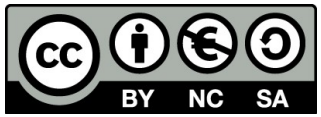
Laboratorio 12:

Uso de funciones y procedimientos

procesado de vídeo en tiempo real

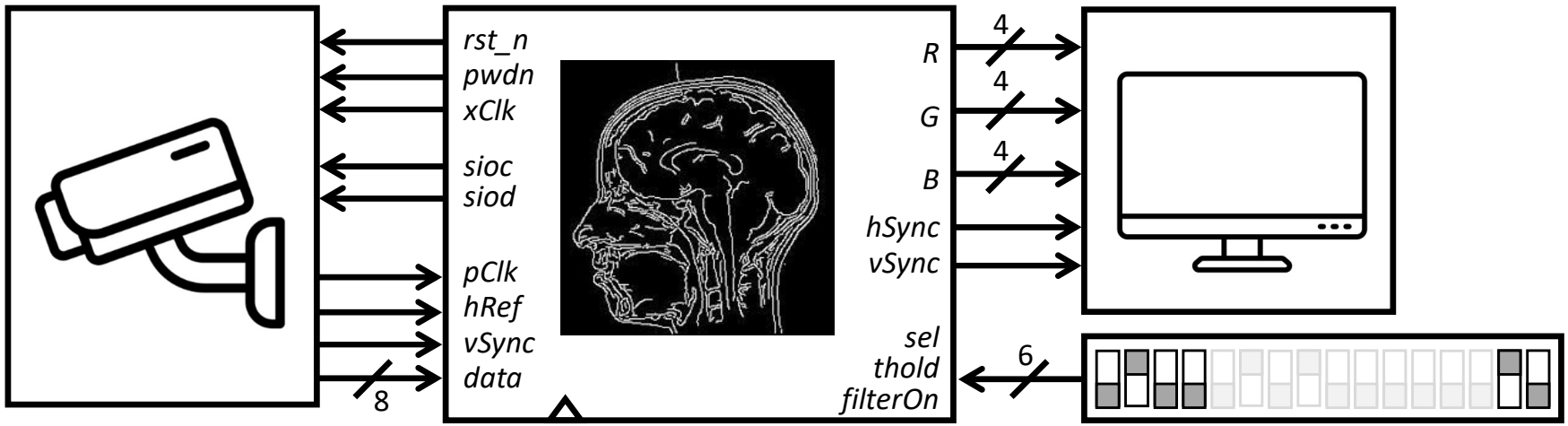
Diseño automático de sistemas

José Manuel Mendías Cuadros
Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid



Presentación

- Diseñar un procesador digital de vídeo elemental:
  - En tiempo real detectará los bordes de la imágenes recibidas por una cámara digital.
  - Su comportamiento será diferente según el valor de los switch 0:
    - Si 0, se visualizan las imágenes recibidas (viéndose video original).
    - Si 1, se visualizan las imágenes recibidas una vez procesadas (viéndose video procesado).
  - El algoritmo de detección de bordes se elegirá según el valor de los switch 1:
    - Si 0, se aplicará el algoritmo de Prewitt.
    - Si 1, se aplicará el algoritmo de Sobel.
  - Ambos algoritmos aplican un umbral (thold) que lo fijan los switches 15 al 12.

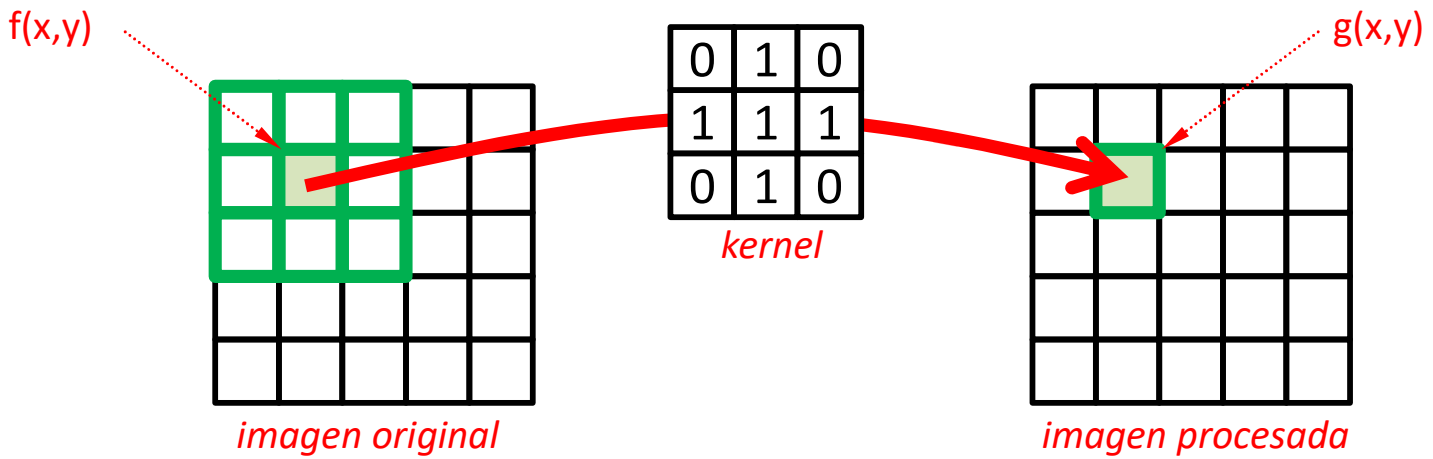


J.M. Mendías
2024

Procesado digital de imagen

presentación

- Los algoritmos espaciales de procesado digital de imagen transforman directamente los valores de los pixeles que forman la imagen.
  - Típicamente, cada pixel de la imagen procesada es una convolución (combinación lineal) de sus píxeles cercanos en la imagen original.
  - La matriz de coeficientes de la convolución se denomina kernel.



$$g(x,y) = kernel \cdot f(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} kernel(i,j) \cdot f(x-i, y-j)$$

J.M. Mendías
2024

Procesado digital de imagen

detección de bordes

- Un detector de bordes es un algoritmo que localiza en una imagen en escala de grises los cambios bruscos de brillo.
  - Es un paso previo común a muchos algoritmos de análisis de imagen.

- Algoritmo de detección de bordes de Prewitt:

$$g(x,y) = \begin{cases} 0 & si\ G \leq thold \\ G & en\ otro\ caso \end{cases}$$

$$donde:$$

$$G = \sqrt{(xKernel \cdot f(x,y))^2 + (yKernel \cdot f(x,y))^2}$$

+1	0	-1
+1	0	-1
+1	0	-1

xKernel

+1	+1	+1
0	0	0
-1	-1	-1

yKernel

- Algoritmo de detección de bordes de Sobel:

$$g(x,y) = \begin{cases} 0 & si\ G \leq thold \\ G & en\ otro\ caso \end{cases}$$

$$donde:$$

$$G = \sqrt{(xKernel \cdot f(x,y))^2 + (yKernel \cdot f(x,y))^2}$$

-1	0	+1
-2	0	+2
-1	0	+1

xKernel

-1	-2	-1
0	0	0
+1	+2	+1

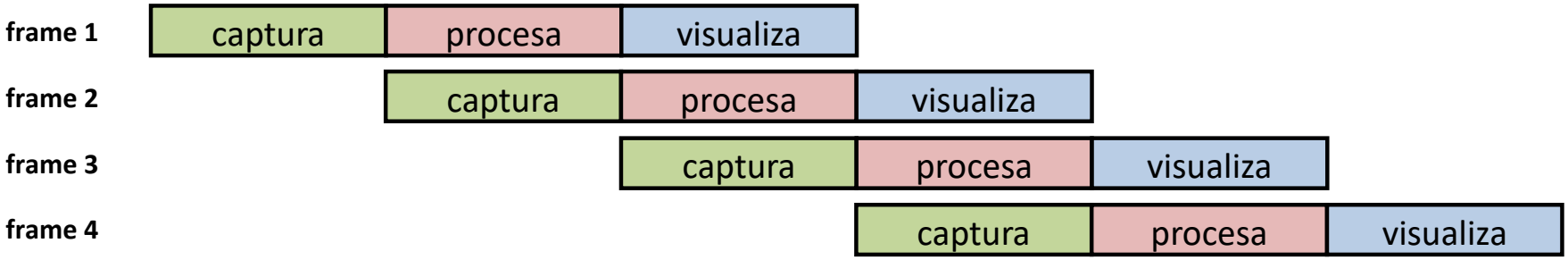
yKernel

J.M. Mendías
2024

Procesado digital de vídeo

arquitectura (i)

- Las arquitecturas de procesamiento de video en tiempo real:
  - Procesan individualmente cada *frame* que reciben.
  - Suelen segmentarse funcionalmente en, al menos, 3 etapas que se solapan.



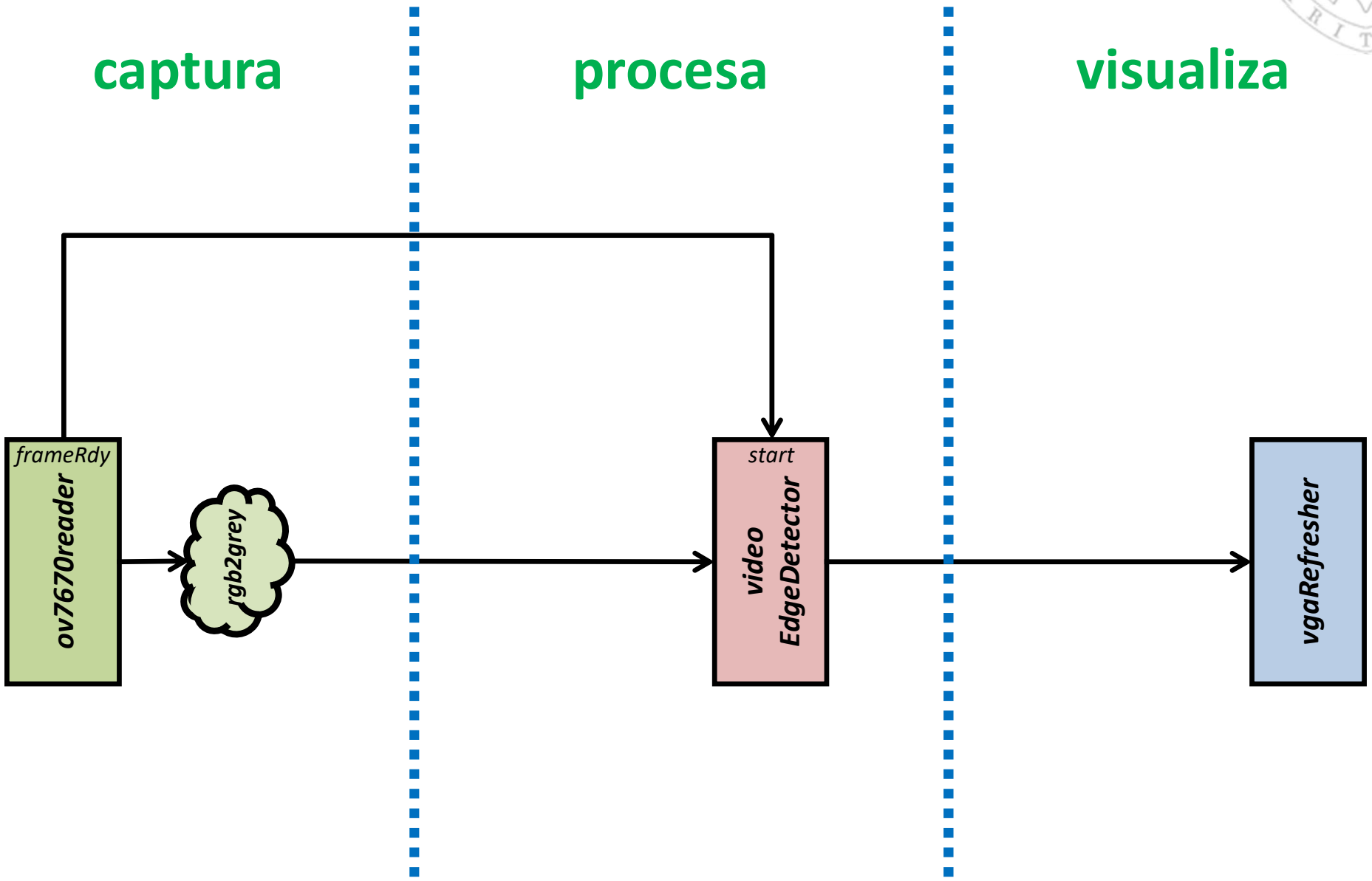
- Las etapas se separan por *bufferes* dobles (*ping-pong*).
  - *Frames* sucesivos se almacenan alternativamente en *buffers* distintos.
    - Mientras se escribe un *frame* en un *buffer*, el *frame* anterior se lee desde el otro.
    - Cuando se termina de escribir y leer un *frame* completo, los *buffers* intercambian de rol.
  - Necesario porque los pixeles se procesan en bloques y en orden distinto al de llegada.

PING	captura *frame* 1	procesa *frame* 1	captura *frame* 3	procesa *frame* 3	captura *frame* 5	procesa *frame* 5
PONG		captura *frame* 2	procesa *frame* 2	captura *frame* 4	procesa *frame* 4	captura *frame* 6

J.M. Mendías
2024

Procesado digital de vídeo

arquitectura (ii)

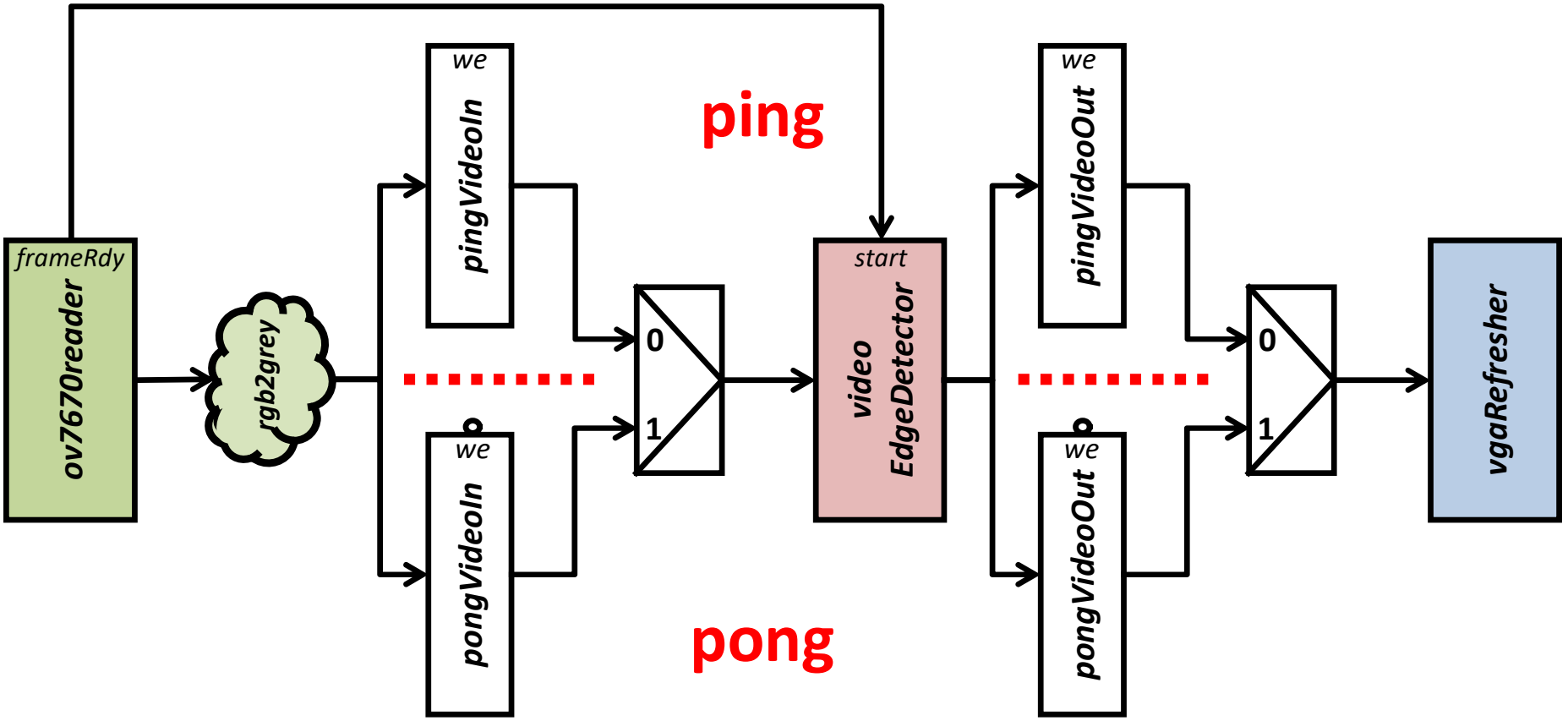


J.M. Mendías
2024

Procesado digital de vídeo

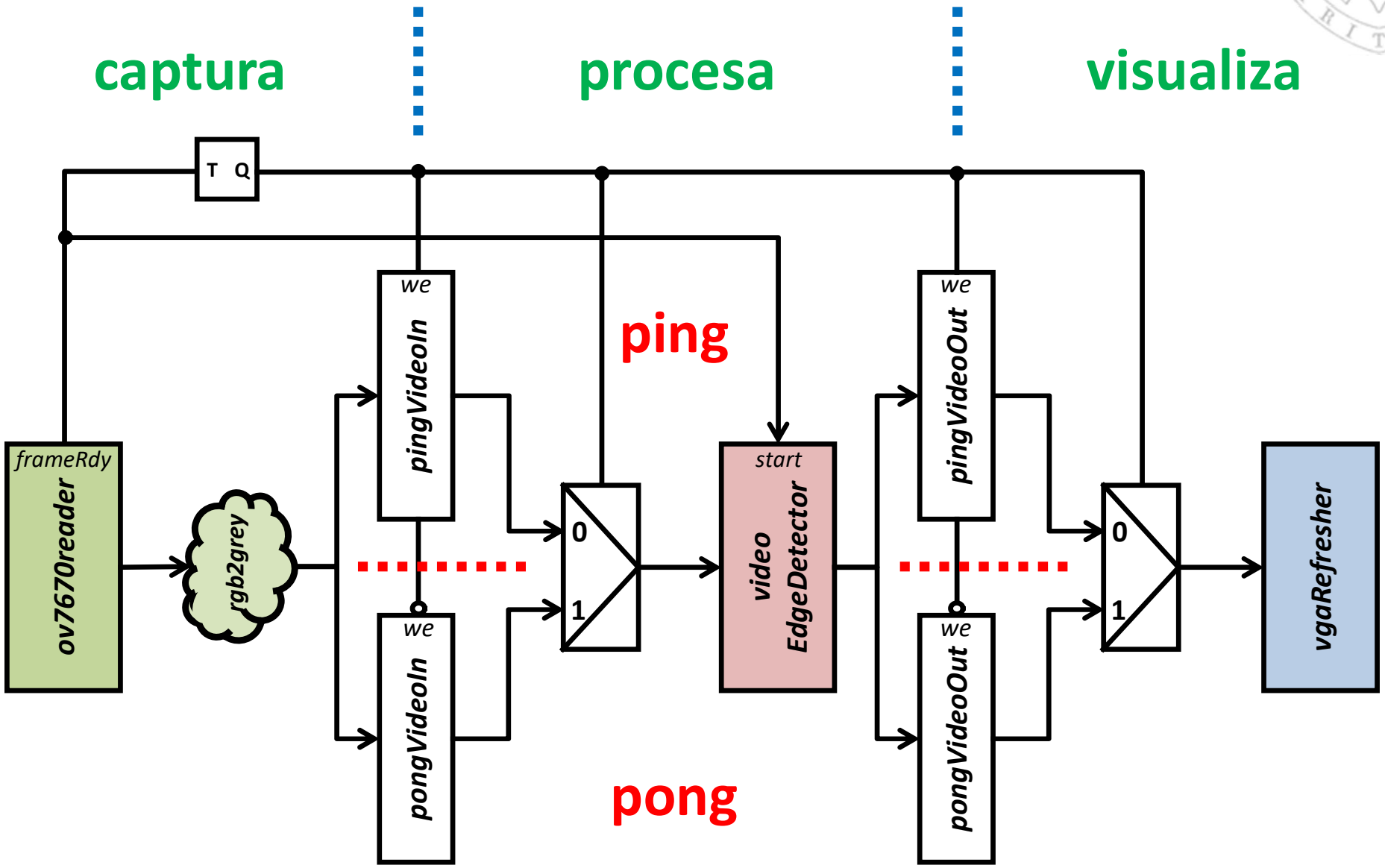
arquitectura (iii)

captura ⋮ procesa ⋮ visualiza



Procesado digital de vídeo

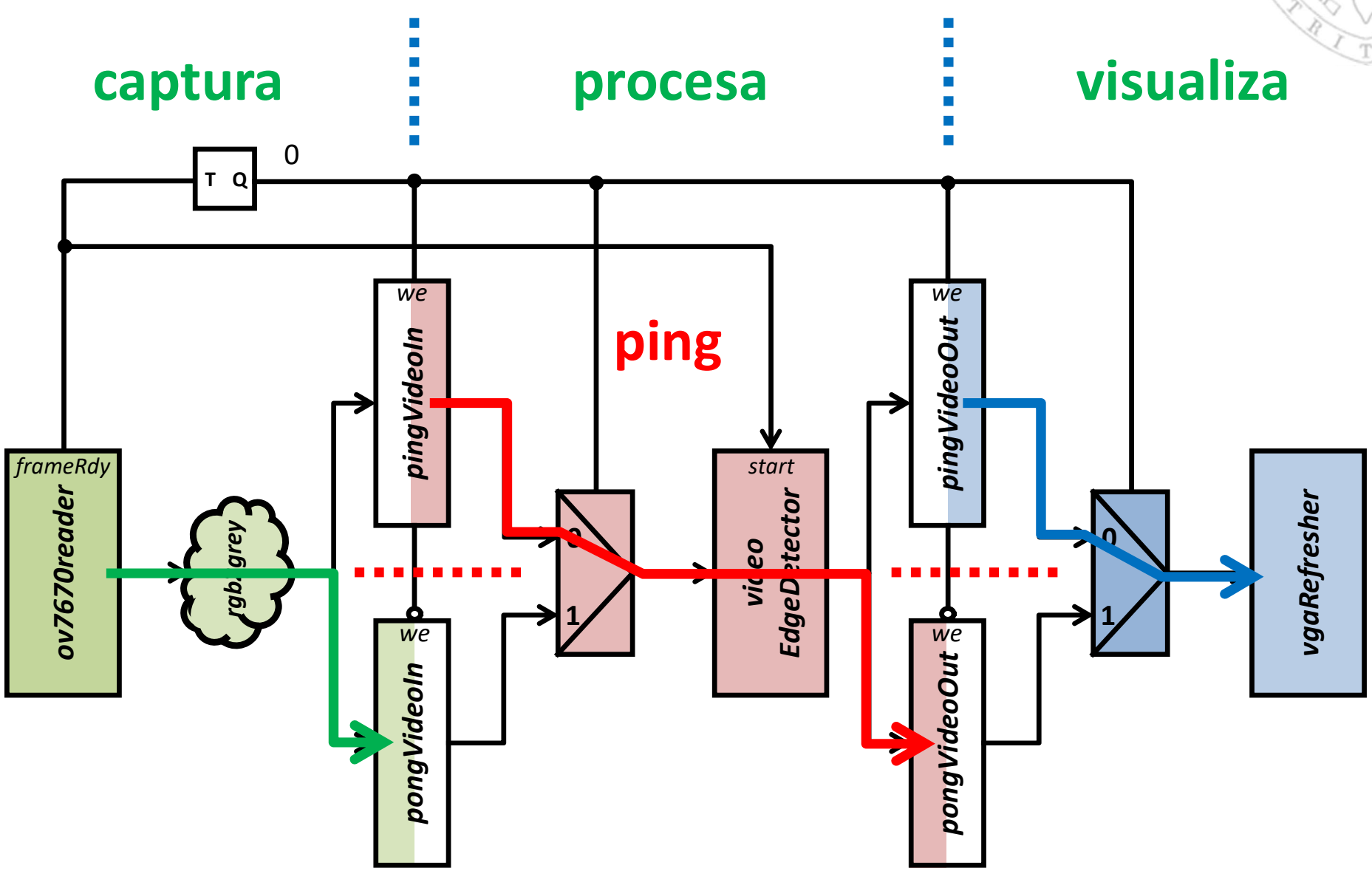
arquitectura (iv)



J.M. Mendías
2024

Procesado digital de vídeo

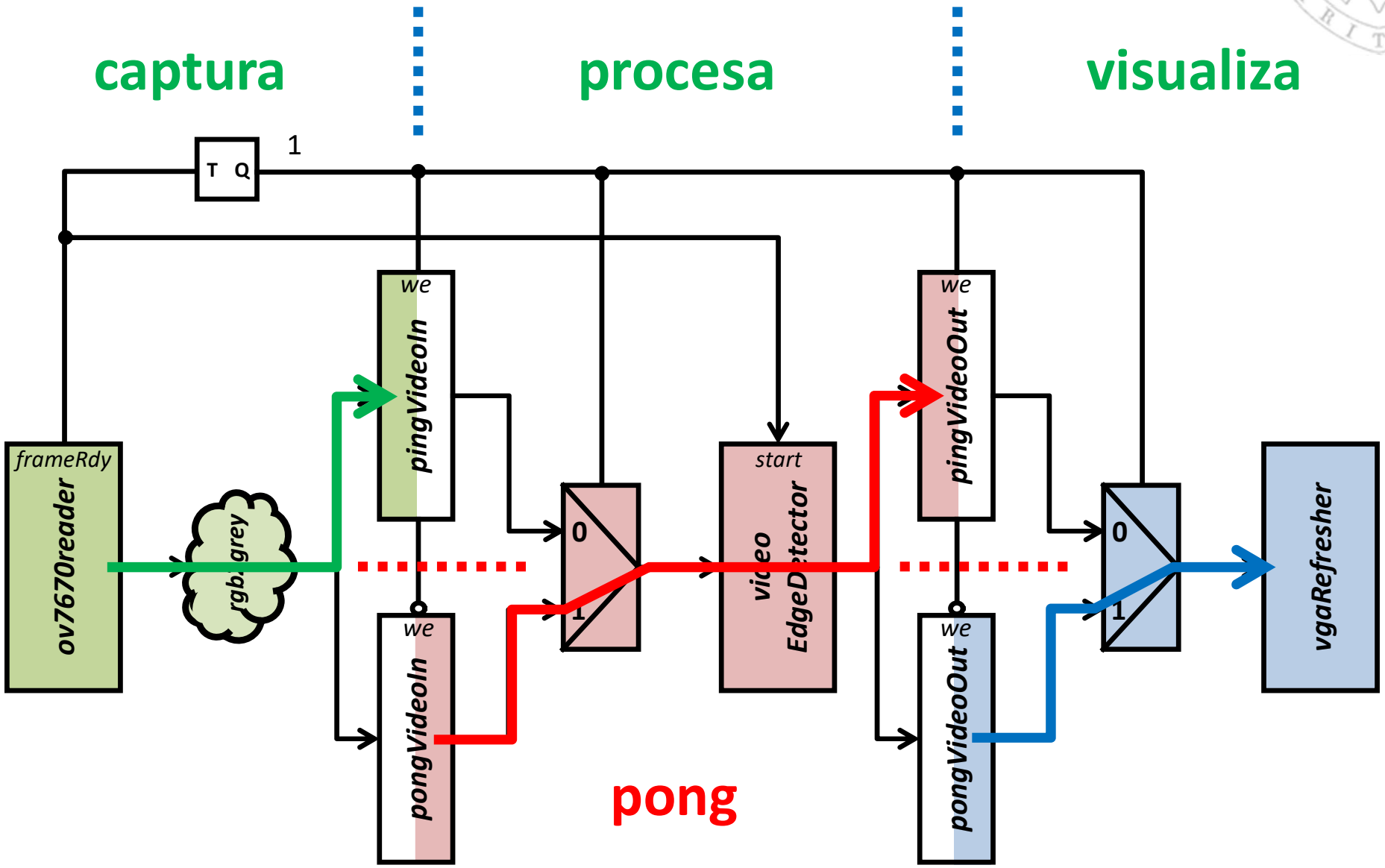
arquitectura (v)



J.M. Mendías
2024

Procesado digital de vídeo

arquitectura (vi)



Diseño principal

dimensionamiento de *buffers*

- El diseño requiere 4 *buffers* de vídeo en escala de grises:
  - La FPGA de la placa dispone de 1800 Kb de memoria Block RAM.
  - Deberemos trabajar con imágenes de tamaño QVGA.

tamaño	# pix	gris 4b/px	× 4
VGA 640×480	307200	1200 Kb	**4800 Kb**
QVGA 320×240	76800	300 Kb	**1200 Kb**
QQVA 160×120	19200	75 Kb	**300 Kb**

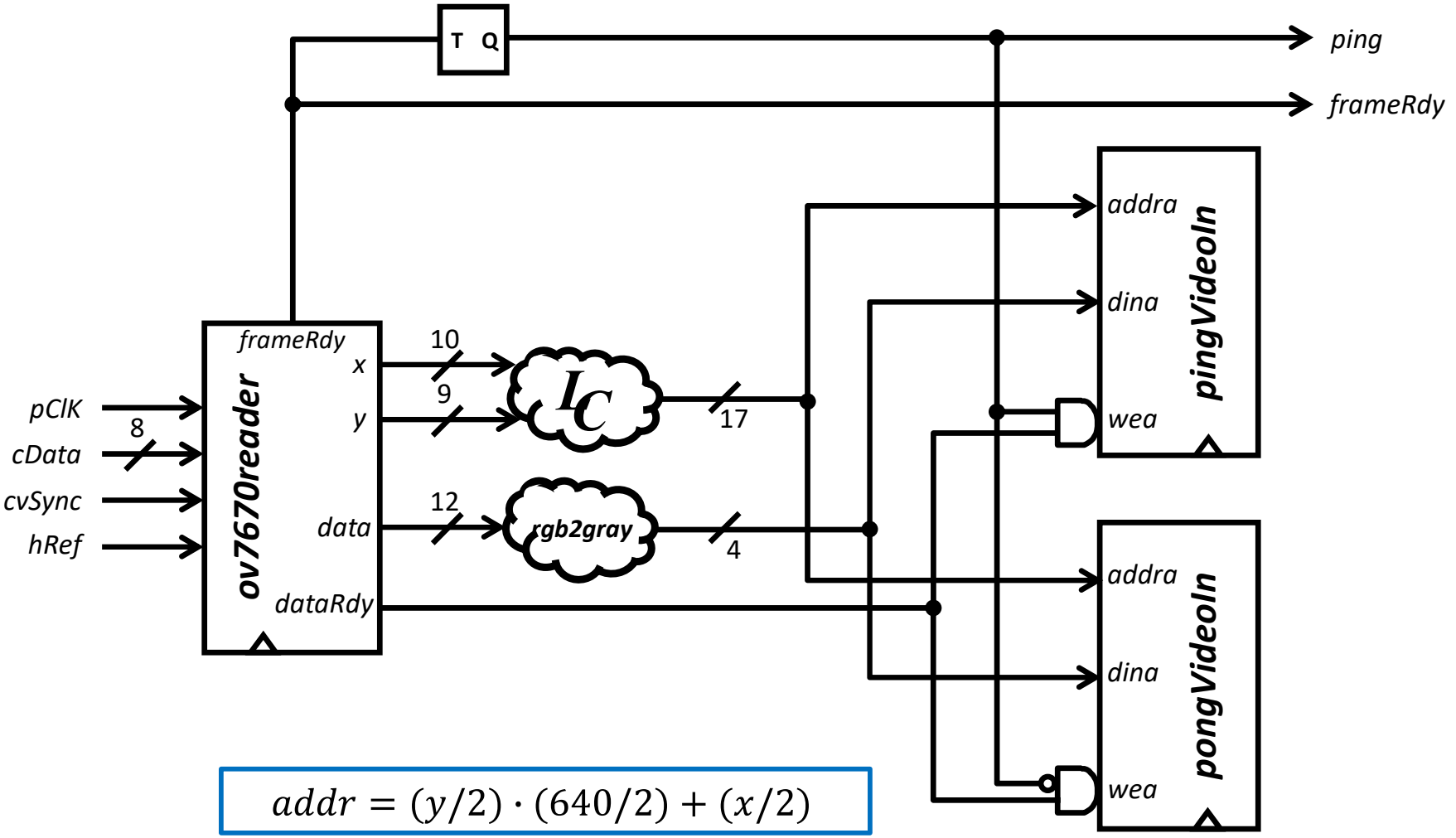
- La dirección del buffer que ocupa un pixel en posición (x,y) se calcula:

tamaño	# pix	# bits x	# bits y	# bits addr	addr
QVGA 320×240	76800	9	8	17	(640/2)·(y/2) + (x/2)

J.M. Mendías
2024

Diseño principal

etapa de captura: esquema RTL

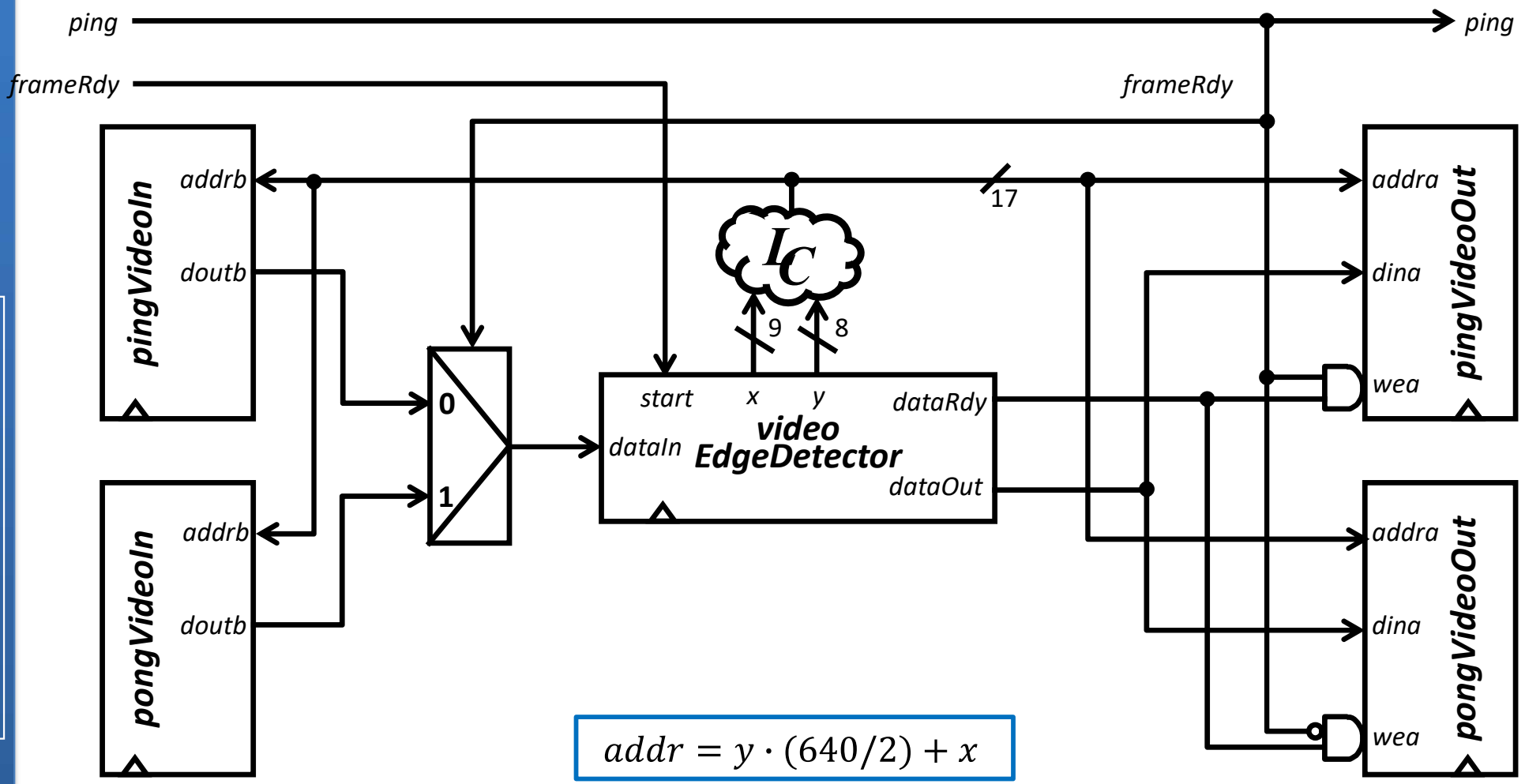


$$addr = (y/2) \cdot (640/2) + (x/2)$$

J.M. Mendías
2024

Diseño principal

etapa de procesado: esquema RTL

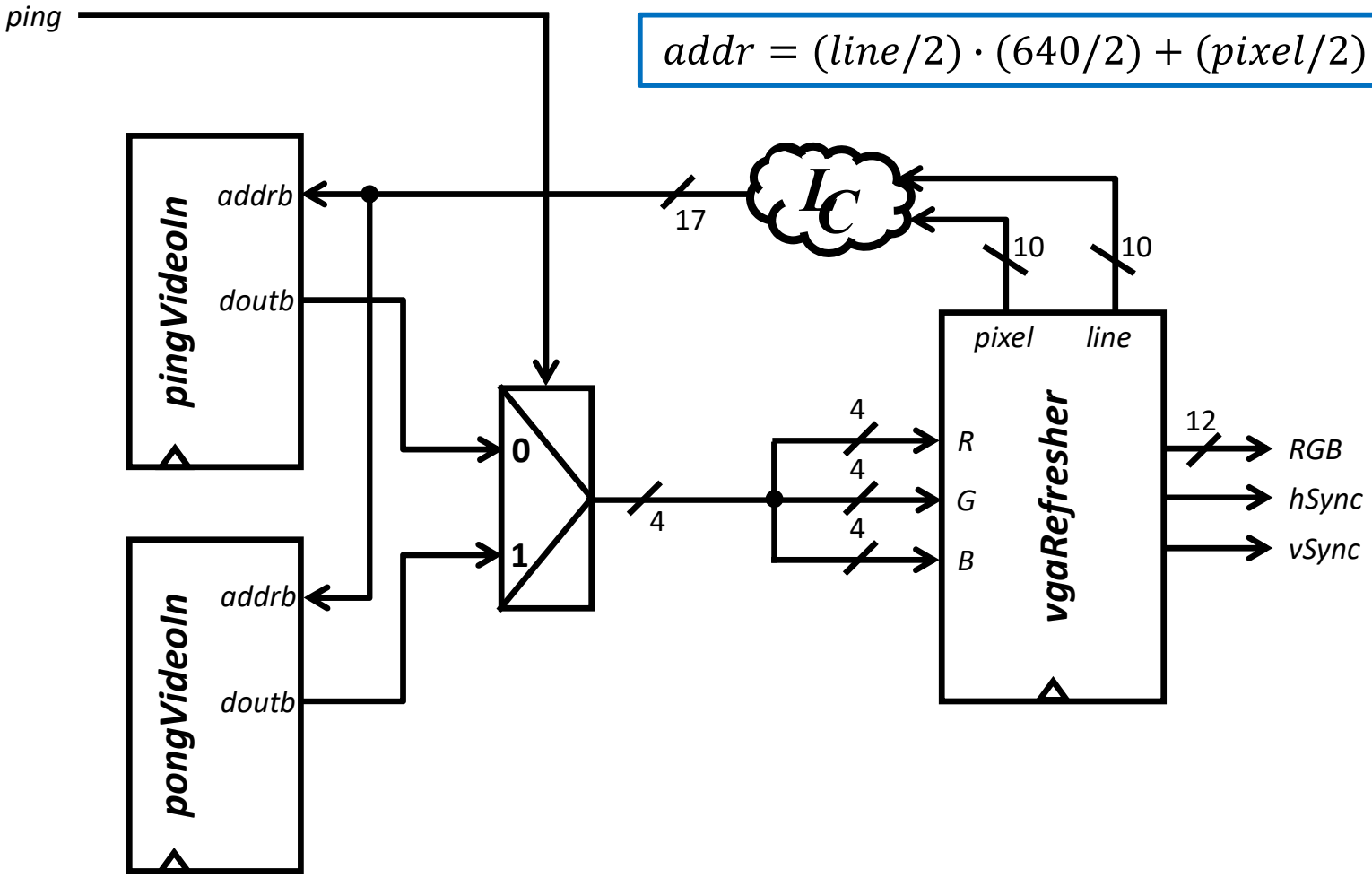


$$addr = y \cdot (640/2) + x$$

J.M. Mendías
2024

Diseño principal

etapa de visualización: esquema RTL



J.M. Mendías
2024

Video Edge Detector

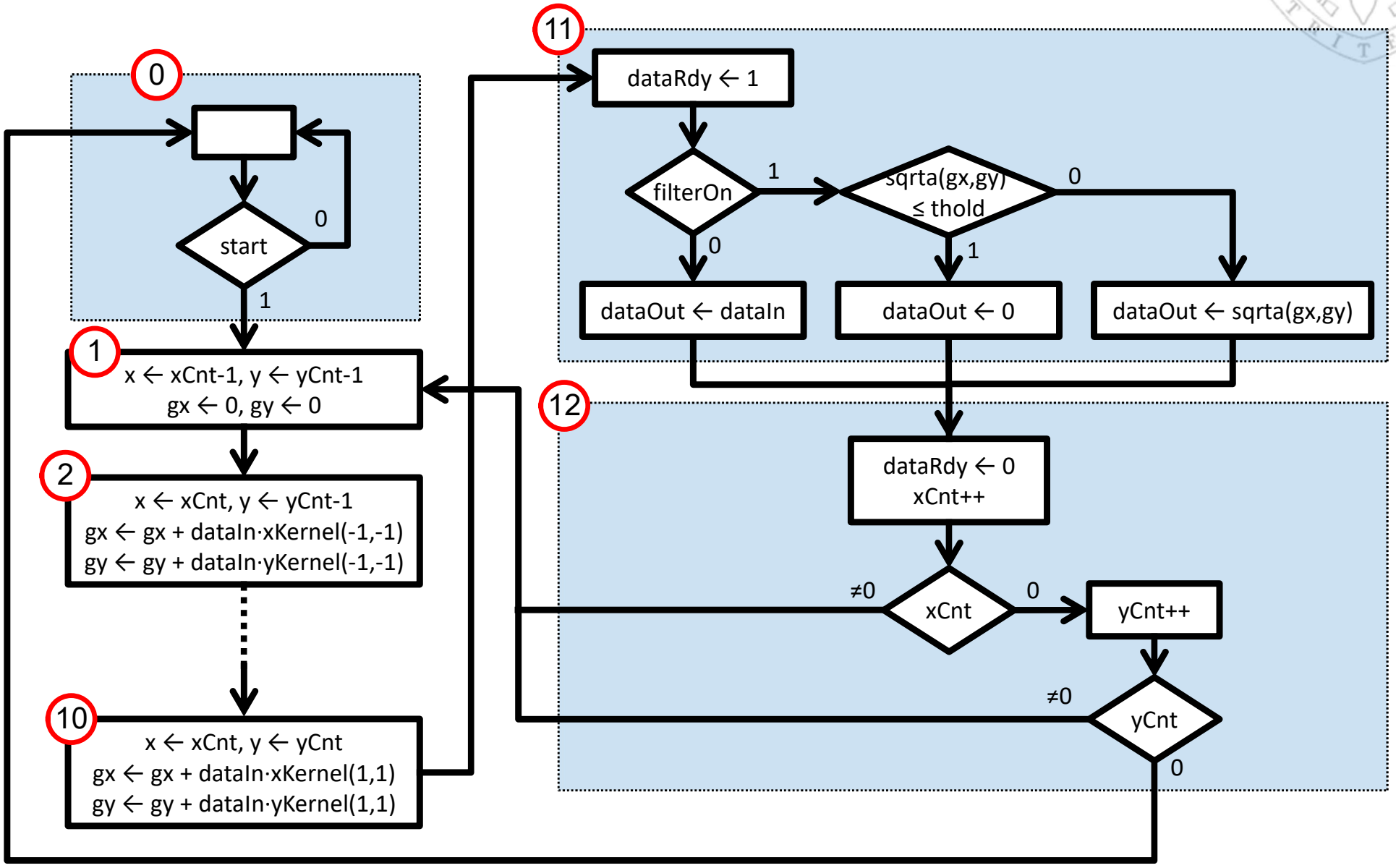
implementación multiciclo (i)

- Recorrerá todos los píxeles de la imagen calculando la convolución.
  - El procesamiento comenzará cada vez que se haya cargado una nueva imagen.
  - Se realizará pixel a pixel de arriba abajo y de izquierda a derecha.
  - Debe tener en cuenta que la lectura/escritura de memoria tiene latencia de 1 ciclo.
  - El cálculo de cada pixel de la imagen de salida requerirá 12 ciclos:
    - 1 ciclo de latencia de lectura del primer pixel de memoria.
    - 9 ciclos para acumular la multiplicación de cada pixel de la imagen original por un coeficiente del kernel.
    - 1 ciclo para realizar la raíz cuadrada y almacenar el pixel calculado.
    - 1 ciclo para avanzar los contadores que almacenan las coordenadas del pixel procesado.

- Tiempo de captura de una imagen
  - Num de píxeles VGA = 640×480 = 307200 px
  - Num. de ciclos = 307200 px × 8 ciclos/px =2457600 ciclos
- Tiempo de cómputo
  - Num de píxeles QVGA = (640/2)×(480/2) = 76800 px
  - Num. de ciclos = 12 ciclos/pixel: 76800 px × 12 ciclos/px = 921600 ciclos ✓

Video Edge Detector

implementación multiciclo (ii)



J.M. Mendías 2024

Video Edge Detector

implementación multiciclo (iii)

- La raíz cuadrada es una operación compleja.
  - Es común usar aproximaciones más simples.
  - Para aproximar la raíz cuadrada de la suma del cuadrado de 2 enteros puede usarse:

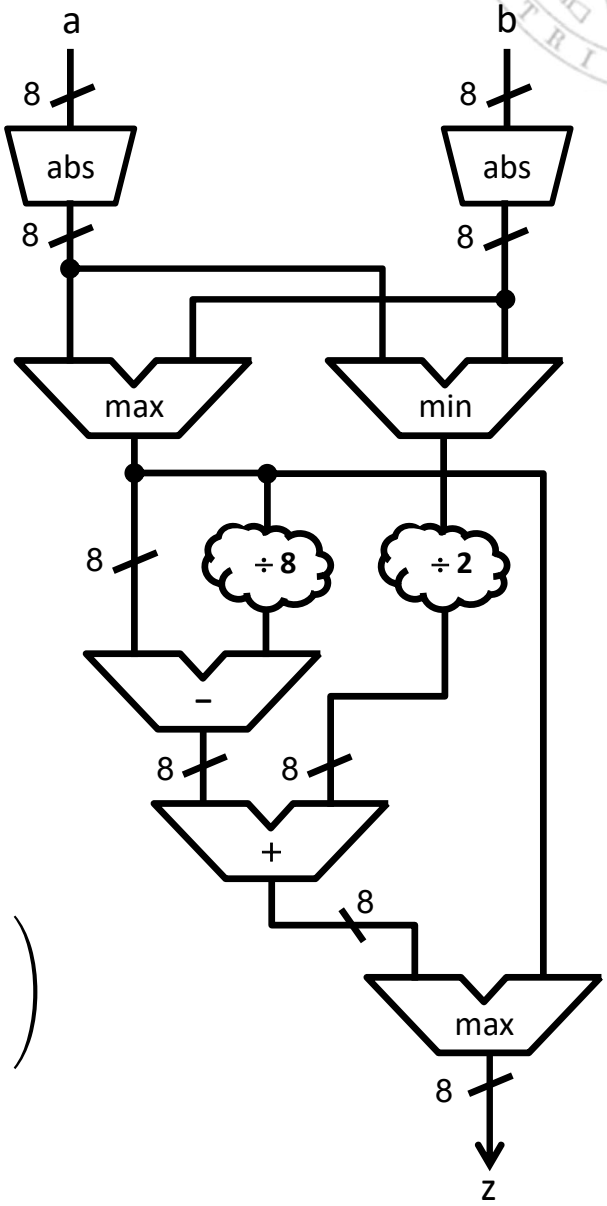
$$z = \sqrt{a^2 + b^2}$$

$$z \approx max\left(\begin{array}{c} 0.875 \cdot max(\,|a|,\,|b|\,) + 0.5 \cdot min(\,|a|,\,|b|\,), \\ max(\,|a|,\,|b|\,) \end{array}\right)$$

$$0.875 \cdot x = x - 0.125 \cdot x = x - \frac{x}{8}$$

$$0.5 \cdot x = \frac{x}{2}$$

$$z \approx max\left(\begin{array}{c} \left(max(\,|a|,\,|b|\,) - \frac{max(\,|a|,\,|b|\,)}{8}\right) + \frac{min(\,|a|,\,|b|\,)}{2}, \\ max(\,|a|,\,|b|\,) \end{array}\right)$$



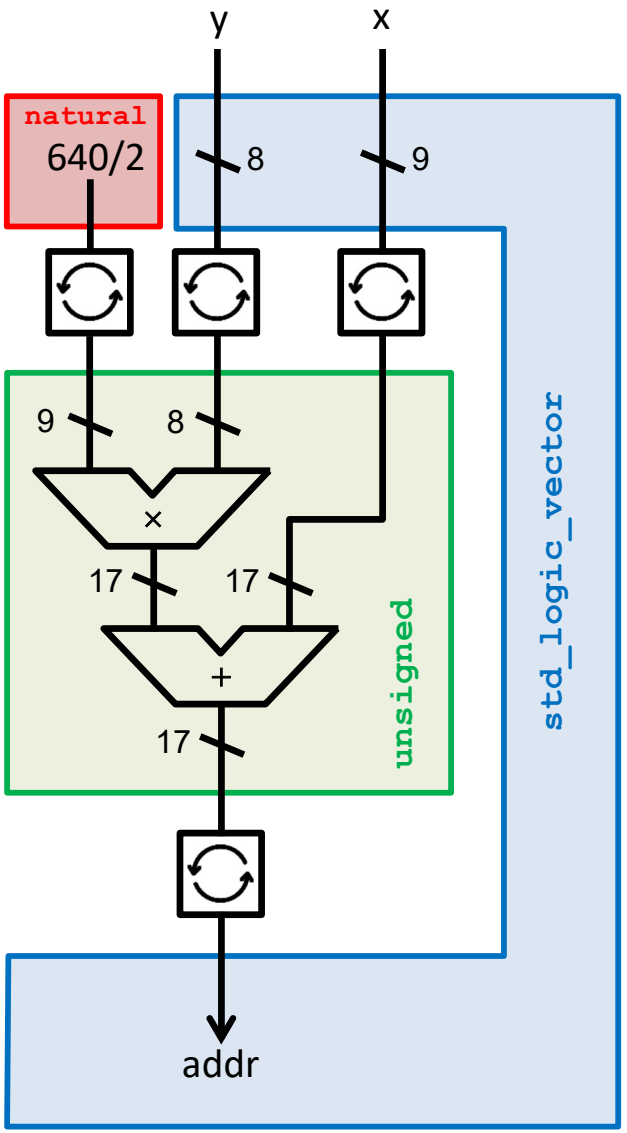
Reutilización

uso de funciones

- En VHDL los componentes permiten encapsular y reutilizar lógica.
  - Sin embargo, para lógica combinacional recurrente puede ser más cómodo usar funciones o procedimientos.

$$addr = y \cdot (640/2) + x$$

```
function xy2addr(
  x : std_logic_vector;
  y : std_logic_vector
) return std_logic_vector
is
  variable xVal    : unsigned(x'range);
  variable yVal    : unsigned(y'range);
  variable result : unsigned(16 downto 0);
begin
  xVal   := unsigned( x );
  yVal   := unsigned( y );
  result := to_unsigned( 640/2, 9 )*yVal
            + resize( xVal, 17 );
  return std_logic_vector( result );
end function;
```



J.M. Mendías
2024

Reutilización

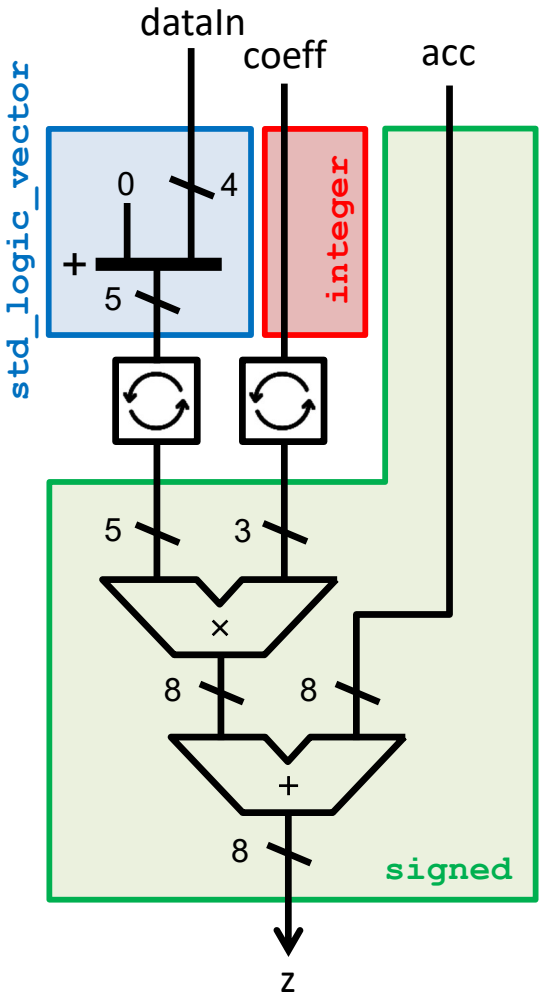
uso de procedimientos

$$gx \leftarrow gx + dataIn \cdot xKernel(1,1)$$
$$gy \leftarrow gy + dataIn \cdot yKernel(1,1)$$

```
function multAdd(
  acc     : signed;
  dataIn  : std_logic_vector;
  coeff   : integer
) return signed
is
  variable result : signed (acc'range);
begin
  result ...;
  return result;
end function;

procedure mac(
  signal   dataIn : in std_logic_vector;
  variable gx, gy : inout signed;
  constant xCoeff : integer;
  constant yCoeff : integer )
is
begin
  gx := multAdd( gx, dataIn, xCoeff );
  gy := multAdd( gy, dataIn, yCoeff );
end procedure;
```

es necesario indicar la clase del parámetro



J.M. Mendías
2024

Tareas

1. Crear el proyecto **`lab12`** en el directorio **`DAS`**
2. Descargar de la Web los ficheros en:
   - **`lab12:`** **`lab12.vhd`**, **`videoEdgeDetector.vhd`** y **`lab12.xdc`**
3. Completar el código omitido en los ficheros:
   - **`lab12.vhd`** y **`videoEdgeDetector.vhd`**
4. Añadir al proyecto los ficheros:
   - **`common.vhd`**, **`synchronizer.vhd`**, **`edgeDetector.vhd`**, **`ov7670programmer.vhd`**, **`ov7670reader.vhd`**, **`rgb2grey.vhd`**, **`freqSynthesizer.vhd`**, **`vgaRefresher.vhd`**, **`videoEdgeDetector.vhd`**, **`lab12.vhd`** y **`lab12.xdc`**
5. Configurar y añadir al proyecto los IPs:
   - **`frameBuffer`**
6. Sintetizar, implementar y generar el fichero de configuración.
7. Conectar la cámara y el monitor a la placa y encenderla.
8. Descargar el fichero **`lab12.bit`**

Acerca de *Creative Commons*

- Licencia CC (Creative Commons)
  - Ofrece algunos derechos a terceras personas bajo ciertas condiciones. Este documento tiene establecidas las siguientes:

Reconocimiento (*Attribution*):
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.

No comercial (*Non commercial*):
La explotación de la obra queda limitada a usos no comerciales.

Compartir igual (*Share alike*):
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Más información: https://creativecommons.org/licenses/by-nc-sa/4.0/