (PCB)* tiene un único origen (PM) y como posibles destinos al ID, al AG, al DAB o al DCB.
- El *bus de direcciones de datos (DAB)* es un bus de 8 bits que tiene como posibles orígenes al PM (direccionamiento directo), a los registros B o X (direccionamiento indexado), o al SP (manejo de la pila) , y como posible destino al DM, I/O, B, X y SP.
- El *bus de datos (DCB)* transporta información desde el PM, A, o el espacio de memoria de datos hacia A o al espacio de memoria de datos. La complejidad de este flujo de información está dada por la existencia de múltiples actores mapeados en este espacio de memoria de datos (múltiples orígenes, múltiples destinos) así como por la existencia de ciertas instrucciones (***X-eXchange***) donde dos datos se intercambian a la vez, entre A y el otro actor.

Los buses internos de un procesador pueden resolverse mediante buses bidireccionales y buffers tristate o mediante multiplexores [7, 14], y cada una de estas soluciones presenta ciertas ventajas y desventajas:

- El empleo de buses bidireccionales y buffers tri-state disminuye el uso de recursos de cableado, pero genera el riesgo de "bus-contention" (dos salidas excitando el bus simultáneamente con valores opuestos), haciendo necesario considerar tiempos de guarda y circuitos de seguridad que a veces anulan las ventajas de cableado.
- El ruteado de buses unidireccionales mediante multiplexores es inherentemente seguro, más rápido, y permite cierto grado de concurrencia, aunque al precio del consumo de mayores recursos de cableado.

3. Consideraciones de diseño usando FLEX10K

Al realizar el diseño de un dispositivo usando la familia FLEX10K de ALTERA debe tenerse en cuenta su arquitectura:

3.1. Los elementos lógicos (LEs): son los elementos constructivos elementales (*"atómicos"*) de la familia FLEX, y se componen de una etapa combinatoria basada en el uso de tablas (o ***LUT***) de 16 bits (4 entradas) seguida de un registro opcional. Las reducidas dimensiones de cada LE y la gran cantidad de ellos hacen que la familia FLEX10K se preste a diseños que hacen uso intensivo de registros.
Además, los LEs poseen la capacidad de configurarse en un modo llamado *aritmético*, donde una función auxiliar llamada CARRY CHAIN permite alta eficiencia de propagación de Carry entre LEs contiguos para la realización de sumadores, contadores u otros circuitos numéricos. También ofrecen una función llamada CASCADE CHAIN que permite realizar el AND lógico de las salidas de LEs contiguos, y así facilitar el cálculo de ciertas funciones de muchas variables (gran *fan-in*).
Al planear un diseño, y previo a ir a detalles de diseño, conviene por lo tanto considerar como lograr el máximo aprovechamiento de estos elementos "atómicos".

3.2. Recursos de cableado: en las FLEX10K existe una jerarquía de conectividad de varios niveles, compuesta por la conexión entre vecinos inmediatos (las mencionadas CARRY y CASCADE), a nivel de un grupo de LE (el LAB), y a nivel global (matriz de interconexión FastTrack)

Un grupo de ocho LEs, con facilidades especiales de conexión entre sí, es llamado LAB (*Logic Array Block*), y en muchos casos es conveniente agrupar ciertas funciones en un LAB para el mejor uso de estos recursos locales de conexión, con mejoras de velocidad y menor compromiso de los recursos globales.

La matriz *FastTrack* se compone de un conjunto de líneas, en forma de filas y columnas, que atraviesan todo el chip, permitiendo la interconexión entre LEs de distintos LABs. Cierto tipo de conexiones son más ventajosas que otras al usar este recurso, y estas ventajas son usadas por el compilador cuando agrupa funciones.

La familia FLEX10K no ofrece buffers tristate internos y todo el ruteado debe realizarse mediante multiplexores; para ello es de sumo valor el uso del CASCADE, pues permite realizar un multiplexor 4:1 con sólo 2 LEs, y un multiplexor 8:1 con sólo 5 LEs.

3.3. Los EABs: dentro de cada FLEX10K existen múltiples bloques de memoria RAM, llamados EAB (Embedded Array Blocks) cada uno de 2048 bits, cuyo formato puede ser definido individualmente como 256 palabras de 8 bits, 512 palabras de 4 bits, 1024 palabras de 2 bits, o 2048 palabras de 1 bit. Cada uno de estos bloques funciona separadamente, y su contenido inicial puede ser definido en la configuración (emulando una ROM).

4. Una primer aproximación a la CPU. Criterios globales de diseño:

Para evaluar las posibles ventajas del uso de una arquitectura serial, paralela o mixta, deben definirse metas de performance, la topología de conexión y los posibles bloques constructivos.

4.1. Metas de performance:

El primer parámetro de diseño de un procesador empotrado es la performance deseada [7], que puede medirse en MIPS (millones de instrucciones por segundo), en consumo de recursos (LEs y EABs en el caso de los FLEX10K), en consumo de energía, en generación de EMI, etc. Por el tipo de procesador bajo análisis en este artículo es claro que en este caso el enfoque es minimizar el uso de recursos, bastando una velocidad de operación comparable a la de un COP8SAx estándar, de 1 MIPS, con un reloj de 10 MHz. Este parámetro determina un largo período de reloj (100ns), permitiendo el diseño de bloques combinatorios con 10 o más LEs en serie, incluso en el caso de los FLEX10K más lentos, sin necesidad de pensar en soluciones tipo "pipeline" o similares.

4.2. Topología de buses:

Se sugiere cierta forma de implementar los distintos buses, cuya justificación se ve a medida que se detallan los distintos bloques constructivos:

- que *el bus de datos (DCB)* sea realizado en forma serial, dado que es el de mayor complejidad por la variedad de orígenes y destinos posibles.
- que el *bus de direcciones de programa (PAB)* sea distribuido en paralelo, dado que tiene un único origen y un único destino posibles, bastando asegurar la proximidad física de los módulos AG y PM. Además, si el PM es realizado usando EABs, queda predefinida la necesidad de un direccionamiento en paralelo.
- que el *bus de direcciones de datos (DAB)* sea también distribuido en paralelo, dado que direcciona al EAB usado como memoria de datos. En función del valor de este bus de 8 bits de ancho se generarán las líneas de selección de registros (RSi) que irán en forma individual a cada uno de los registros mapeados en memoria de datos, con sólo 2 LEs en modo CASCADE por RSi.
- que el *bus de contenidos de programa (PCB)* sea de tipo mixto, y llegue en forma paralela a los módulos relacionados con el PM (ID y AG) y a las líneas de direcciones del EAB usado para memoria de datos, y en forma serial (PCBS) a los demás actores conectados por el DCB.

4.3. El ciclo serial:

Al haber elegido la utilización de un bus serial se hace necesario determinar el formato temporal de transferencia de datos. En forma similar a lo que sucede en el COP8SAx, se define un ciclo de instrucción compuesto por 10 ciclos de reloj, llamados PRELD, TBIT0..TBIT7, TXFR con las siguientes funciones:

- ***PRELD***: prepara los registros de desplazamiento serial para su carga paralela en la siguiente transición de reloj
- ***TBIT0..TBIT7:*** presenta en el respectivo bus serial el dato a transmitir, comenzando por el menos significativo

- ***TXFR***: transfiere los datos ingresados al registro de desplazamiento al registro paralelo.

Como se verá al analizar los distintos módulos, es conveniente distribuir 3 líneas globales de control (BITSEL[2..0]) para identificar los ciclos TBIT0..TBIT7, así como líneas que decodifican algunos de estos estados (BIT0 y BIT4).

4.4. Módulos constructivos:

Definida la estructura de los buses y los ciclos de transferencia es posible comenzar a especificar la posible arquitectura de los bloques funcionales:

4.4.1. Puerta de entrada:

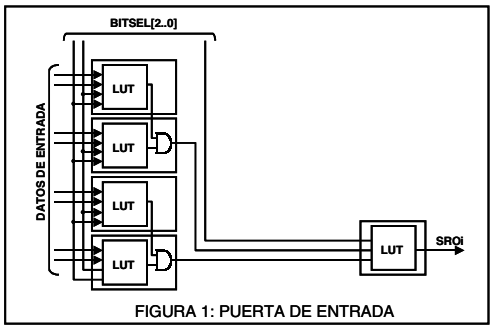

FIGURA 1: PUERTA DE ENTRADA

El bloque constructivo más simple de tipo serial es el de una puerta de entrada (figura 1), que sólo puede ser leída, y para su realización sólo se requiere un multiplexor 8:1, que en forma secuencial va poniendo sobre el bus de datos serial DCB el valor de los sucesivos bits de entrada,; usando FLEX10K este módulo sólo necesita 5 LEs.

4.4.2. Puerta de salida:

El bloque constructivo de complejidad siguiente es el de un registro de salida (figura 2), cuyo contenido puede ser leído pero también escrito en forma serial.

En este caso se observa cómo un multiplexor de entrada selecciona el posible origen de los datos a cargar (se muestra un multiplexor 2:1 porque en los COP8SAx sólo es posible escribir un registro con un dato inmediato, que viene en forma serial desde el PM (PCBS), o con el contenido del acumulador A), y a medida que los datos ingresan al registro, los datos previos egresan por SROi, posibilitando la ejecución de la instrucción ***eXchange***.

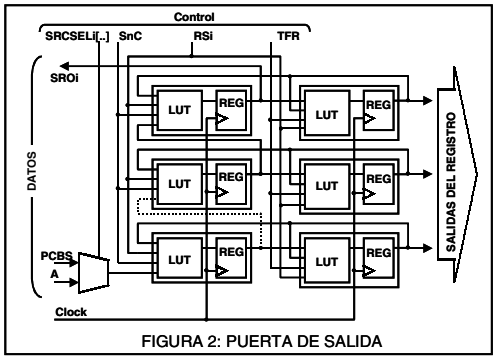

FIGURA 2: PUERTA DE SALIDA

La complejidad de un registro de salida de 8 bits es de sólo 19 LEs, dados por 16 LEs para los registros, más los requeridos para la selección del origen de datos (en este caso, sólo un LE adicional), más los usados para la generación de la señal de selección RSi (no mostrados en la figura), que al igual que en el módulo de entrada son sólo 2 LEs más en modo CASCADE.

Existen varios elementos importantes a destacar:

- es un diseño absolutamente sincrónico: a todos los registros llega un único reloj
- el esquema de "*double buffering* o *Shift/Store*" asegura que durante el proceso de carga los datos de salida permanezcan inmutables.
- para el control del bloque sólo se necesitan dos señales globales (SnC y TFR) cuya acción está habilitada por RSi.

4.4.3. Una puerta bidireccional:

Una puerta bidireccional es una combinación de una puerta de entrada y dos puertas de salida, para los datos de salida, control de Tri-State, y eventuales resistencias de pull-up/pull-down. Acá debe considerarse que el caso de un microcontrolador empotrado es distinto al de un microcontrolador estándar, donde todas las puertas deben ser bidireccionales para que cada usuario las configure según su aplicación; en un microcontrolador empotrado el diseñador sabe exactamente cuantas puertas necesita y de qué tipo, por lo que una puerta bidireccional "estándar" carece de sentido.

4.4.4. Registros B, X y StackPointer SP:

Estos registros de índice (figura 3) tienen grandes similitudes con una puerta de salida, sólo que los valores de salida son usados internamente en vez de ir a una pata de I/O.

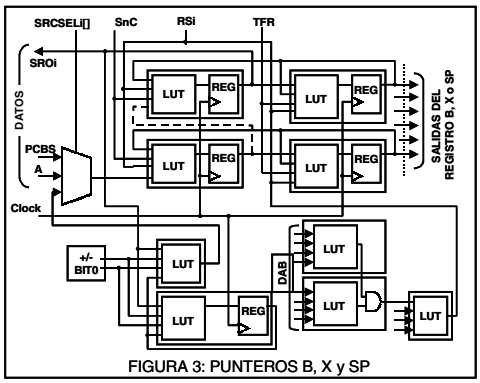

FIGURA 3: PUNTEROS B, X y SP

Desde el punto de vista circuital requieren del agregado de nuevos elementos para satisfacer las funciones de auto-incremento y auto-decremento, con lo que cada uno de los registros B, X y SP consume 23 LEs (4 LEs más que un registro de salida):

- 16 LEs para los 8 bits "double buffered" del registro, similar a un registro de salida.
- 2 LEs para un sumador serial tipo "*carry save*" [9], donde el acarreo producido en la suma de un bit es almacenado en un registro para ser usado en la suma del bit siguiente. En estos registros se da la circunstancia que uno de los sumandos es una entre dos posibles constantes (según deba hacerse el incremento o decremento), por lo que es controlado por una línea global de control (BIT0) y una línea de control (+/-) que resulta de la decodificación de instrucciones.
- 2 LEs para un multiplexor 3:1, que ahora además de PM y A agrega como posible fuente de datos a la realimentación del propio registro, ya incrementado o decrementado.
- 3 LEs para la generación de RSi. En este caso, se necesita un LE extra para decodificación pues los registros B o X no sólo son modificados cuando se accede a ellos en el mapa de memoria, sino también de modo complementario durante las instrucciones en que se produce su incremento o decremento automático.

4.4.5. El acumulador A:

El acumulador A (figura 4) es similar a un registro de salida, siendo la única gran diferencia la cantidad de posibles fuentes de entrada:

- la memoria de programa (PCBS), para el caso de la carga inmediata
- la memoria de datos (DCBS), para el caso de ***LoaD/eXchange*** directo
- los registros mapeados en memoria de datos, para el caso de ***LoaD/eXchange*** directo
- la ALU, como resultado de operaciones aritméticas, lógicas o de shift.

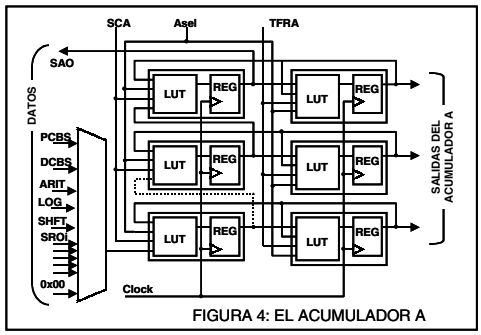

FIGURA 4: EL ACUMULADOR A

Este gran abanico de entrada es donde mayores recaudos deben tomarse si se desea mayor eficiencia. En este caso, más que un multiplexor estándar, puede convenir el uso de un esquema GATED OR (suma de ANDs) que es realizado en forma muy eficiente y veloz mediante LEs en modo CASCADE, a razón de un LE cada dos entradas.

Al analizar el acumulador es importante notar que si los datos inrgesados al registro de desplazamiento (*shift*) no son transferidos a los registros de salida (*store*), es posible usar a este módulo también como registro temporario, sin modificar el valor de A, para operaciones RMW sobre memoria que no involucran al acumulador (como SBIT/RBIT y DRSZ).

4.4.6. La memoria de datos y la generación de direcciones de datos:

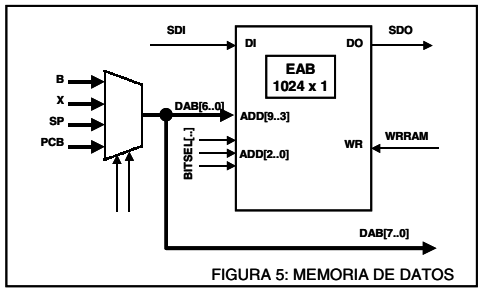

FIGURA 5: MEMORIA DE DATOS

Para la memoria de datos pueden aprovecharse ciertas características de la familia FLEX10K que facilitan su empleo en procesadores de arquitectura serial, usando para esta RAM de 128 bytes un EAB (figura 5) configurado como memoria de 1024 bits (10 líneas de direcciones), donde tres de estás líneas corresponden a BITSEL[..] y las 7 restantes a DAB[6..0]. Para generar las líneas de direccionamiento del bus DAB, un conjunto de 8 multiplexores 4:1 (16 LEs, en pares enlazados en modo cascada) permite elegir la dirección entre los índices B y

X, el stack pointer SP, o una dirección definida en la instrucción (modo directo).

4.4.7. La Unidad Aritmético Lógica (ALU):

Esta unidad realiza operaciones de modificación y comparación entre dos posibles operandos, pudiendo almacenar el resultado de la operación en A, un registro temporario interno, un registro del mapa de memoria de datos, o en ninguna parte. Para analizar la ALU deben dividirse a las posibles instrucciones según el tamaño y tipo de operandos:

Instrucciones a nivel de byte:

- ***aritméticas:*** ADD, ADC, SUBC, INC, DEC, DRSZ, DCOR
- ***lógicas:*** AND, OR, XOR, ANDSZ, CLR
- ***de desplazamiento:*** SWAP, RLC y RRC
- ***de comparación:*** IFEQ, IFNE, IFGT

Instrucciones a nivel de bits:

- manejo de bits: SBIT, RBIT
- comparación al bit: IFBIT

Donde el manejo de las instrucciones a nivel de bit puede homologarse al de instrucciones a nivel de byte que hagan el AND (RBIT, IFBIT) o el OR (SBIT) del dato a procesar con una constante compuesta por un '1' y 7 '0's, o un '0' y 7 '1's.

Instrucciones sobre otros registros:
Existen además instrucciones de comparación que trabajan sobre campos de 4 bits del registro B (IFBNE) o sobre 1 bit del PSW (IFC, IFNC), que, si bien son de tipo lógico, por sus particularidades hacen razonable su procesamiento por separado.

4.4.7.1. Módulos que componen la ALU:

Aunque macroscópicamente puede verse a la ALU como un único bloque funcional, en realidad es posible identificar claramente distintos módulos funcionales que la componen.

4.4.7.1.1. De generación de constantes: al analizar el origen de los operandos es posible detectar la necesidad de ciertos valores constantes, y por eso surge la necesidad de un bloque generador de constantes (CG, figura 6), que en función de las líneas de control K[..] y del valor de BITSEL[..] genera en forma serial las siguientes constantes:

- ***0x9A, 0xA0, 0xFA, 0x00:*** para la instrucción de ajuste decimal DCOR; las distintas constantes corresponden a posibles valores de los flags C/HC (00, 01, 10 o 11 respectivamente) y son seleccionadas a través de $K_2K_1K_0$=000 a 011. La constante 0x00 puede también usarse para el caso de CLRA
- ***0x01*** ($K_2K_1K_0$=100)***:*** para la instrucción INCA (incremento de A).
- ***0xFF*** ($K_2K_1K_0$=101)***:*** para la instrucción DECA.
- ***la máscara para las instrucciones de a bit:*** (caso de K_2K_1=11) que surge de una comparación entre las líneas BITSEL[..] con los 3 bits bajos del código de operación, usando K_0 para definir si es la máscara con sólo un '0' o con sólo un '1'.

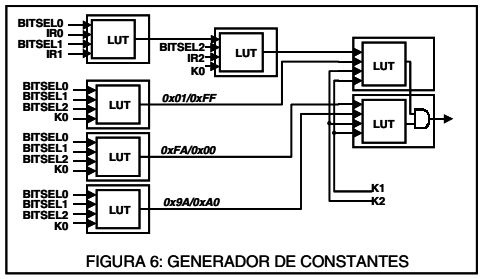

FIGURA 6: GENERADOR DE CONSTANTES

Como se observa en la figura 6, todo este bloque sólo consume 7 LEs, con complejidad equivalente a la de una solución paralela.

4.4.7.1.2. La unidad lógica: realiza operaciones lógicas (AND, OR, XOR) o de comparación (EQ, GT) sobre operandos que provienen del acumulador o la memoria de datos DCB con datos que provienen del DCB, del CG, o la memoria de programa PCBS

- **A,PCBS**: AND; OR; XOR; ANDSZ; IFEQ A,#; IFNE A,#; IFGT A,#;
- **A, DCB:** AND; OR; XOR; IFEQ A,MD; IFNE A,MD; IFGT A,MD;
- **A,CG:** CLR A
- **CG,DCB:** SBIT #,MD; RBIT #,MD; IFBIT #,MD;
- **DCB,PCBS:** IFEQ MD,#

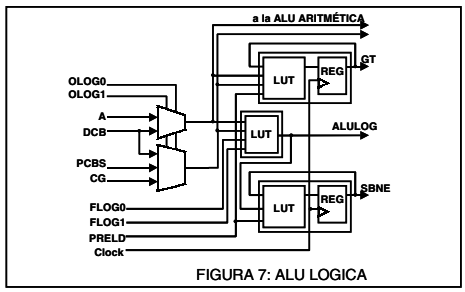

FIGURA 7: ALU LOGICA

Para este módulo (figura 7) sólo se usan 6 LEs, necesarios para configurar un multiplexor de 2 entradas (1 LE), uno de 3 entradas (2 LEs en modo CASCADE), 1 LE que realiza sobre estos dos datos una función lógica entre 3 posibles (AND, OR, XOR), 1 LE operando como flipflop

RS para la detección de igualdad (para IFEQ) o bit seteado (para IFBIT) y 1 LE para la detección de mayor (para la instrucción IFGT). Una solución paralela para este módulo requeriría muchos más recursos de hardware por la multiplicación de multiplexores.

4.4.7.1.3. La unidad aritmética: es sumamente simple, estando a cargo de las instrucciones ADD, ADC, SUBC, DRSZ, INCA, DECA, y DCOR.

- **A,PCBS**: ADD A,#; ADC A,# ; SUBC A,#
- **A,DCB:** ADD A,MD; ADC A,MD; SUBC A,MD;
- **A,CG:** INC A; DEC A; DCOR
- **DCB,CG:** DRSZ

Como se observa, las posibles fuentes de datos son un subconjunto de las necesarias en la unidad lógica, pudiendo usarse el mismo conjunto de multiplexores de selección.

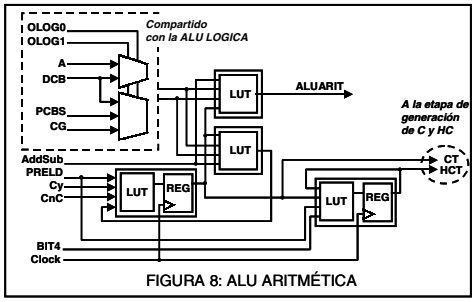

FIGURA 8: ALU ARITMÉTICA

La unidad aritmética (fig.8) consume sólo 4 LEs, y se basa en un sumador tipo CarrySave [9] donde en la realimentación del acarreo se ha incorporado un LE adicional para considerar o no el valor del flag C durante la suma del bit menos significativo (necesario en operaciones de suma o resta con carry, ADC y SUBC). Otra particularidad es la línea AddSub, que niega al minuendo en la operacion SUBC. A esta unidad se le ha adicionado un LE para capturar el valor del acarreo entre los bits 3 y 4, que será luego usado para actualizar el flag HC. Una solución paralela hubiera necesitado muchos más LEs, por la necesidad de sintetizar un sumador/restador de 8 bits y algun multiplexor.

4.4.7.1.4. La unidad de shift: otro modo de procesamiento es el definido por las instrucciones de shift RLC, RRC y SWAP. En este caso, si se llama A[7..0] y Cy al valor previo del Acumulador y el flag de Carry, deben generarse en forma serial los valores:

- RLC : $A_6A_5A_4A_3A_2A_1A_0Cy$
- RRC: $CyA_7A_6A_5A_4A_3A_2A_1$
- SWAP: $A_3A_2A_1A_0A_7A_6A_5A_4$

en función del código de operación y el valor de BITSEL[..]. La unidad de shift completa (figura 9) consume 19 LEs, con 2 LEs para generar las señales que serán usadas para actualizar los flags de C y HC.
Una solución paralela requeriría algo más de

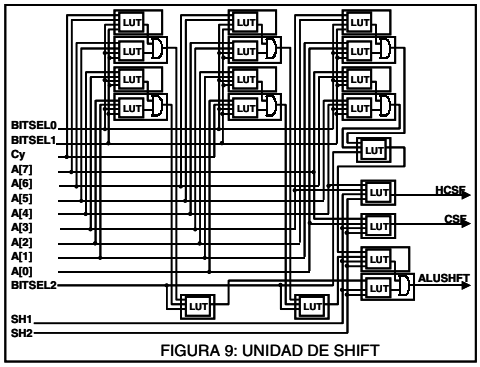

FIGURA 9: UNIDAD DE SHIFT

recursos (22 LEs).

4.4.7.1.5. Esquema global: Definidos los bloques que la forman, el esquema de la ALU se observa en la figura 10, donde el multiplexor 3:1 indicado en la salida decide cuál de los resultados (lógico, aritmético o de shift) será empleado.

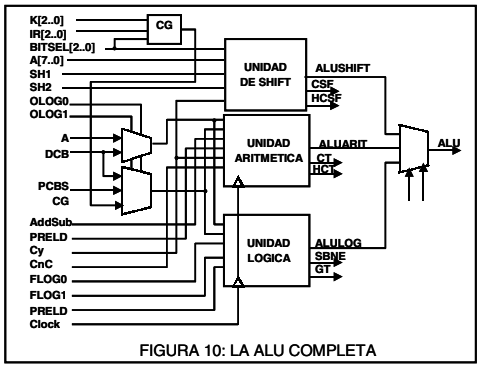

FIGURA 10: LA ALU COMPLETA

4.4.8. Sobre el registro PSW: los registros de status/control son un caso especial de registros internos, dado que son modificados no sólo en forma explícita por instrucciones de escritura sino también como consecuencia indirecta de otro tipo de instrucciones.
Uno de estos casos es el del registro PSW, que puede ser leído y escrito por programa, pero donde los bits HC (Half-Carry), C(Carry), y GIE (Global Interrupt Enable) también son afectados por las instrucciones de shift RLC y RRC, por las aritméticas ADC y SUBC, por las especiales SC y RC, y por los procesos de interrupción.

En la figura 11 se muestran las modificaciones requeridas para los bits HC y C, donde entre la etapas *shift* y *store* se incorpora un multiplexor con 5 señales de control, 4 posibles fuentes de datos reales (su propio feedback, el *shift*

register, o las señales provenientes de las etapas aritméticas y de shift) y dos datos "virtuales" (0 y 1, necesarios para las instrucciones RC y SC).

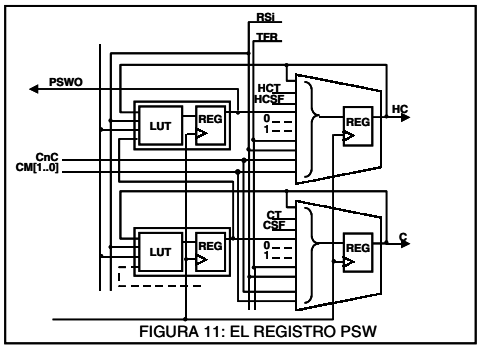

FIGURA 11: EL REGISTRO PSW

4.4.9. La memoria de programa y otras unidades:

Un procesador requiere muchos otros módulos funcionales además de los descriptos, tales como la memoria de programa, y los bloques de generación de direcciones de programa, de decodificación de instrucciones, de manejo de ramificaciones, de manejo de interrupciones y de generación de señales de control (fig.12).

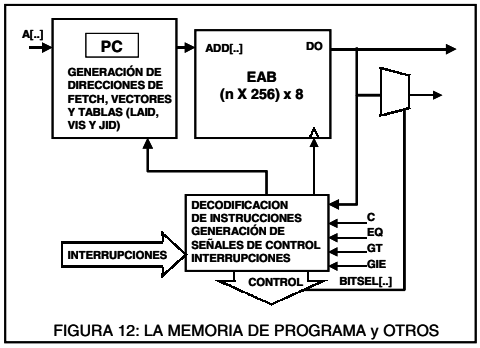

FIGURA 12: LA MEMORIA DE PROGRAMA y OTROS

En este caso para la operación de estos bloques no interesa si el movimiento de datos es serial o paralelo sino la funcionalidad, por lo que procesadores seriales o paralelos requieren señales de control similares y circuitos de complejidad equivalente.

5. Análisis de los resultados obtenidos

Para comparar la alternativa serial frente a la paralela deben evaluarse procesadores de complejidad equivalente; en este caso se ha tomado como ejemplo de solución paralela el MC6805 diseñado previamente [8], que es una solución altamente eficiente, con un poder de cómputo ligeramente inferior al de un COP8SAx. Para la comparación se han tenido en cuenta las características propias de las FLEX10K (modos de operación aritmético y contador, y uso de CARRY y CASCADE), computando un uso de recursos de 0,5 LE a 0,65 LE por entrada de multiplexor.

5.1. Puerta de entrada: significa un gasto de recursos equivalente a la de un procesador paralelo, aunque con una reducción de casi 4 a 1 en complejidad de cableado, porque si bien el bus de datos disminuye de 8 a 1 bit, se mantienen las líneas de decodificación y aparecen las líneas BITSEL[..].

5.2. Puerta de salida: similar al caso previo; aunque el registro serial requiere más LEs por su estructura *Shift/Store*, en el caso paralelo para realizar la lectura se necesitan cerca de 5 LEs más para el ruteado mediante multiplexores hacia la CPU.

5.3. Registros B, X y SP: la solución serial presenta ventajas de cableado y mejor uso de recursos. Y esta ventaja no es aún mayor sólo porque la familia FLEX10K dispone del modo de operación "up/down counter" en los LEs, que en una solución paralela simplifica las acciones de autoincremento o autodecremento.

5.4. Otros registros especiales: el caso del PSW es similar a los previos, dado que el incremento de complejidad requerido para considerar los comportamientos especiales de ciertos flags es simlar en cualquier solución.

5.5. Acumulador: la existencia de un gran abanico de entrada genera una penalidad importante para los procesadores paralelos, que deben poseer tantos multiplexores como bits tenga el bus de datos; la solución serie sólo requiere algo más que 16 LEs, frente a los cerca de 35 LEs de un 6805.

5.6. ALU: se obtiene una reducción importante de complejidad gracias al procesamiento en serie; la ALU sólo necesita 36 LEs, frente a los cerca de 100 LEs requeridos por un 6805.

5.7. La memoria de programa y otras unidades: la simple comparación en cuanto a poder de cómputo, tamaño de palabra de datos y conjunto de instrucciones entre un COP8SAx y un MC6805 hace presumir que ambos casos presentan una complejidad equivalente; en realidad, dada la arquitectura Harvard de los COP8SAx, su unidad de generación de direcciones de programa será más simple que en el 6805.

Considerando todos los bloques circuitales, la reducción en el uso de LEs en relación al caso paralelo puede estimarse en alrededor de un 15%~20%, con una reducción de exigencias de cableado en más del 50%.

Aunque la solución serial se hace a costa de una reducción en la velocidad de operación, debe considerarse que dados los múltiples ciclos de

reloj disponibles en cada ciclo de instrucción y los flipflops propios a cada LE, en este caso se hace posible la inclusión de un cierto grado de pipeline con complejidad adicional casi nula, que podría resultar en mejoras importantes en la performance final.

6. Conclusiones

Se ha presentado cómo una solución híbrida, que combine procesamiento serial y paralelo, es una alternativa válida para el desarrollo de microprocesadores empotrados.

Si bien la serialización del movimiento de datos tiene como desventaja la menor performance de velocidad, esta desventaja suele ser compensada por mecanismos de ruteado y procesamiento de datos mucho más simples, cuyos beneficios se hacen más evidentes cuanto mayor es el ancho de palabra. Si se tiene en cuenta que los recursos de cableado disponibles dentro de un CPLD suelen tener restricciones (por ejemplo, a un LAB sólo pueden ingresar cerca de 30 señales generadas en otros LABs) las soluciones seriales resultan aún más convenientes.

Lo que debe enfatizarse es que, sea cual sea la solución empleada, el proceso de concepción debe tener en cuenta las ventajas y limitaciones de cada tecnología, de modo de facilitar el proceso de síntesis; por eso en este artículo los esquemas propuestos para los distintos módulos se han elaborado teniendo en cuenta cómo explotar al máximo el ***fan-in*** y los modos de operación los LEs, que son los elementos atómicos disponibles en la arquitectura de las FLEX10K. Este tipo de concepción es abolutamente independiente del método de ingreso del diseño, y previo a la selección de un lenguaje de descripción de hardware, o decisión equivalente.

7. Agradecimientos:

Se agradecen las valiosas críticas y sugerencias del Ing.Horacio Villagarcía Wanza.

8. Bibliografía:

[1]. Altera. “Intellectual Property Catalog”, M-CAT-AIPS-01, Altera Corp., 1999.

[2]. Altera. “ARM-based Embedded Processor Device Overview”. Datasheet A-DS-EXCARM-01, Altera Corp., 2000.

[3]. Altera. “MIPS-based Embedded Processor Device Overview”. Datasheet A-DS-EXCMIPS-01, Altera Corp., 2000.

[4]. Altera. “EXCALIBUR Development Kit with the NIOS Embedded Processor”. Datasheet A-DS-EXCKIT-01, Altera Corp., 2000.

[5]. Altera. “1999 Device Data Book”, A-DB-0599-01, Altera Corp., 1999, USA.

[6]. Andraka, R. “Building a High Performance Bit Serial Processor in a FPGA”. Proceedings of Design SuperCon’96, Juanuary 1996, pp.5.1-5.21

[7]. Hennessy, J. and Patterson, D.: “Arquitectura de Computadores. Un enfoque cualitativo”. Mc.Graw Hill, USA, 1993. ISBN 84-7615-912-9.

[8]. Jaquenod, G. "Diseño de un microcontrolador MC6805 usando lógica programable FLEX de ALTERA". VI Workshop IBERCHIP, Sao Paulo, Brasil, Marzo 2000.

[9]. Jaquenod,G. y De La vega, R. y De Giusti, M.. “Aplicaciones aritméticas usando lógica programable”. TELEC’2000, Santiago de Cuba, Cuba, Julio 2000.

[10]. Maniwa, Tets y otros. “The Embedded Microprocessor report”. Integrated System Design magazine editorial, November 2000.

[11]. National. “COP8SAA7/ COP8SAB7/ COP8SAC7 8-bit OTP Microcontroller”. Data Sheet, Lit # 102130-002, National Semiconductor Corp., USA, 1997.

[12]. National. “COP8 feature family user’s manual”. Lit # 620897-003, National Semiconductor Corp., USA, 1996.

[13]. National. “COP8 Family, Technical Documentation, Tools & Software Version 3.0, 8/99”. Lit # 530094-003, National Semiconductor Corp., USA, 1999.

[14]. Pollard, L.H. : “Computer Design and Architecture”. Prentice Hall, USA, 1990. ISBN 0-13-167255-X.