Pipeline

Diseño de Sistemas con FPGA

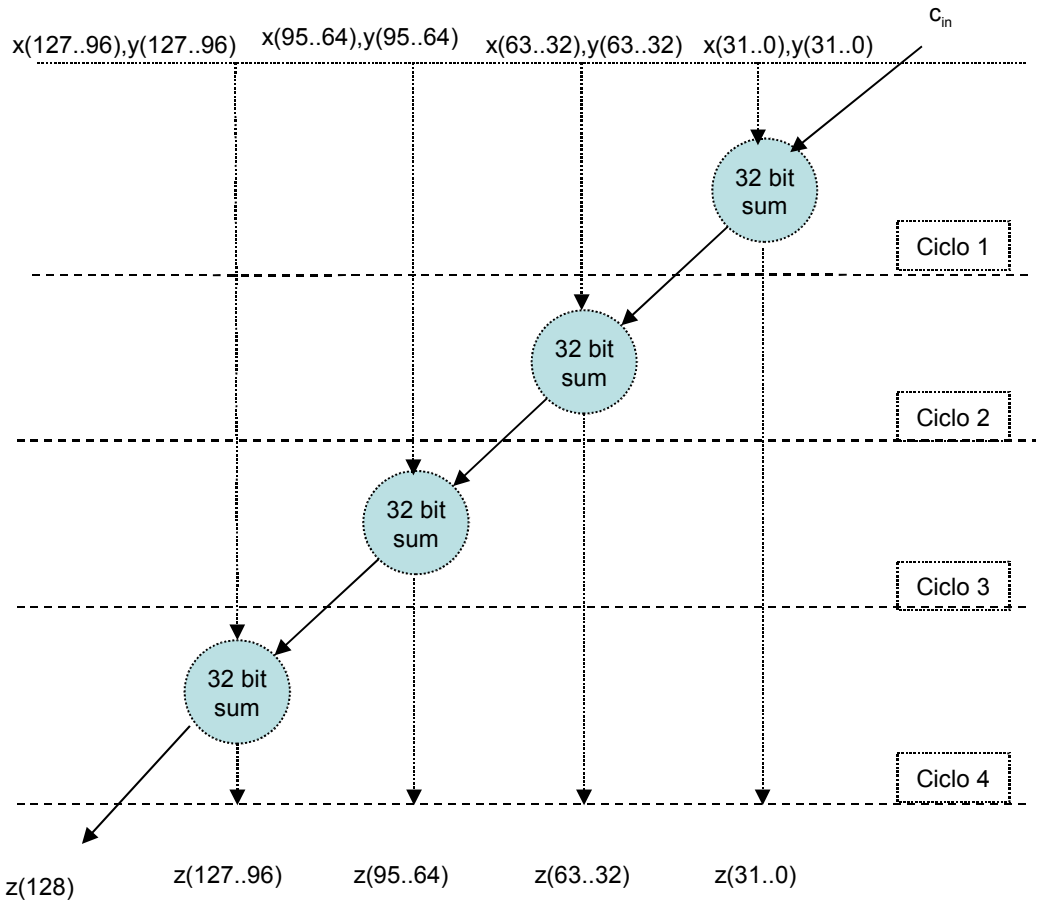
2° cuatrimestre 2008

Patricia Borensztejn

Volvemos a los sumadores

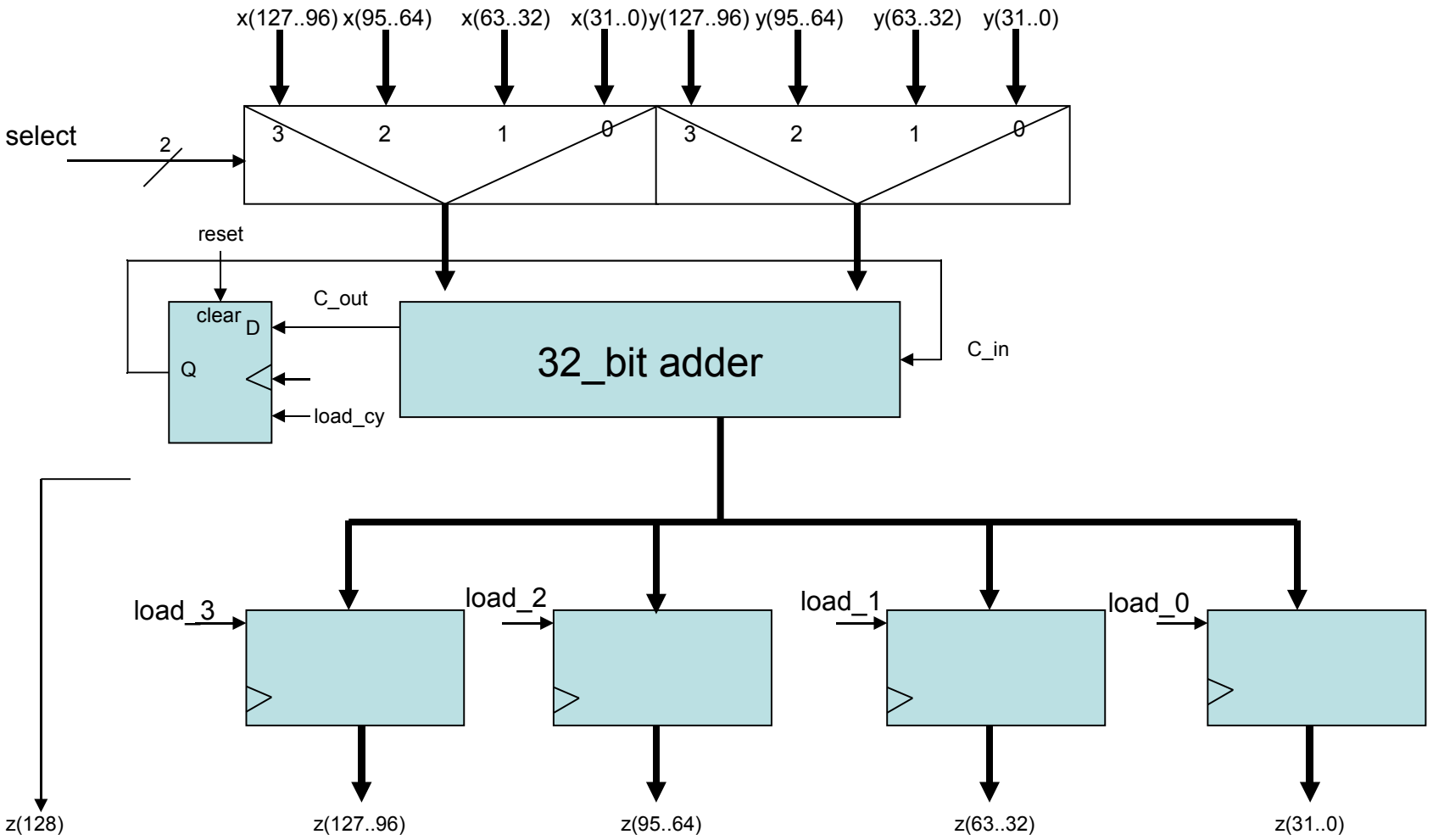
- Queremos implementar un sumador de números grandes, de 128 bits. (n=128) con un sumador de 32 bits (s=32).
- Esta es una implementación secuencial. No paralela. Para que sea paralela, debiéramos tener 4 sumadores de 32 bits.
- Para determinar el número de ciclos necesarios podemos hacer un grafo de flujo de datos (data flow graph) (es obvio en este ejemplo, pero no en otros).

Grafo de Precedencia



- El grafo de precedencia nos indica también los recursos que necesitamos: cada arco que cruza una línea de puntos corresponde a una variable computada en el ciclo anterior que será usada en alguno siguiente➔ o sea que se necesita memoria para guardar su valor.
- En cada ciclo se hará una operación de suma de 32 bits.
- Se necesitan 4 registros para almacenar los resultados parciales y un flip flop para el carry

128 Bit Adder. Implementación secuencial



Costo(C) y Retardo(T) del camino de Datos

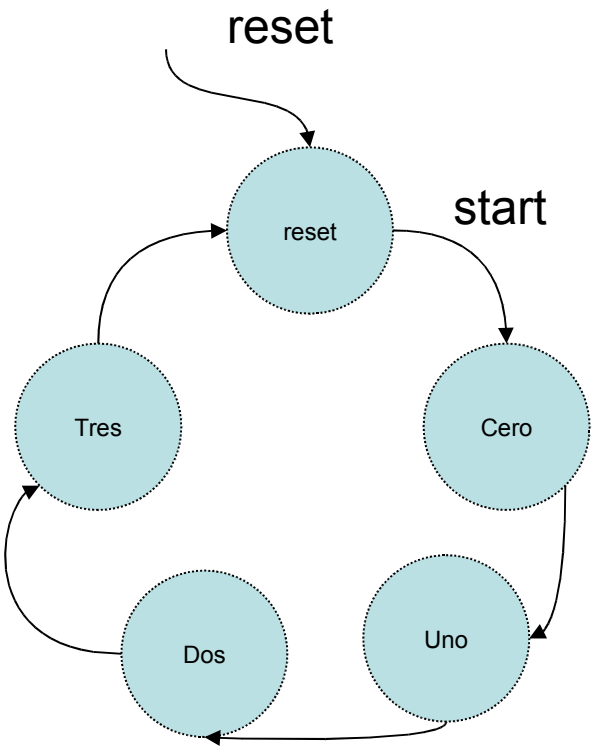
- Costo:

  $C = 192.C_{mux2\text{-}1} + C_{adder(32)} + 129\, C_{FF}$

- Retardo

  $T = 4.T_{clk}$ donde $T_{clk} > 2.T_{mux2\text{-}1} + T_{adder(32)} + T_{FF}$

Grafo de Estados para el Control



- Estado “reset”
  - clear carry
  - done=1
- Estados cero,uno, dos, tres
  - done=0
  - load_cy=1
  - select= estado
  - load(estado)=1

Aclaración: es importante definir antes el comportamiento del automata frente a las señales externas start y reset. Aquí se supone un comportamiento. Puede ser otro

Costo(C) y Retardo(T) del Control

- Costo:

  Hay que agregar el costo de las variables de estado y el costo combinacional.
  Supongamos que el costo combinacional es pequeño comparado con los registros de estado.

  $C(m=4)$ aprox.$= \log_2(m)\, C_{FF}$

- Retardo:

  No agrega a la parte secuencial

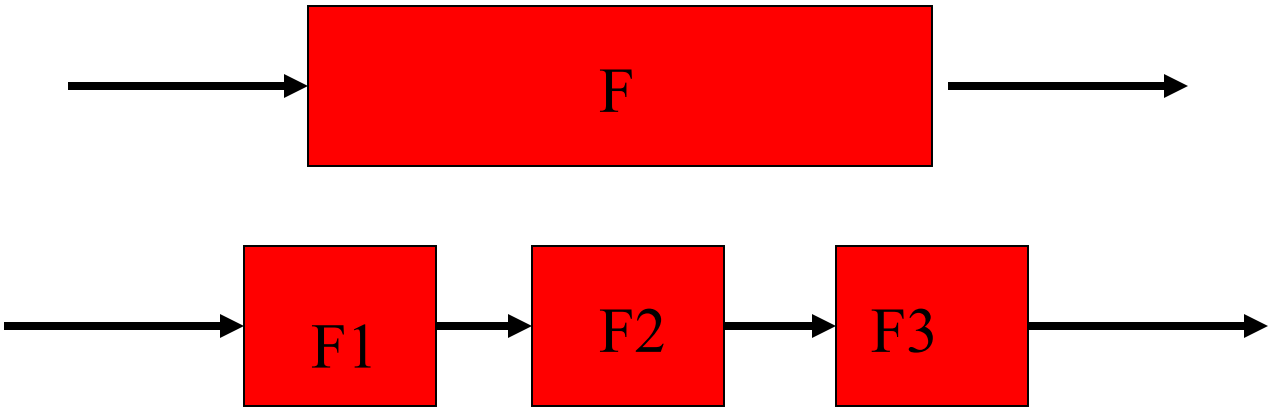
Pipeline

- Es una técnica que permite incrementar el rendimiento de un sistema.
- Es ortogonal a la técnica de paralelismo.

Segmentación

Descompone una determinada operación en *n* suboperaciones a realizar en etapas distintas, de manera que se puedan realizar n operaciones simultáneas, cada una en una etapa distinta.

- Divide una operación en suboperaciones
- Desacopla las suboperaciones



Segmentación

- No reduce la latencia de una instrucción, sino que ayuda a incrementar la productividad de toda la tarea.
- Permite que los recursos se utilizen óptimamente, sin ciclos ociosos.
- La velocidad, o frecuencia con que una instrucción sale del pipeline (cauce) está limitada por el tiempo de proceso de la etapa más lenta.
- Idealmente, la mejora en velocidad debida a la segmentación es igual al número de etapas:
  - la segmentación involucra gastos
  - las etapas pueden no estar equilibradas==> tiempo de inactividad
- El diseñador debe equilibrar la duración de las etapas.

Ejemplo de la lavandería

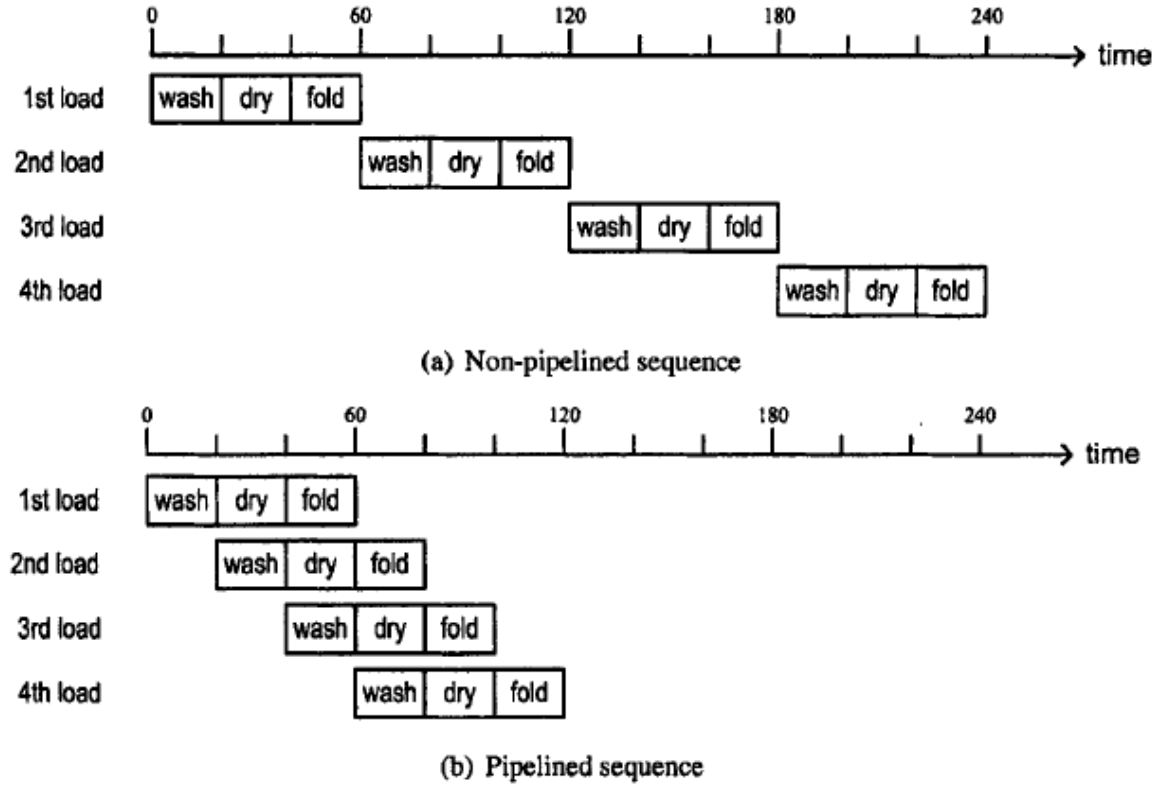


Figure 9.18 Timing diagrams of pipelined and non-pipelined laundry sequences.

Ejemplo de la lavandería

- Si cada etapa tarda 20', una carga completa se procesa en 60' y por lo tanto la "productividad" de la lavandería es de :

$$\frac{1}{60}$$ por minuto

  - Si se procesaran k cargas se tardaría: 40+20k minutos. Y la productividad sería de :

$$\frac{k}{40+20k}$$

- En el otro caso, cada etapa sigue tardando 20', pero como las etapas usan recursos distintos, apenas quedan desocupadas pueden ser usadas para una nueva "colada".
- En este caso, una carga completa tarda 60 ', pero la productividad del sistema (para 4 cargas) es de : $\frac{2}{60}$

Ejemplo de la lavandería

- Si las etapas no duraran lo mismo, por ejemplo:
  - Lavar: 15
  - Secar: 25
  - Doblar: 20

  Entonces: el tiempo de una carga (latencia) sería :
  - En la secuencial: 60 minutos
  - En la segmentada: 75 minutos

  Y la productividad sería:

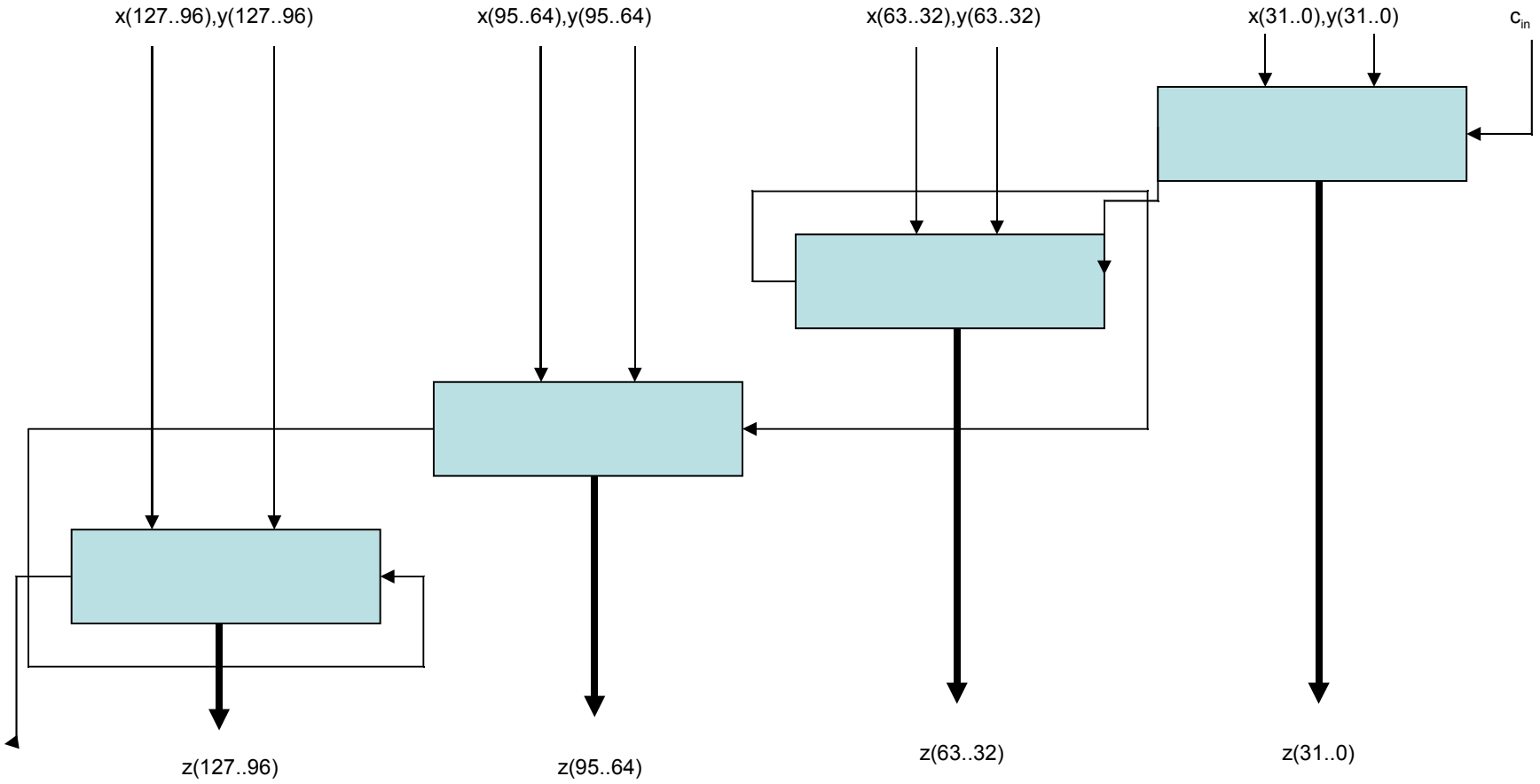
  En la secuencial : $\frac{1}{60}$

  En la segmentada: $\frac{k}{50 + 25k}$

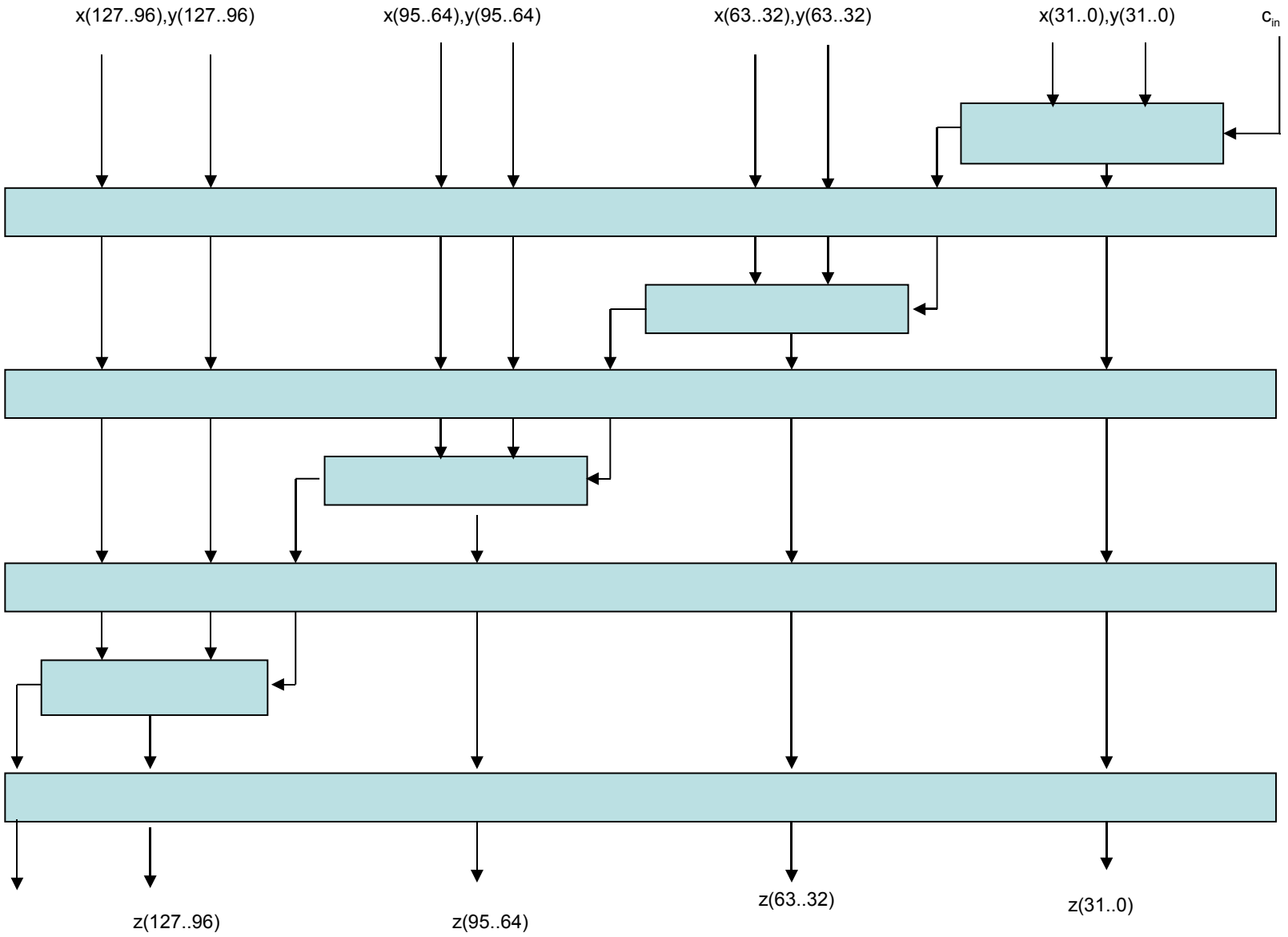
- Si k es muy grande, entonces la productividad es de $\frac{1}{25}$

Implementación en Pipeline

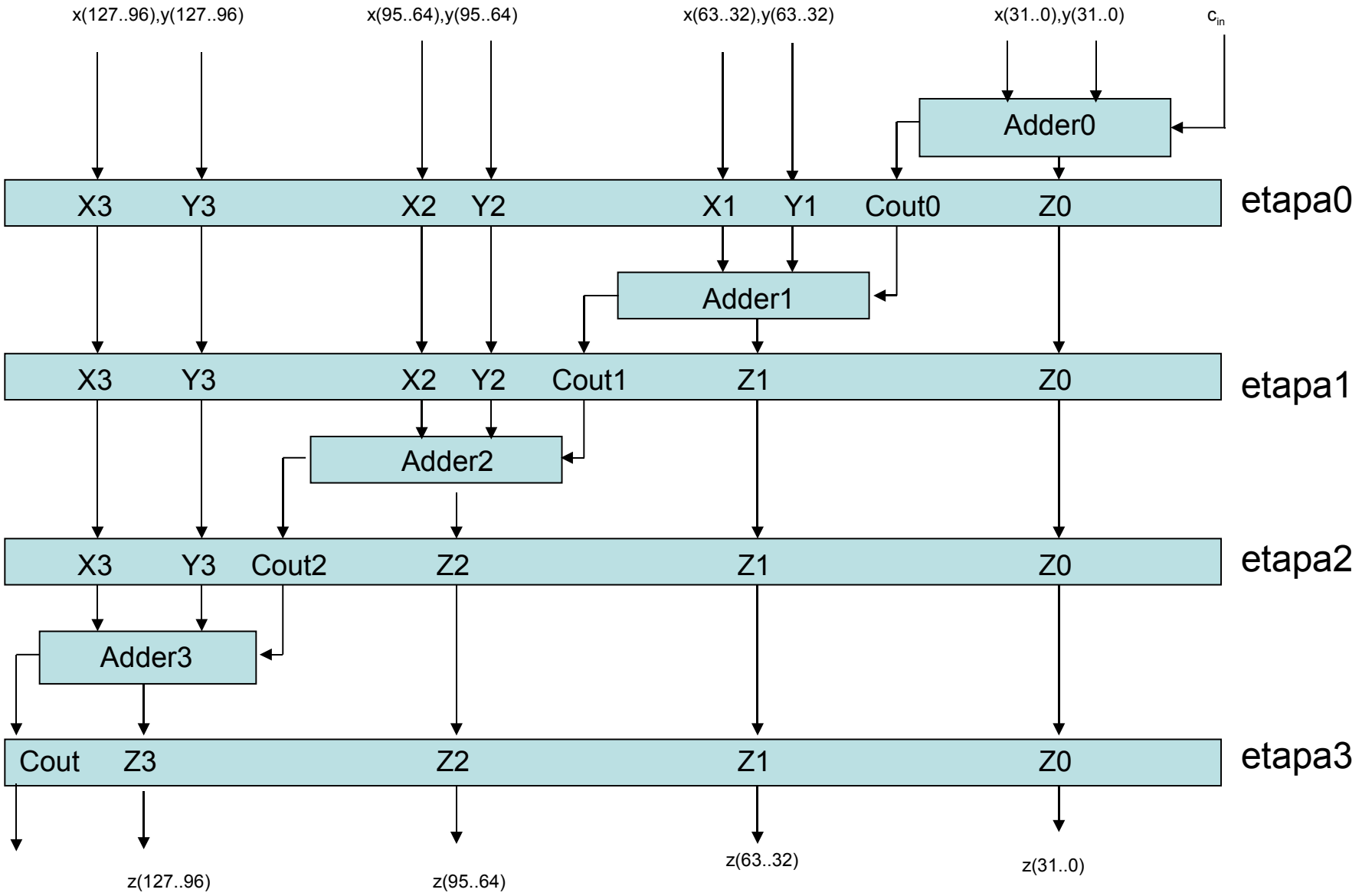
- Partimos de la versión paralela, o sea combinacional:



Implementación en Pipeline



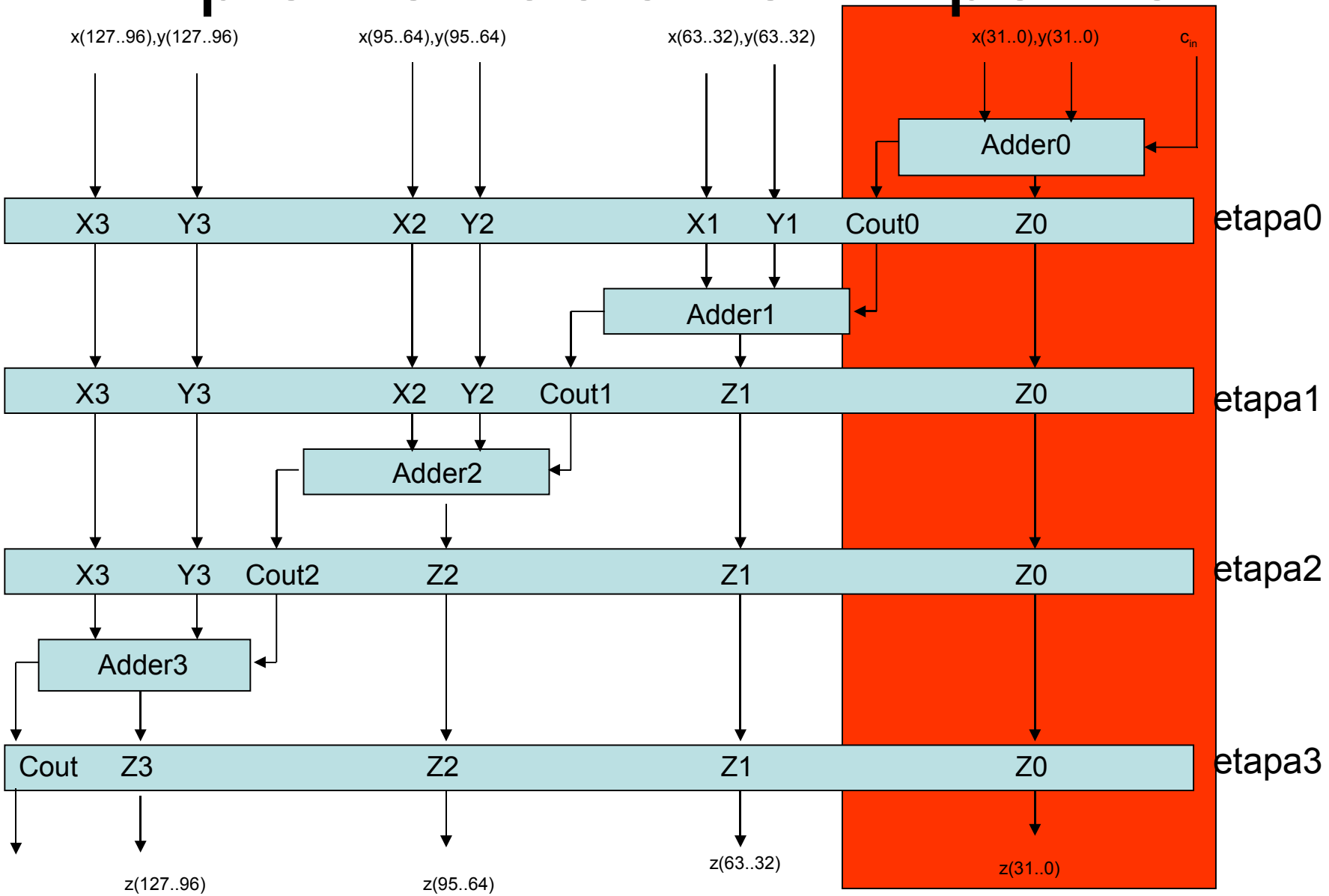
Implementación en Pipeline



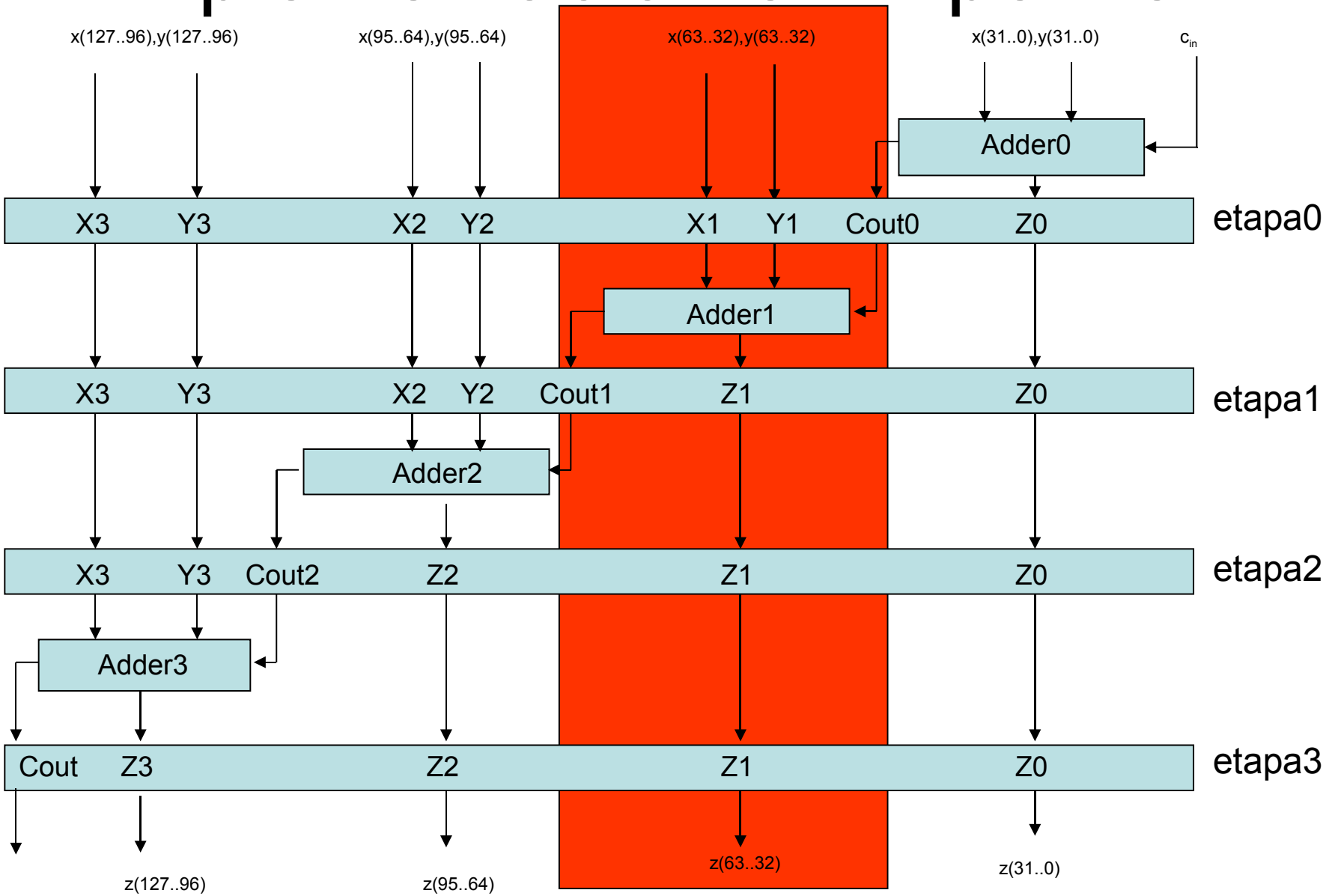
Vista del Pipeline

Tiempo	sumador0	sumador1	sumador2	sumador3
1	X0(0)+Y0(0)			
2	X1(0)+Y1(0)	X0(1)+Y0(1)		
3	X2(0)+Y2(0)	X1(1)+Y1(1)	X0(2)+Y0(2)	
4	X3(0)+Y3(0)	X2(1)+Y2(1)	X1(2)+Y1(2)	X0(3)+Y0(3)
5	X4(0)+Y4(0)	X3(1)+Y3(1)	X2(2)+Y2(2)	X1(3)+Y1(3)
...				
n	Xn(0)+Yn(0)	Xn-1(1)+Yn-1(1)	Xn-2(2)+Yn-2(2)	Xn-3(3)+Yn-3(3)
n+1	---	Xn(1)+Yn(1)	Xn-1(2)+Yn-1(2)	Xn-2(3)+Yn-2(3)
n+2	---	---	Xn(2)+Yn(2)	Xn-1(3)+Yn-1(3)
n+3	---	---	---	Xn(3)+Yn(3)

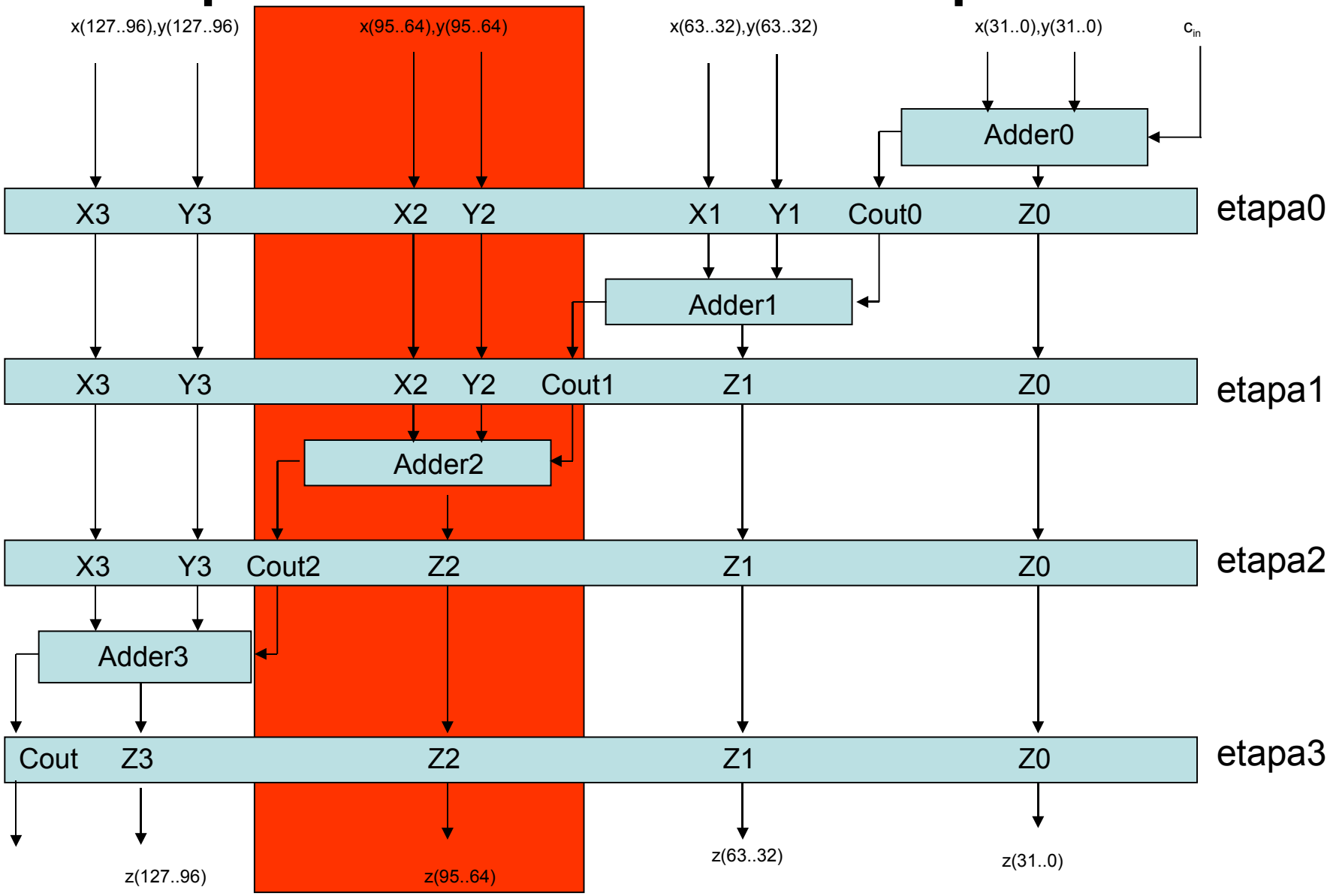
Implementación en Pipeline



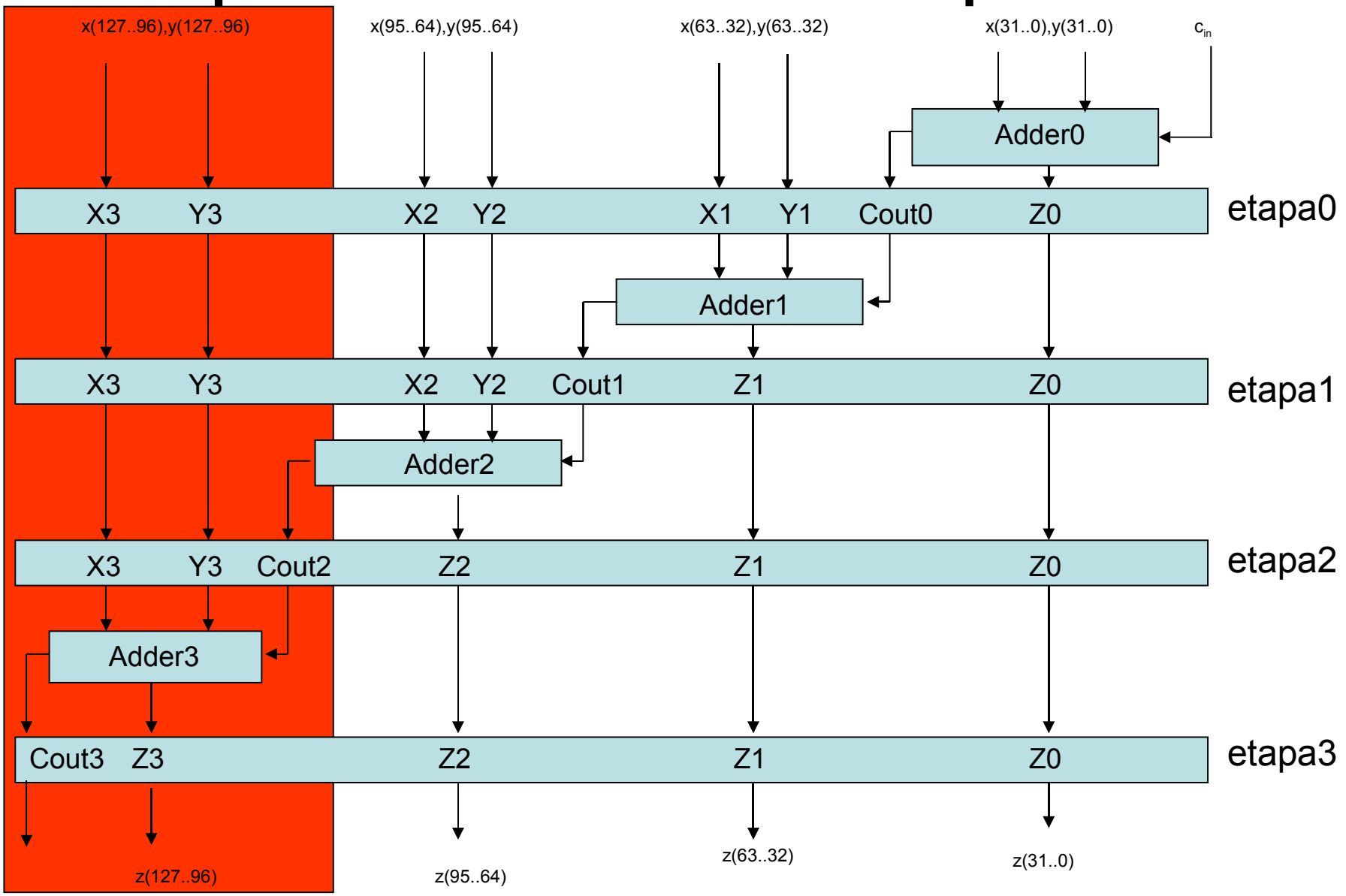
Implementación en Pipeline



Implementación en Pipeline



Implementación en Pipeline



Entonces …

- Hacer una versión en pipeline del sumador de 128 bits, y compararla con la versión secuencial y con la versión paralela.