Diseño Automático de sistemas

Diseño digital de algoritmos

UNIVERSIDAD NEBRIJA

Objetivos Clase

- Aprender a implementar un algoritmo en tiempo real.
- Entender las métricas de una implementación.
- Entender como representar un algoritmo.
- Entender las diferentes estrategias a la hora de realizar una implementación digital.
- Entender conceptos: *Pipeline*, paralelizar, *retiming, unfolding, folding*

Referencias

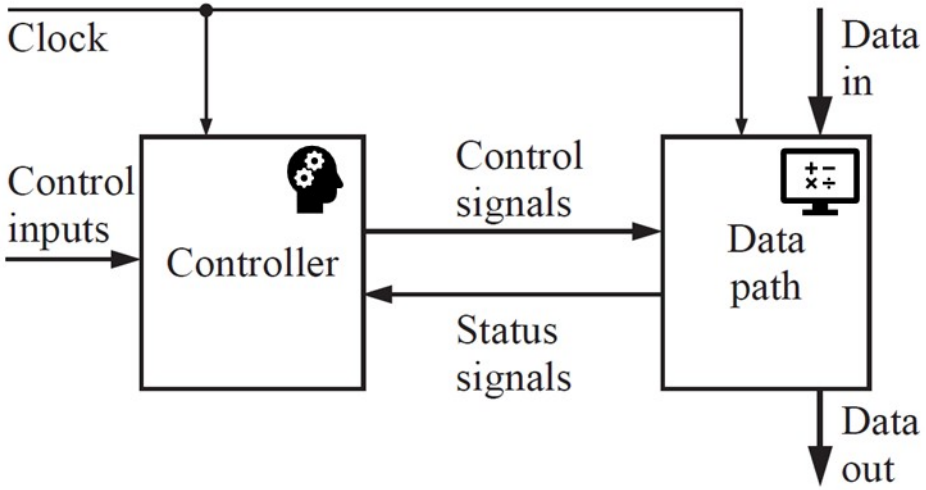
- **VLSI Digital Signal Processing Systems**: Design and Implementation, Keshab K. Parhi, (1999).

Contenidos

1. Algoritmos en tiempo real.
2. Ejemplo: Media móvil.
3. Representación
4. Camino crítico
5. Pipeline
6. Retiming
7. Loop Folding
8. Loop Unfolding

Repaso: Controller + Datapath

Recordando la estructura de controller + datapath.



Datapath: Procesado de datos

Hasta ahora todo lo que hemos visto ha sido de la parte del controlador.

En esta clase vamos a ver diferentes algoritmos y como se pueden representar, optimizar e implementar.

Filtros digitales(I)

Son sistemas que procesan las señales discretizadas, es decir, digitales.

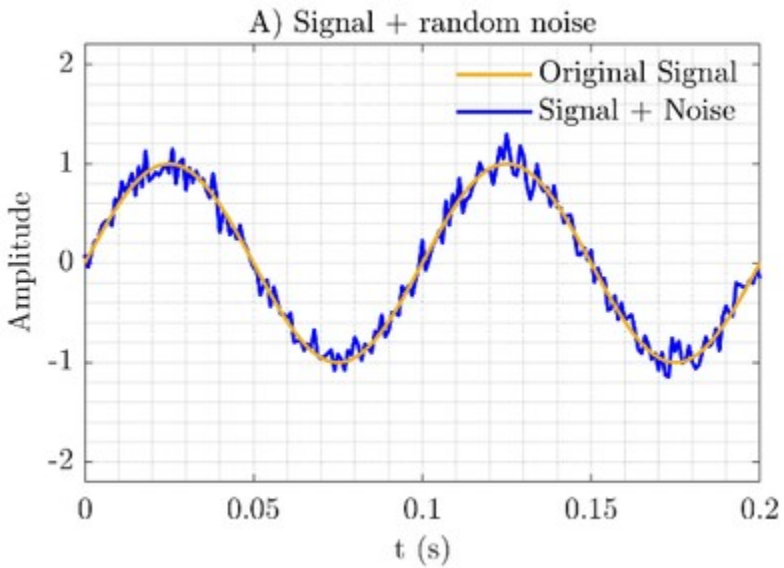
Tienen muchas aplicaciones desde: Audio, telecomunicaciones, procesado de texto, clasificación, detección de patrones, compresión/descompresión, análisis de datos, acondicionamiento de sensores, etc…

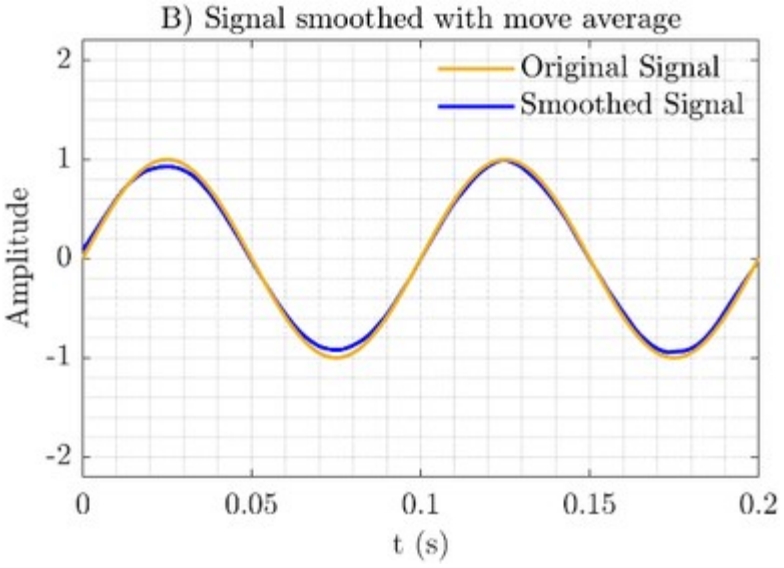
Filtros digitales(II)

Al final casi cualquier procesado que se realice sobre datos digitales esta basado en filtros.

Vamos a ver el ejemplo de quitar ruido de una señal.

Media Movil: Problema señal con ruido





Media móvil

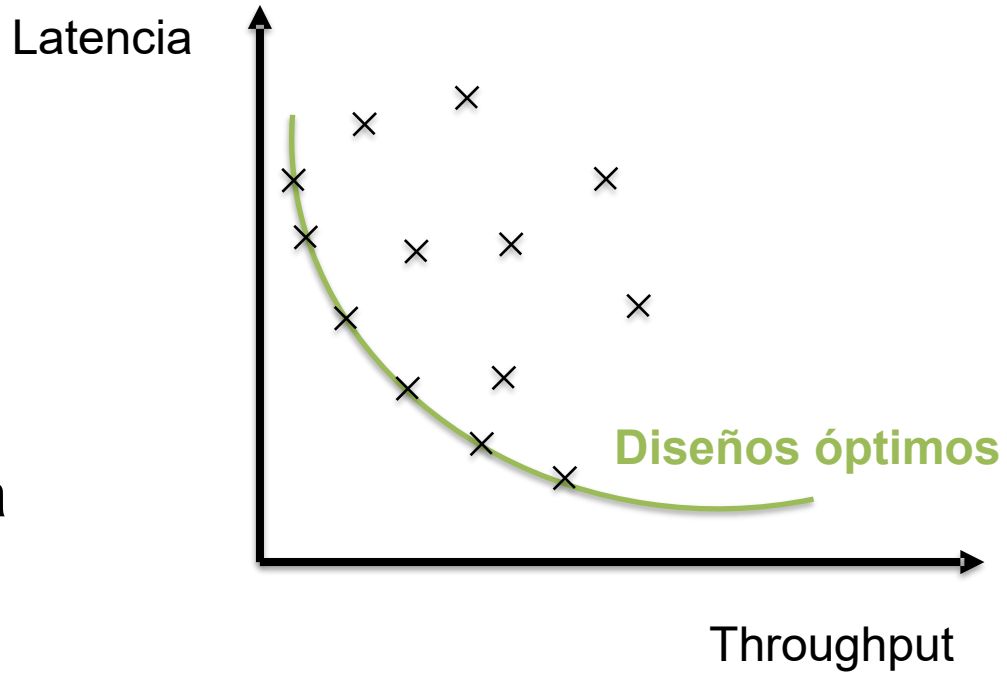
$$SMA_k = \frac{p_{n-k+1} + p_{n-k+2} \cdots + p_n}{k}$$

Métricas de una implementación

- **Latencia:** Tiempo en salir la primera muestra procesada.
- **Troughput:** Datos por ciclo.
- **Área:** Número de operadores utilizado
- **Frecuencia:** Ciclos por segundo.

Objetivo

- Reducir latencia.
- Aumentar troughput.
- Reducir área.
- Aumentar frecuencia



Composición de un algoritmo

- Operaciones
- Datos

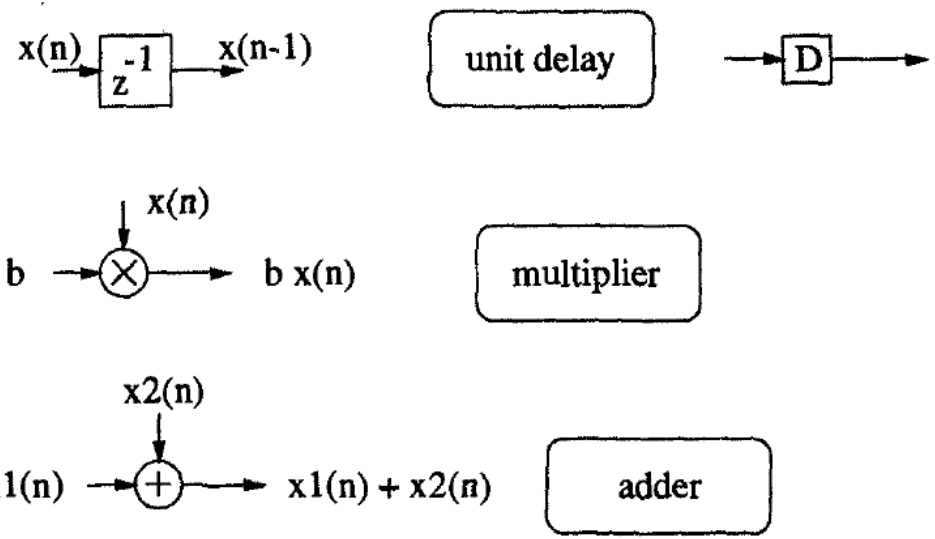
} Relaciones

Cómo representar un proceso (I)

Deseamos representar las operaciones y las relaciones entre ellas, a diferencia de lo habitual en SW es necesario tener en cuenta los retrasos y los tiempos que llevan realizar las operaciones.

Cómo representar un proceso (II)

Diagrama de bloques:



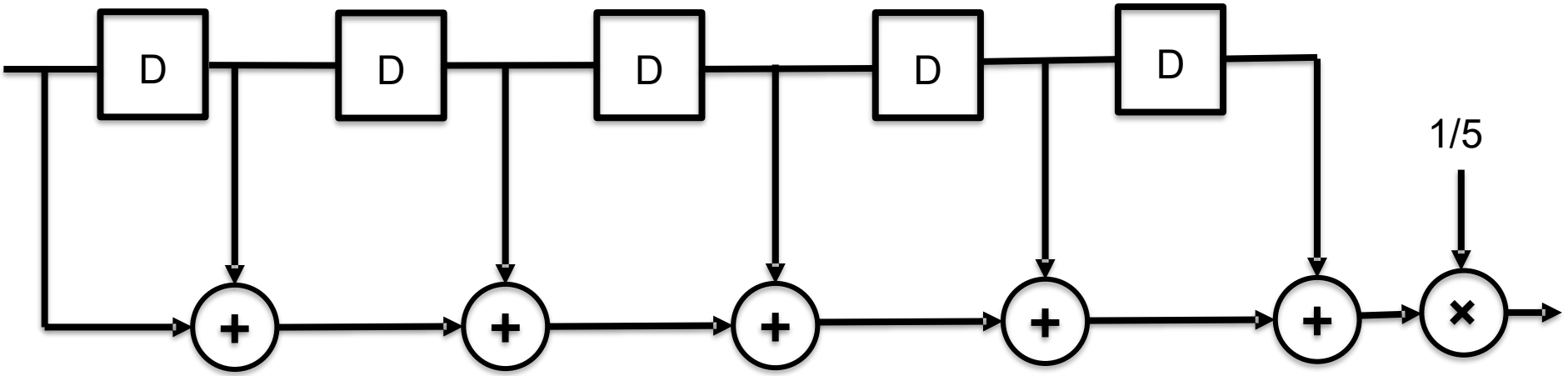
Media móvil: código en C

$$SMA_k = \frac{p_{n-k+1} + p_{n-k+2} \cdots + p_n}{k}$$

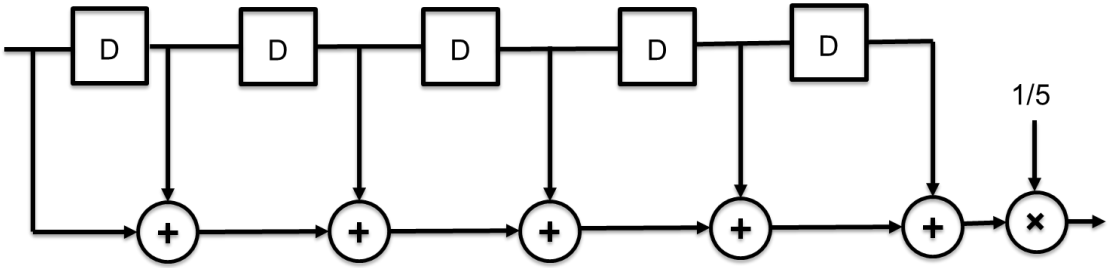
```
1
2  float sample = 0;
3  float sample_1 = 0;
4  float sample_2 = 0;
5  float sample_3 = 0;
6  float sample_4 = 0;
7  while 1 {
8      sample_4 = sample_3;
9      sample_3 = sample_2;
10     sample_2 = sample_1;
11     sample_1 = sample;
12     sample = new_sample;
13
14     acc = sample + sample_1 + sample_2 + sample_3 + sample_4;
15     mediaMovil = acc/5;
16 }
```

Cómo representar un proceso (II)

$$SMA_k = \frac{p_{n-k+1} + p_{n-k+2} \cdots + p_n}{k}$$



Cómo representar un proceso (II)



Latencia: 5 ciclos

Throughput: 1 dato/ciclo

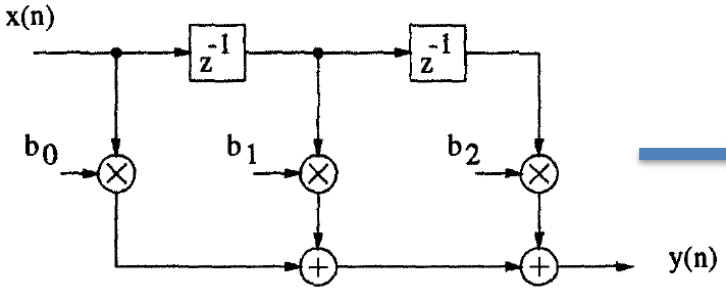
Área:

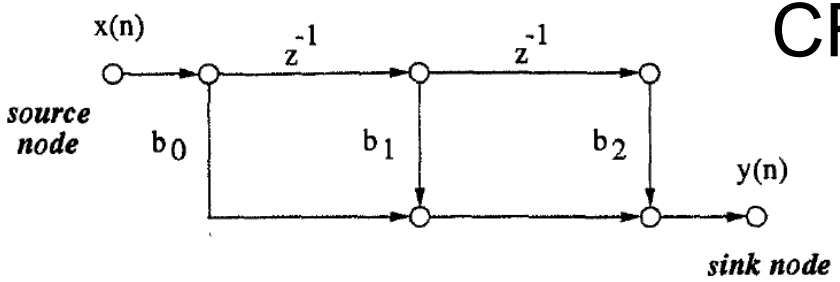
- 5 sumadores
- 1 multiplicador
- 5 registros

Cómo representar un proceso (II)

- CFG: Control Flow Graph, sirve para representar el flujo de operaciones. Muy parecido a los diagramas de estado.
- DFG: Data Flow Graph, representan el flujo de los datos, más útil en procesado digital.

Cómo representar un proceso (II): CFG





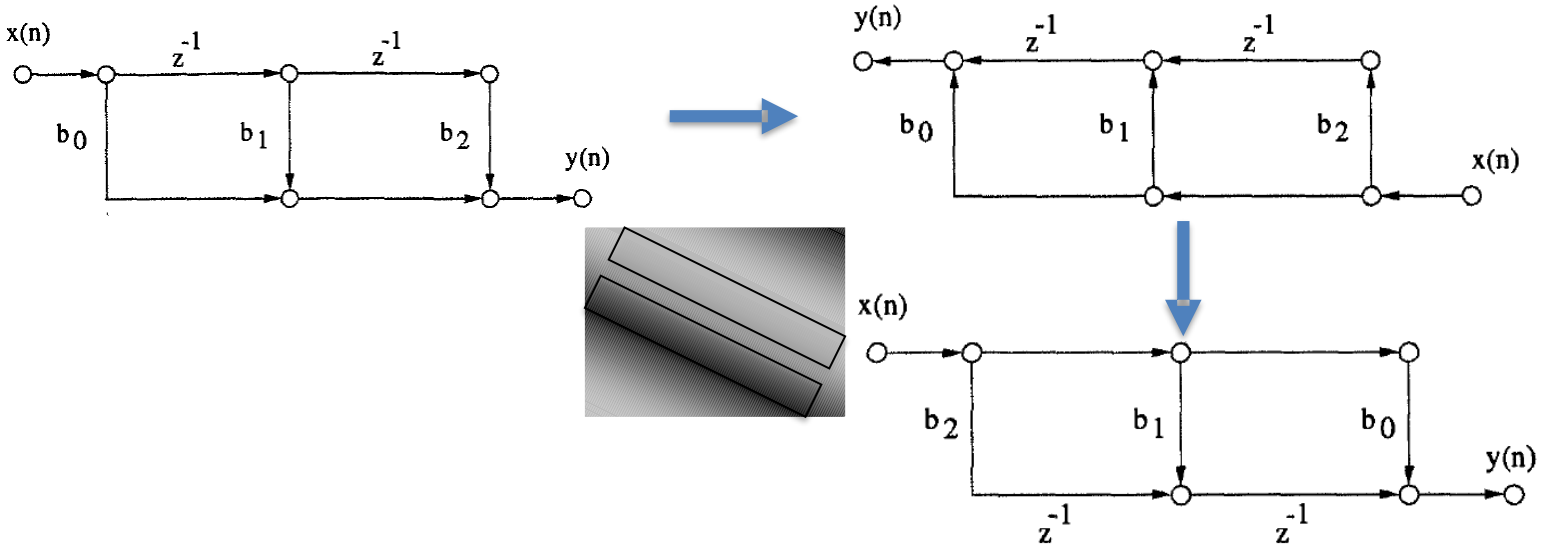
CFG

$$y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2)$$

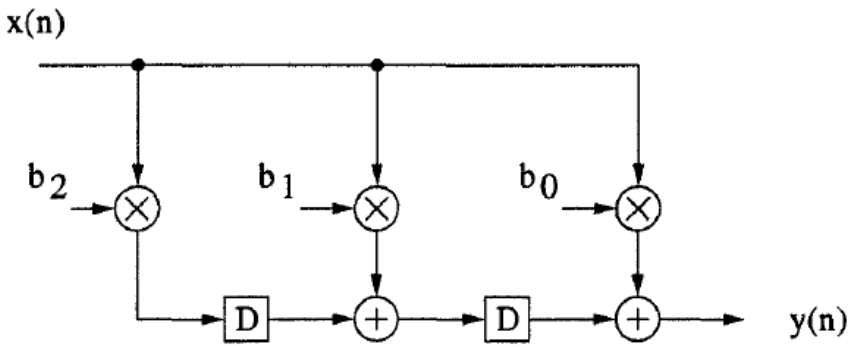
- Cada arco/arista (edge) es una transformación lineal (multiplicacion)
- Una suma es dos arcos entran al mismo nodo.
- Hay dos nodos especiales source/sink

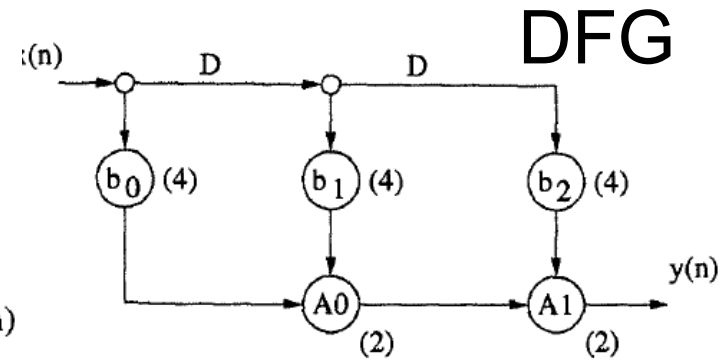
CFG: Invertir grafos

- Los grafos se pueden invertir/trasponer de la siguiente manera: El primer y ultimo grafo son equivalentes.



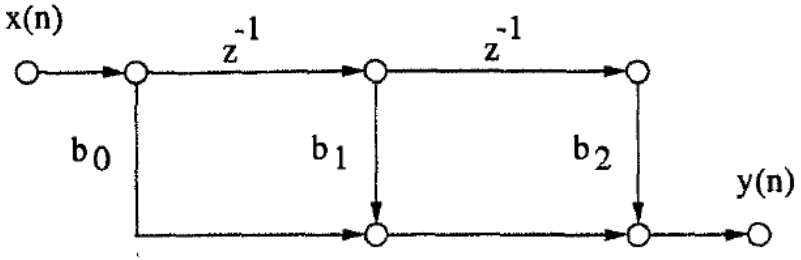
Cómo representar un proceso (III): DFG

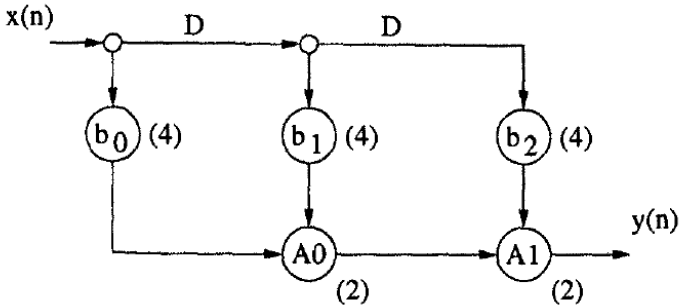




- Cada nodo es una operación
- Cada arista es una conexión de datos que puede incluir delay.
- Encima de cada nodo se escribe el tiempo de procesado.

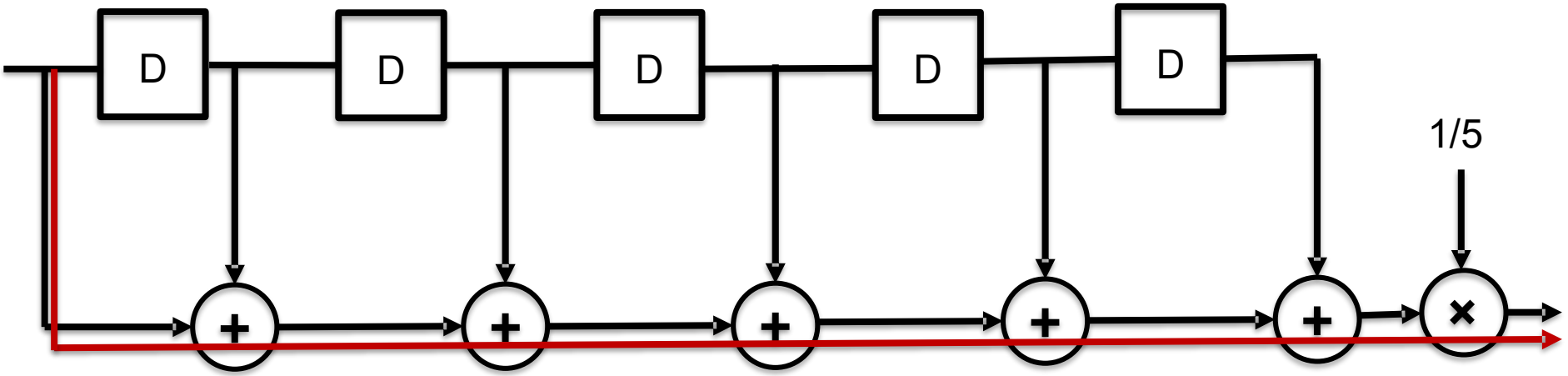
CFG vs DFG





Son equivalentes

Camino crítico



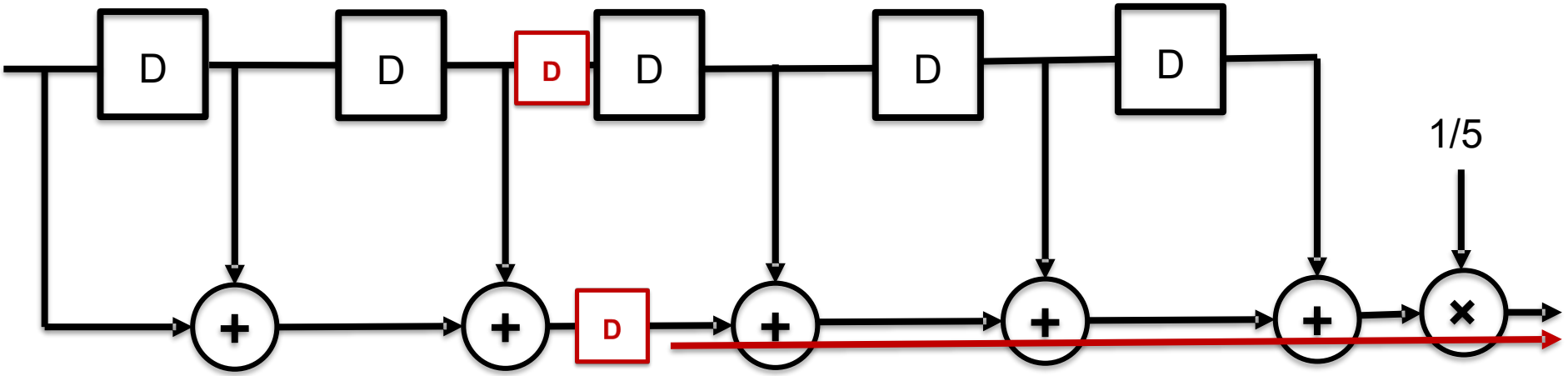
Iteration bound

Existe un limite teórico para la ejecución de algoritmo, se puede calcular usando los DFGs. (En este curso no vamos a trabajar en ello).

Pipeline

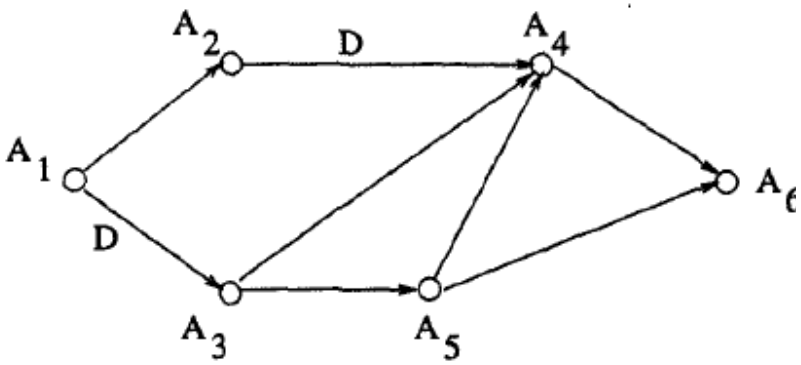
Consiste en introducir registros para aumentar la frecuencia, aumentando la latencia y levemente el área.

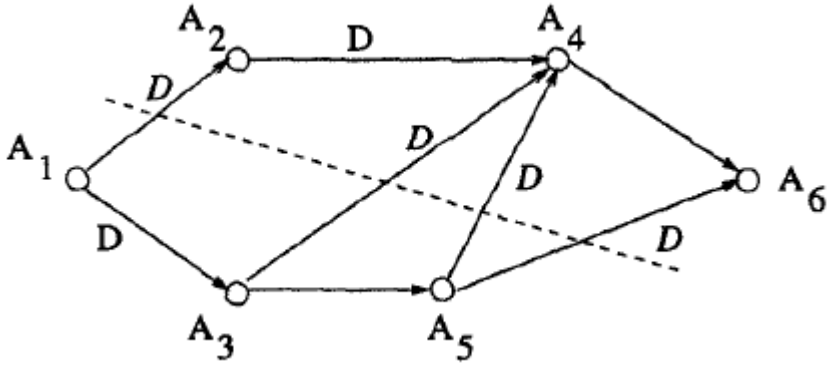
Camino crítico: Pipeline



Pipeline

Para poder hacer pipeline es necesario cortar todas las aristas en la misma dirección, sin dejar ninguna fuera.



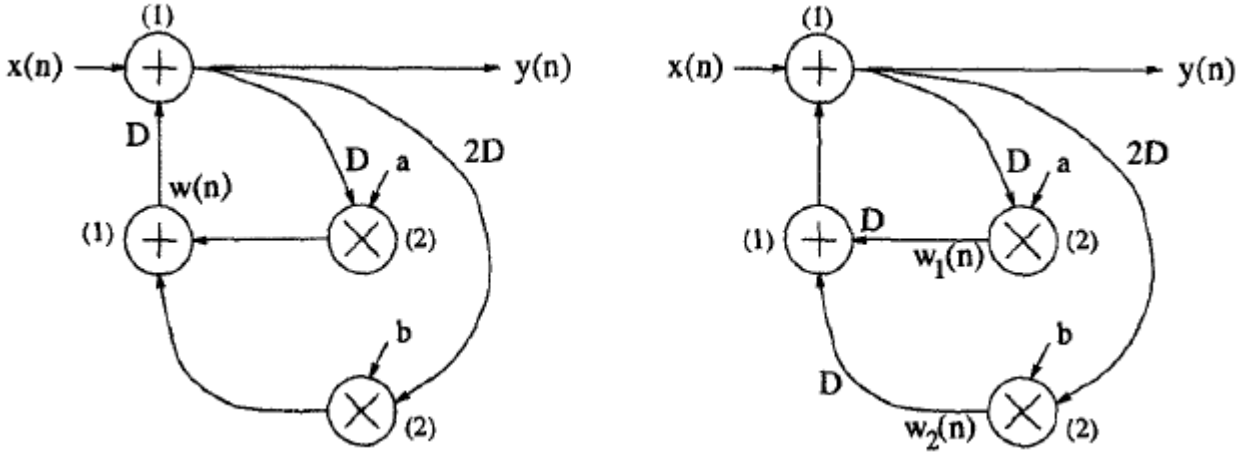


Retiming (temporización)

Consiste en transformar el algoritmo para cambiar la posición de los delays sin afectar a las características de entrada y salida del algoritmo.

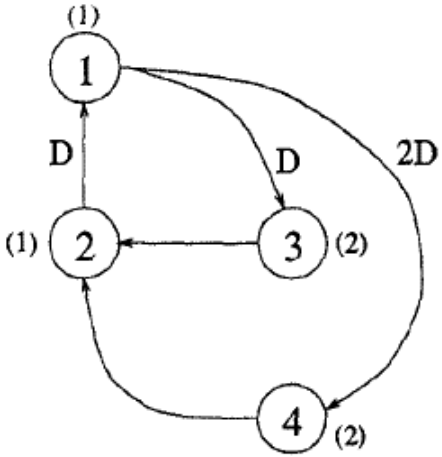
Retiming (temporización)

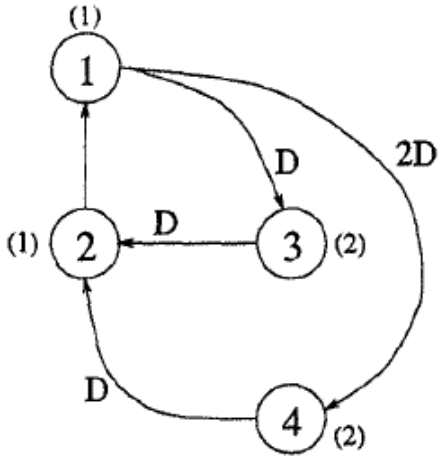
Los delays en una arista se pueden transferir a las aristas anteriores, ej:



Retiming (temporización)

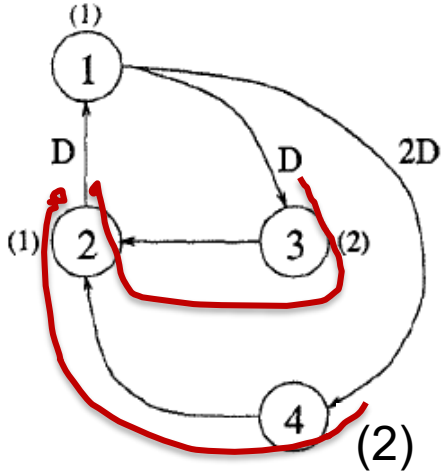
Los delays en una arista se pueden transferir a las aristas siguientes, ej:



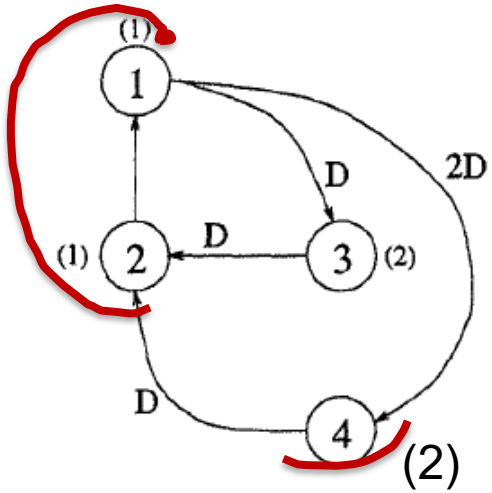


Retiming (temporización)

Usando el retiming se puede reducir el camino crítico, ej:



Camino crítico: 3



Camino crítico: 2

Paralelizar: Loop Unfolding

Consiste en desdoblar los bucles, para paralelizar las operaciones. Incrementamos el throughput ocupando más área.

$$y(n) = ay(n-9) + x(n)$$

$$\begin{aligned} y(2k) &= ay(2k-9) + x(2k) \\ y(2k+1) &= ay(2k-8) + x(2k+1) \end{aligned}$$

Paralelizar: Loop Unfolding

Consiste en desdoblar los bucles, para paralelizar las operaciones.

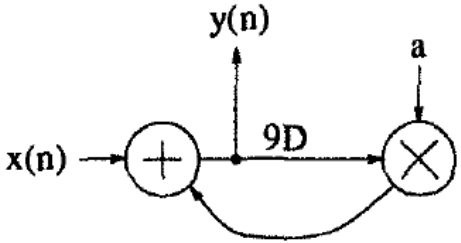
```
1   int x;
2   for (x = 0; x < 100; x++)
3   {
4       delete(x);
5   }
```

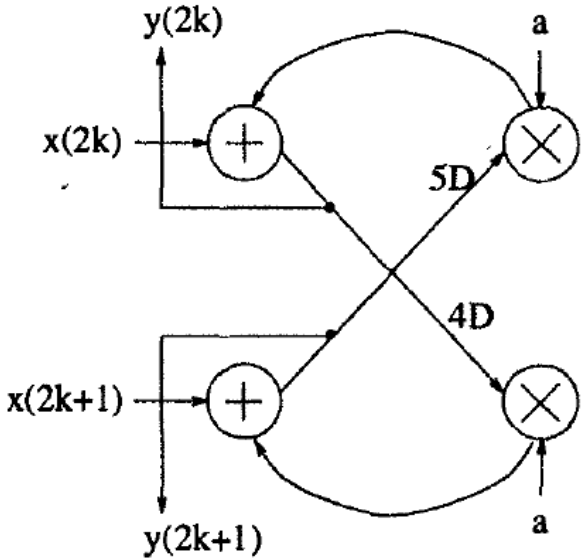
```
7   int x;
8   for (x = 0; x < 100; x += 5 )
9   {
10      delete(x);
11      delete(x + 1);
12      delete(x + 2);
13      delete(x + 3);
14      delete(x + 4);
15  }
```

Paralelizar: Loop Unfolding

$$y(n) = ay(n-9) + x(n)$$

$$\begin{aligned} y(2k) &= ay(2k-9) + x(2k) \\ y(2k+1) &= ay(2k-8) + x(2k+1) \end{aligned}$$





Resumen Clase anterior

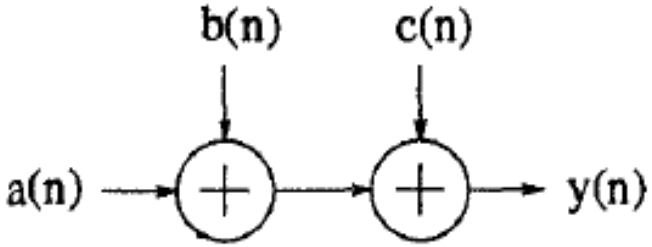
- Características de una implementación: frecuencia, área, throughput y latencia.
- Hemos explicado como usar diferentes grafos cambiar entre ellos y convertirlos a ecuaciones en diferencias.
- Calculo del camino crítico.
- Camino critico inversamente proporcional a frecuencia de funcionamiento.
- Pipeline como forma de reducir el camino crítico.
- Loop unfolding como forma de aumentar el troughput
- Retiming como forma de optimización sin coste

Optimizar en área: Loop Folding

Consiste en reutilizar operadores, logramos reducir el área y como consecuencia perdemos troughput.

Optimizar en área: Loop Folding

Ejemplo: y(n) = a(n) + b(n) +c(n)

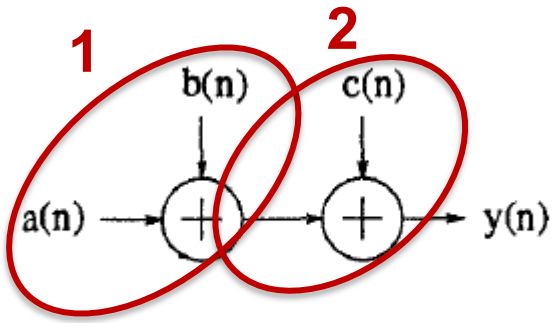


Loop Folding: Proceso

1. Observar dependencias entre operaciones
2. Decidir que operaciones se hacen en cada ciclo
3. Registros necesarios para almacenar resultados temporales.
4. Generación de señales de control.

Loop Folding: Ejemplo

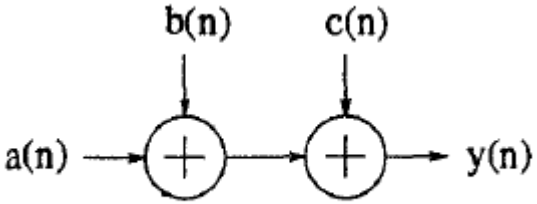
Ejemplo: y(n) = a(n) + b(n) +c(n)



1. r_sum <= a + b;
2. r_sum <= c + r_sum;

Loop Folding: Ejemplo

Ejemplo: y(n) = a(n) + b(n) +c(n)



Cycle	Adder Input (left)	Adder Input (top)	System Output
0	$a(0)$	$b(0)$	–
1	$a(0) + b(0)$	$c(0)$	–
2	$a(1)$	$b(1)$	$a(0) + b(0) + c(0)$
3	$a(1) + b(1)$	$c(1)$	–
4	$a(2)$	$b(2)$	$a(1) + b(1) + c(1)$
5	$a(2) + b(2)$	$c(2)$	–

Loop Folding: Problemas

Al implementar loop folding nos encontramos con que nuestro algoritmo pasa a tener estados dichos estados obligan a que sea necesario un handshake para la entrada y salida de datos.

Cuantificacion

- Consiste en pasar los algoritmos de variables continuas a variables con tamaño y ancho.
- Habitualmente el paso de punto flotante a fijo.
- Muchas veces es inevitable introducir un error al cuantizar.
- Conocer los limites y el error máximo y promedio según el valor de las entradas.

Cuantificacion: Operadores en punto fijo

Las multiplicaciones necesitan tantos bits como tengan sea la suma de los bits de los operandos.

Las sumas necesitan un bit más que el sumando con el tamaño mayor.