Pipeline

Diseño de Sistemas con FPGA

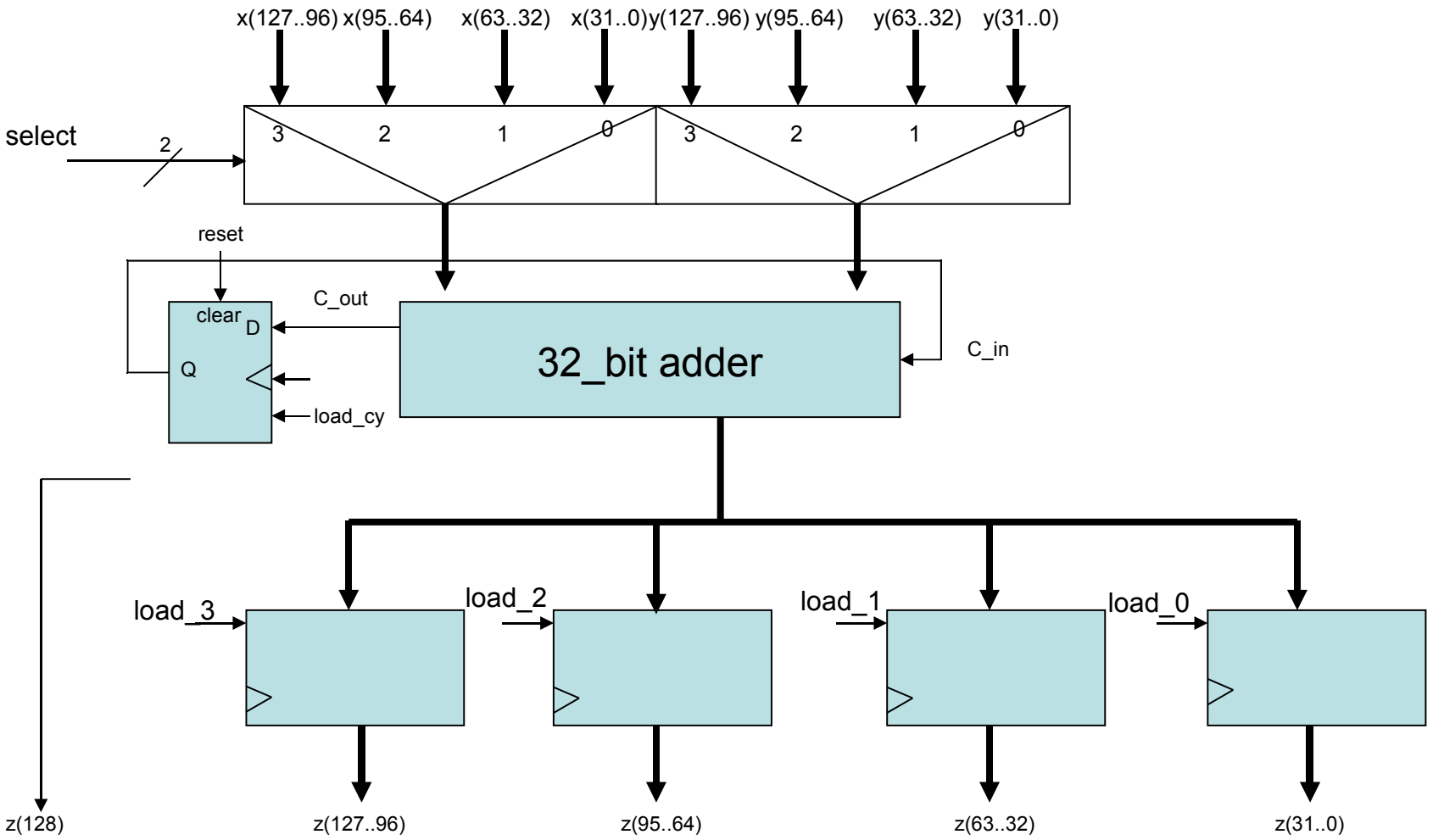
1er cuatrimestre 2009

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).

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

- Costo Paralelo:

  $C = 4 * C_{adder(32)} + 129\, C_{FF}$

- Retardo Paralelo

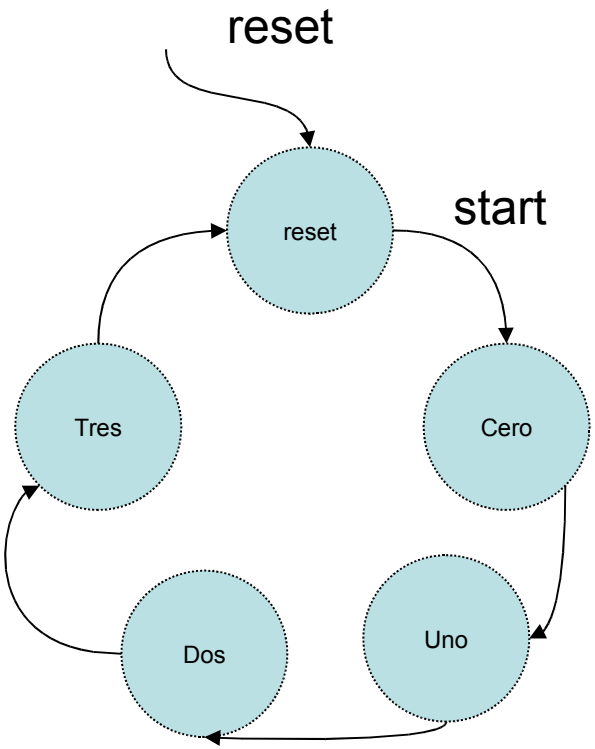
  $T = T_{clk}$ donde $T_{clk} > T_{adder(128)} + T_{FF}$

Implementación secuencial

- Señales de control del camino de datos:
  - select : dos bits
  - load_cy
  - load_3, load_2, load_1, load_0
- Señales externas:
  - Entradas: reset, start, clk, x, y
  - Salidas: done, z.

```
module sumador_m*n_seq
#(parameter n=8,
                    m=4,
 )
     (input wire[m-1:0] x, y[n-1:0],
      input wire clk, start, reset,
      output wire done;
      output reg[m-1:0] z[n-1:0],
      output carry
);
```

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

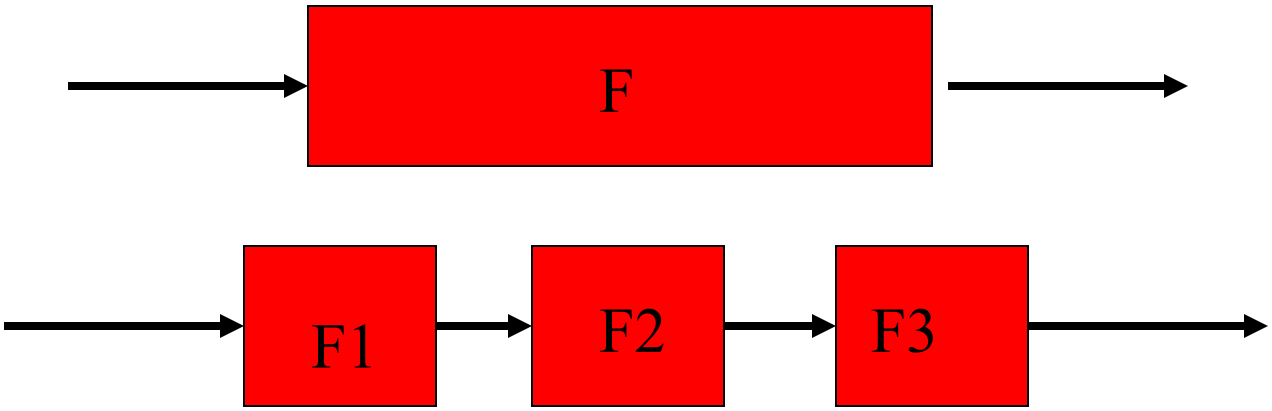
Pipeline

- 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

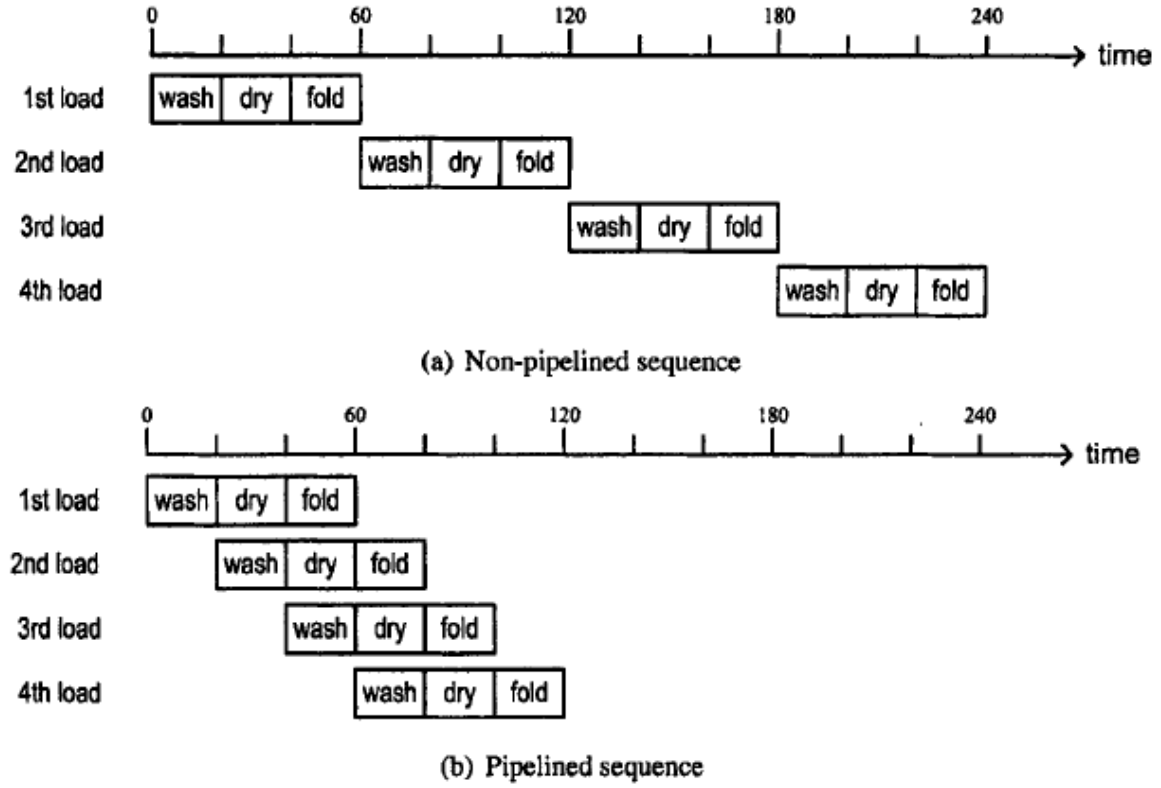


Figure 9.18 Timing diagrams of pipelined and non-pipelined laundry sequences.

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:

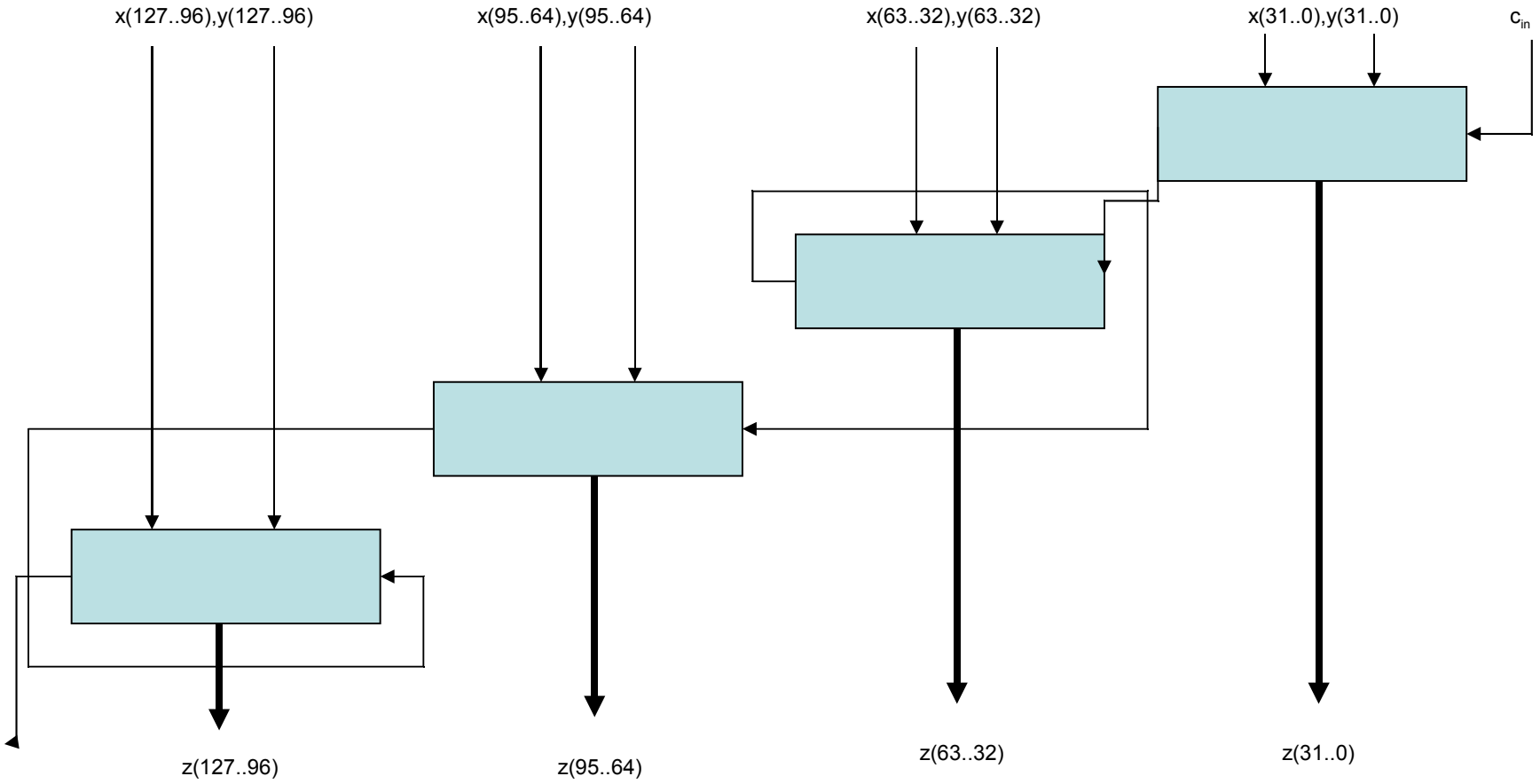
  En la secuencial : $\frac{1}{60}$

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

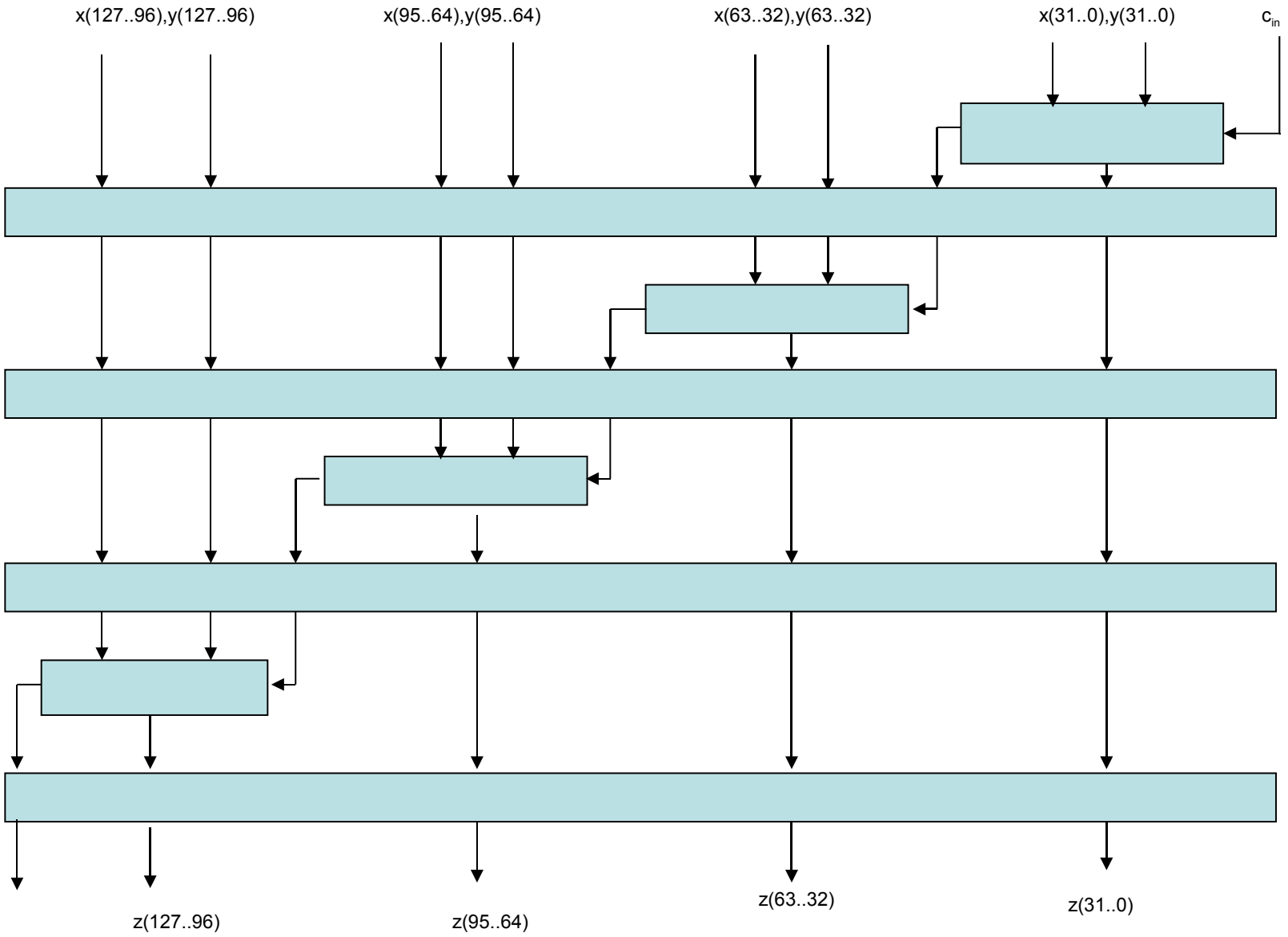
- Si k es muy grande, entonces la productividad es de $\frac{1}{25}$

Implementación en Pipeline

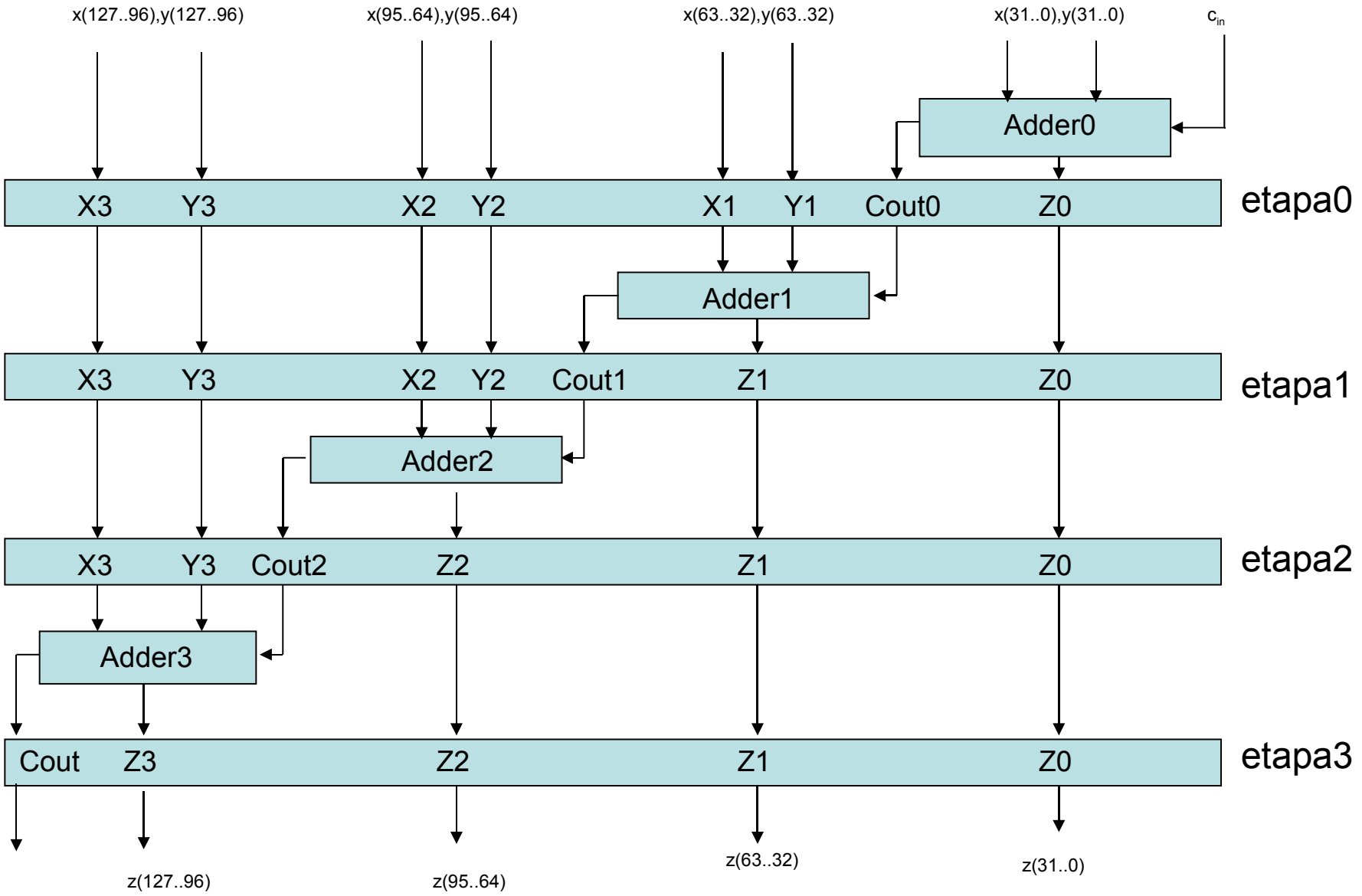
- Partimos de la versión paralela, o sea combinacional:



Implementación en Pipeline



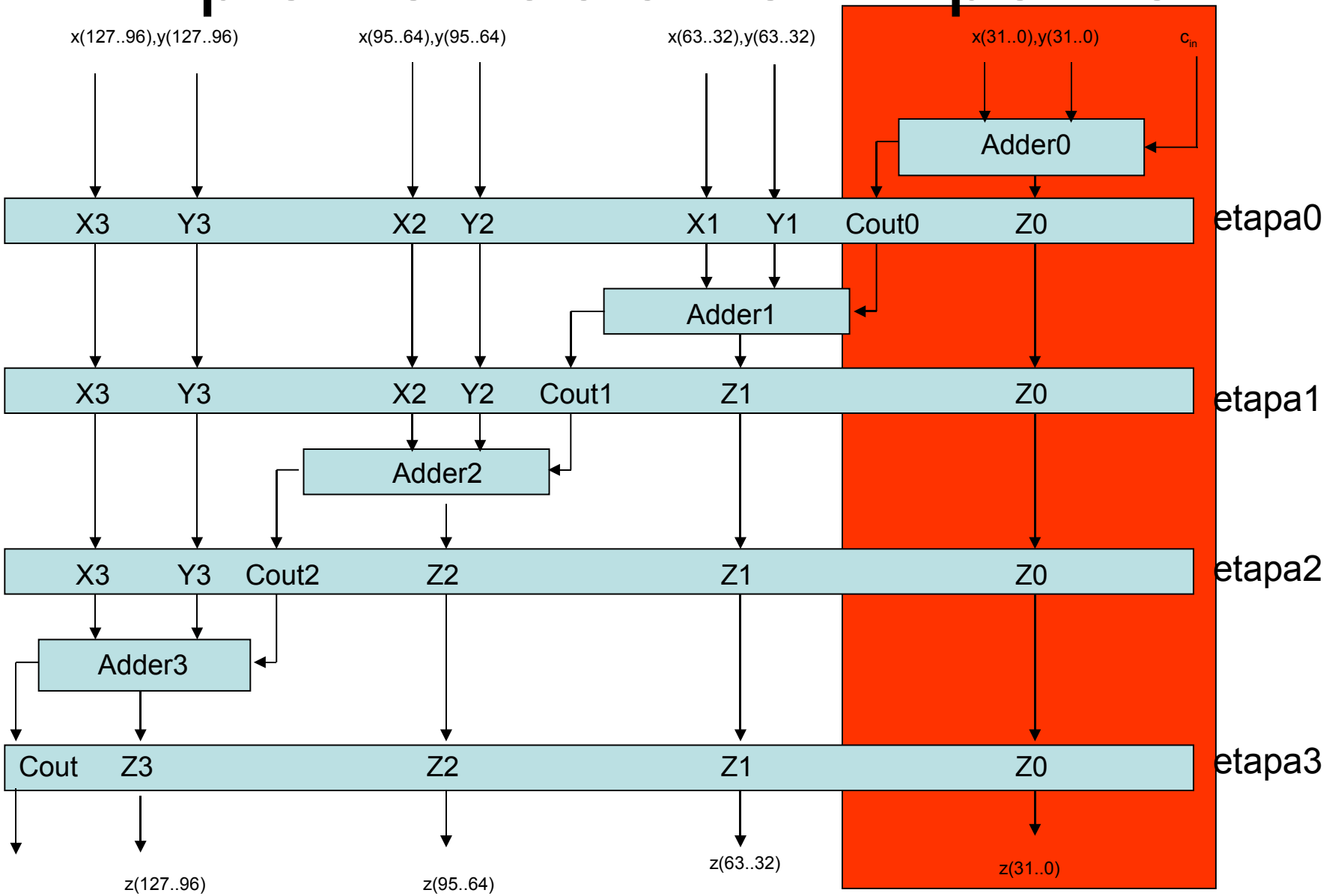
Implementación en Pipeline



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

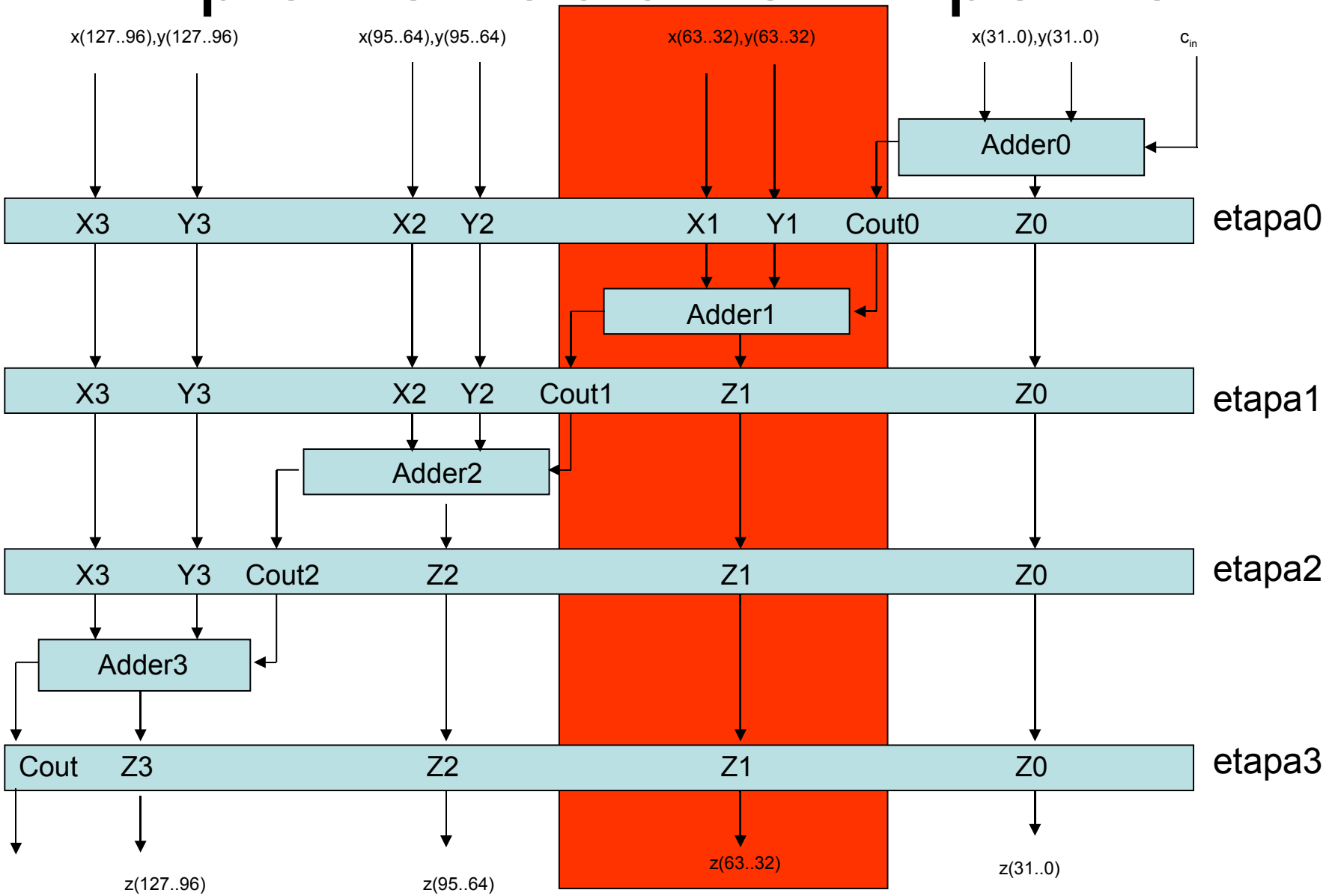


Camino de Datos

```
// instanciación del sumador de 32 bits

reg [3:0] Z0[31:0];

  begin
----- etapa 0 -----
------sumador------

suma0 sumador_32
(.x(x(0)), .y(y(0), .cin(cin), .z(z),.cout(cout));
```

```
always @ (posedge clk, posedge reset)
                if reset
                   cout0 <= 1´b0;
                 else
                    cout0 <= cout;


always @ (posedge clk, posedge reset)
/// para Z0
                if reset
                   for (i=0,i<4,i=i+1)
                      Z0(i) <= 0;
                 else
                    begin
                    for (i=1,i<3,i=i+1)
                       Z0(i) <=Z0(i-1) ;
                    Z0(0) <=z;
                    end;
```

Implementación en Pipeline

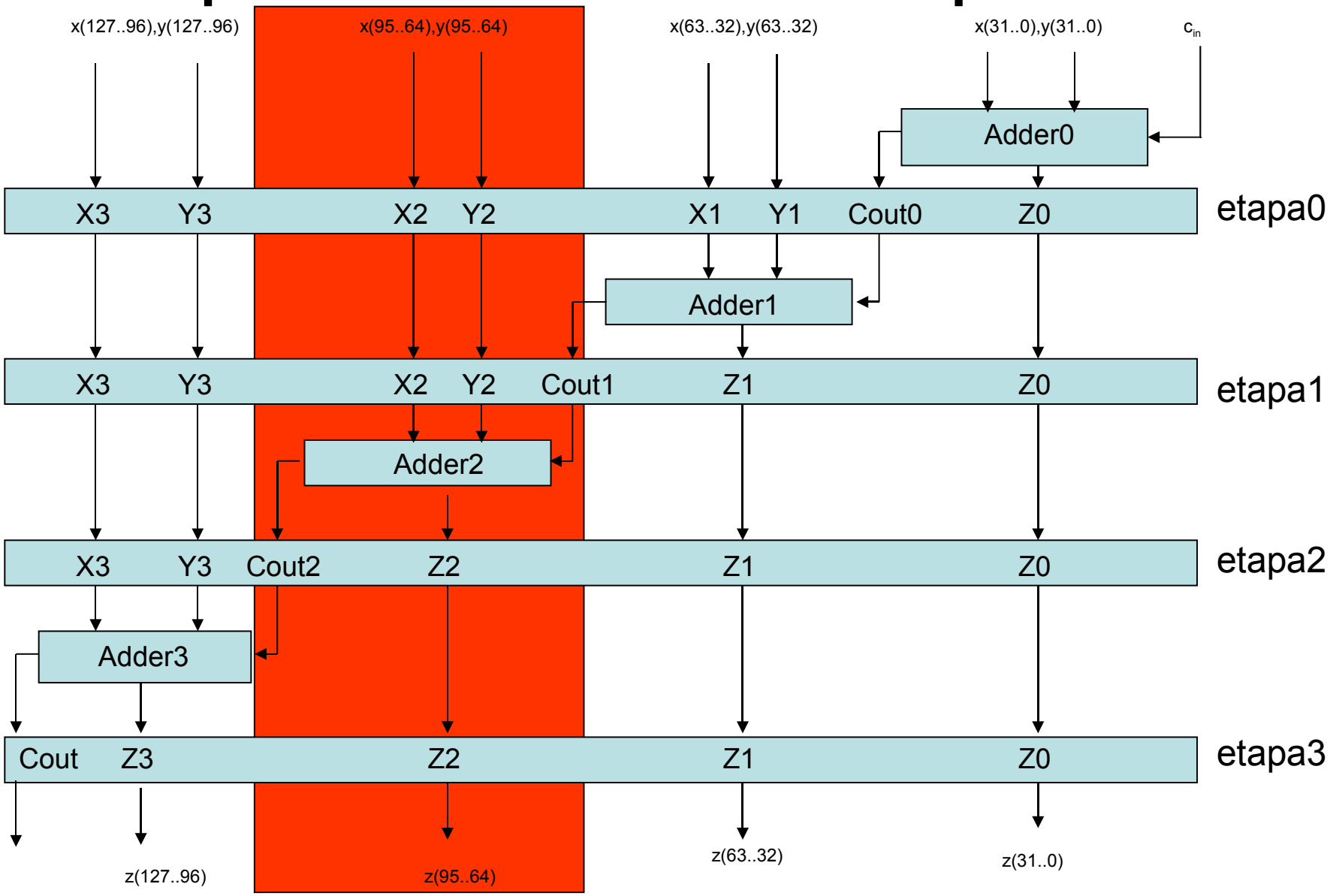


Camino de Datos

```
…
reg [2:0] Z1[31:0];
reg  X1[31:0];
reg  X2[31:0];
reg cout1;

  …

  begin
…
----- etapa 1 -----
------sumador------

suma1 sumador_32
(.x(X1), .y(Y1), .cin(cout0), .z(z),.cout(cout));
```

```
process(clk,reset)
---- para el carry out de la segunda etapa------
begin
        if reset=´1´ then cout1<=´0´;
        elsif clk'event and clk=´1´ then cout1<=cout; end if;
        end if;
end process

process(clk,reset)
---- para Z1-----
begin
        if clk'event and clk=´1´ then
                for i in 1..2 loop  Z1(i)<=Z1(i-1); end loop;
                Z1(0)<=z;
       end if;
end process

process(clk, reset)
----para X1 y Y1-----
begin
if clk'event and clk=´1´ then
   X1<=x(1);
   Y1<=y(1);
end if;
end process;
```

Implementación en Pipeline

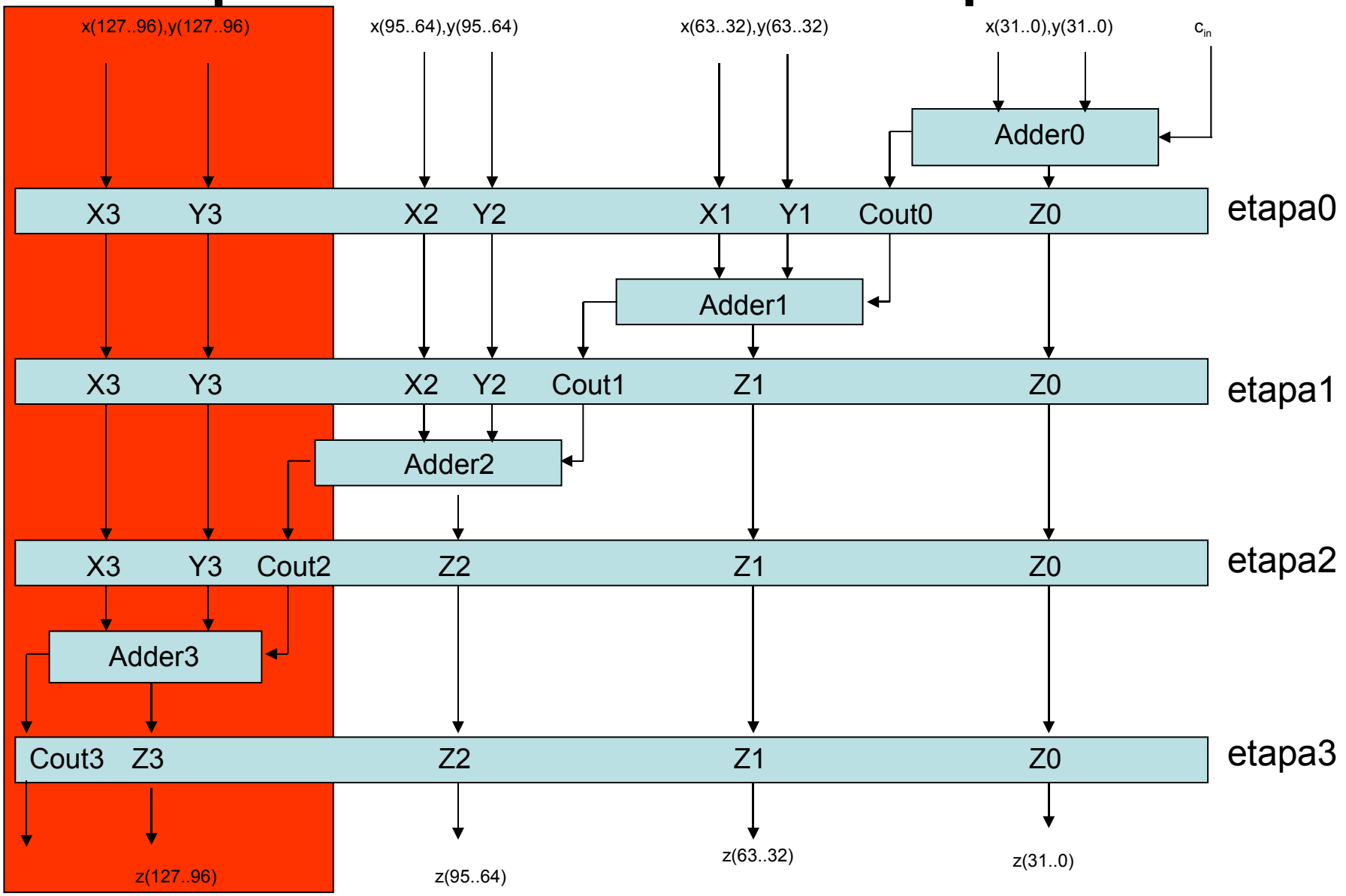


Camino de Datos

```
architecture pipeline of sumador_128 is
…
  signal Z2: array(1 downto 0) of
std_logic_vector(31 downto 0);
 signal X2: array(1 downto0) of
std_logic_vector(31 downto 0);
 signal Y2: array(1 downto 0) of
std_logic_vector(31 downto 0);
signal cout2: std_logic;

  …

  begin
…
----- etapa 3 -----
------sumador------

suma2: entity work.sumador_32 (arch)
port map (x=>X(2), y=>Y(2), cin=>cout1, z=>z,
cout=>cout);
```

```
process(clk,reset)
---- para el carry out de la cuarta etapa------
begin
        if reset=´1´ then cout1<=´0´;
        elsif clk'event and clk=´1´ then cout2<=cout; end if;
        end if;
end process;

process(clk,reset)
---- para Z3-----
begin
        if clk'event and clk=´1´ then
                Z2(1)<=Z2(0);
                Z2(0)<=z;
       end if;
end process

process(clk, reset)
----para X3 y Y3-----
begin
if clk'event and clk=´1´ then
   X2(1)<=X2(0);
   Y2(1)<=Y2(0);
   X2(0)<=x(2);
   Y2(0)<=y(2);
 end if;
end process;
```

Implementación en Pipeline

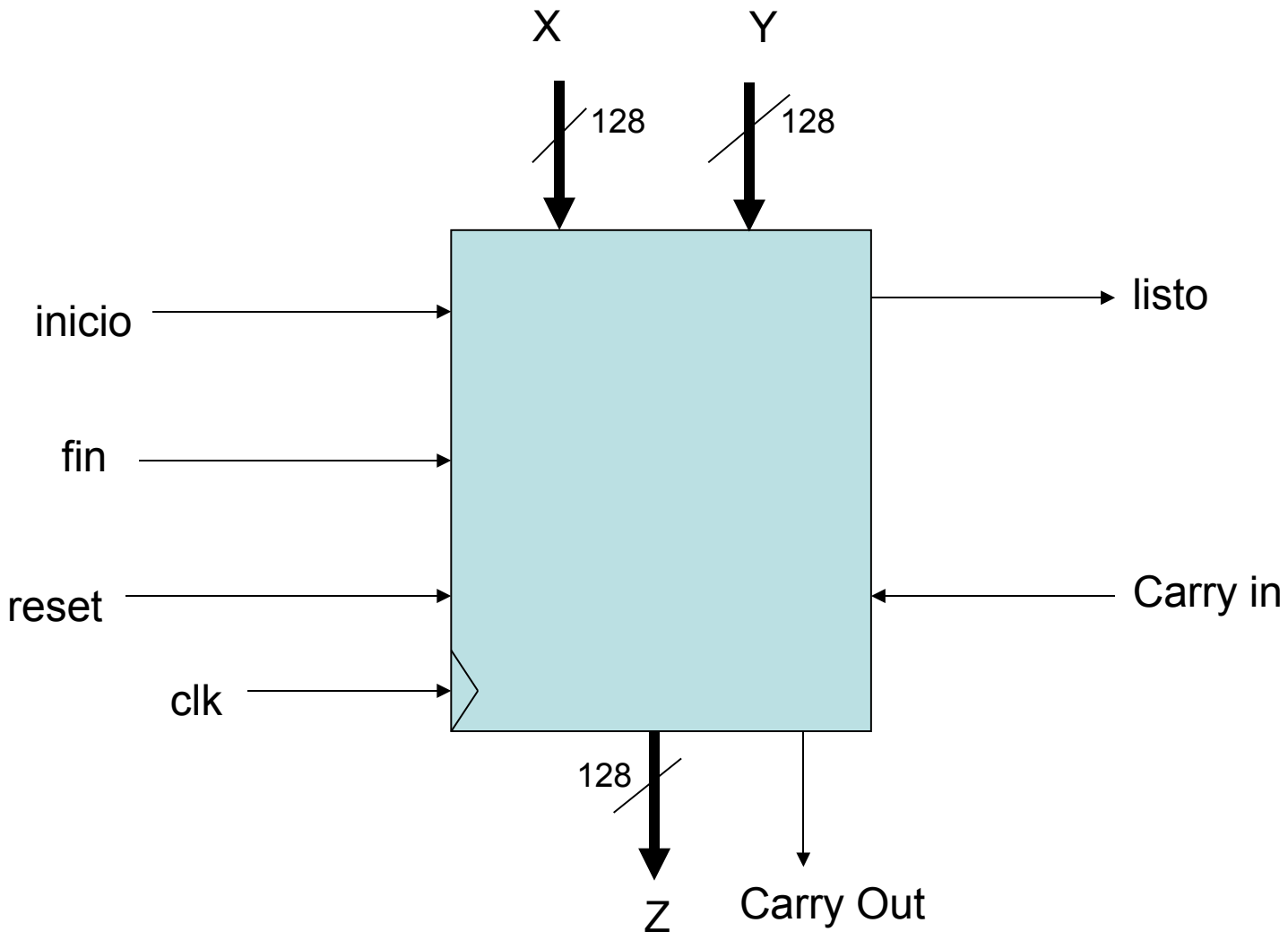


Camino de Datos

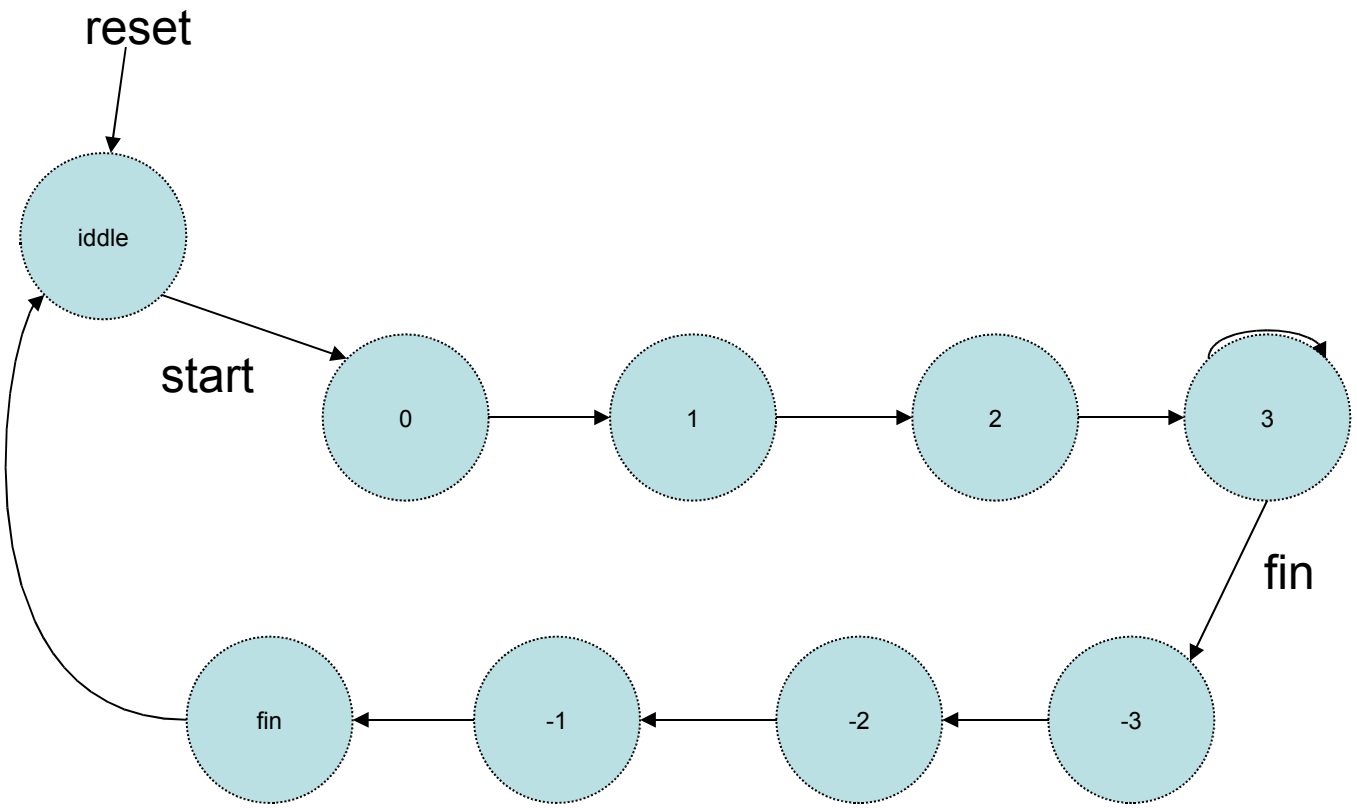
```
architecture pipeline of sumador_128 is
…
  signal Z3: std_logic_vector(31 downto 0);
 signal X3: array(2 downto0) of
std_logic_vector(31 downto 0);
 signal Y3: array(2 downto 0) of
std_logic_vector(31 downto 0);
signal cout3: std_logic;

  …

  begin
…
----- etapa 4 -----
------sumador------

suma3: entity work.sumador_32 (arch)
port map (x=>X(3), y=>Y(3), cin=>cout2, z=>z,
cout=>cout);
```

```
process(clk,reset)
---- para el carry out de la cuarta etapa------
begin
        if reset=´1´ then cout1<=´0´;
        elsif clk'event and clk=´1´ then cout3<=cout; end if;
        end if;
end process;

process(clk,reset)
---- para Z3-----
begin
        if clk'event and clk=´1´ then
                Z3<=z;
       end if;
end process

process(clk, reset)
----para X3 y Y3-----
begin
if clk'event and clk=´1´ then
                for i in 1..2 loop  X3(i)<=X3(i-1); end loop;
                X3(0)<=x(3);
                for i in 1..2 loop  Y3(i)<=Y3(i-1); end loop;
                Y3(0)<=y(3);
end if;
end process;
```

Sumador 128



Grafo de Estados para el Control



Los estados 0,1,2, iddle tiene la salida done=0

Los estados 3,-3,-2,-1 y fin tienen la salida done=1

Entonces …

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