

VHDL para procesamiento de señal

Hipólito Guzmán Miranda
Profesor Contratado Doctor
Universidad de Sevilla
hguzman@us.es

VHDL para procesamiento de señal

- Integer
- Boolean
- Sobre el punto flotante
- Punto fijo
- Tipos complejos
- Planteando interfaces
- Filtros y etapas
- FIFOs y memorias

VHDL para procesamiento de señal

- Integer
- Boolean
- Sobre el punto flotante
- Punto fijo
- Tipos complejos
- Planteando interfaces
- Filtros y etapas
- FIFOs y memorias

¿Para qué se usan?

Para índices que sirvan para acceder a arrays/agregados:

```
signal i : integer range 0 to 7;
```

```
signal vect : std_logic_vector (0 to 7);
```

```
vect(i) <= '1';
```

```
std_logic_vector ( integer ) = std_logic
```



Definid el rango!

```
signal i : integer;           -- 32 bits
signal j : integer range 0 to 255;  -- 8 bits
signal k : integer range -128 to 127; -- 8 bits
```

Puede dar “simulation mismatch” si os salís del rango:

- en simulación se comportará como si tuviera 32 bits (al menos en Xilinx ISIM)
- en implementación no!

Ejemplo

```
signal cont : integer range 0 to 7;
```

```
begin
```

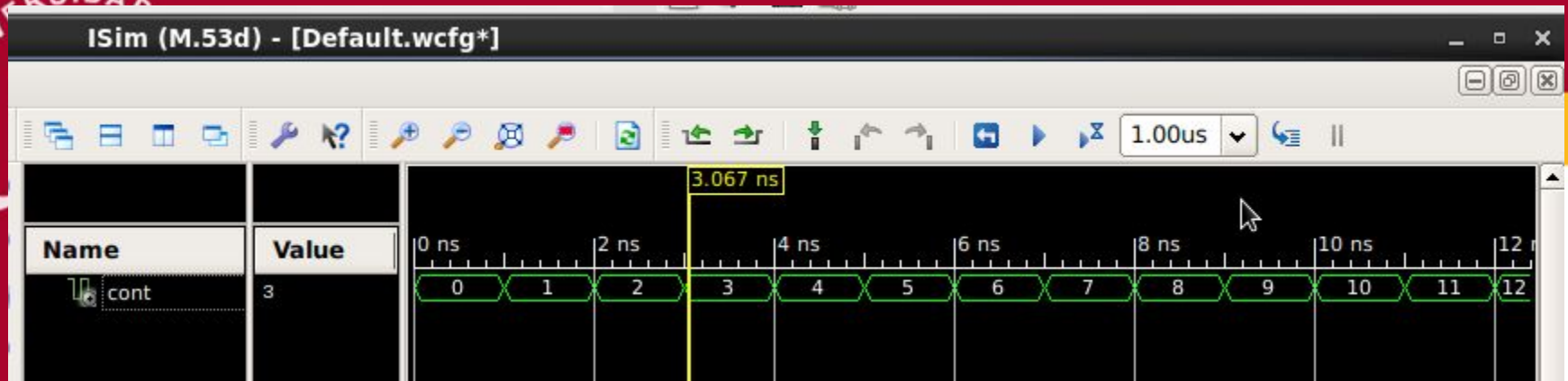
```
process
```

```
begin
```

```
    wait for 1 ns;
```

```
    cont <= cont +1;
```

```
end process;
```



```
begin
```

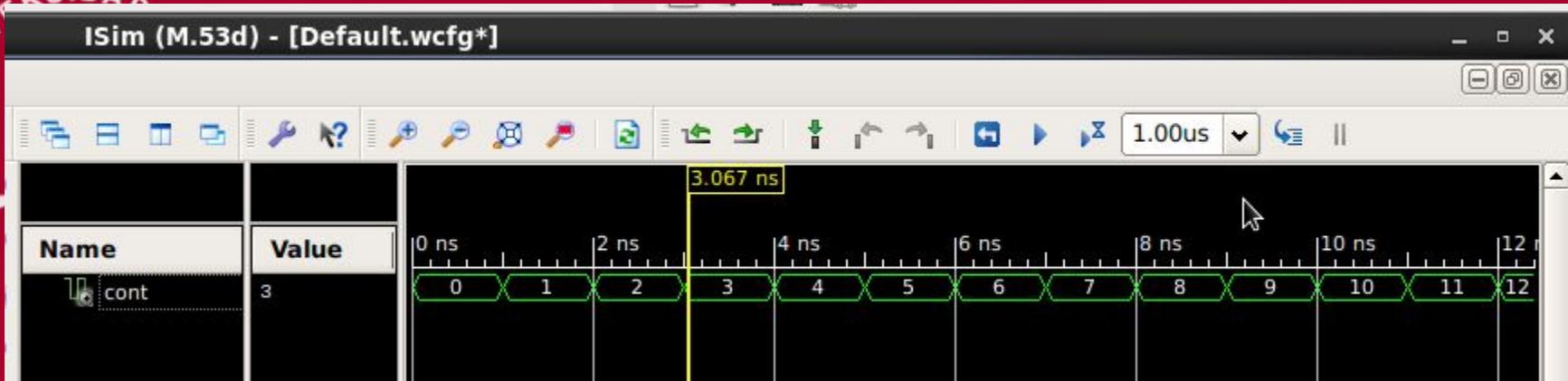
```
process
```

```
begin
```

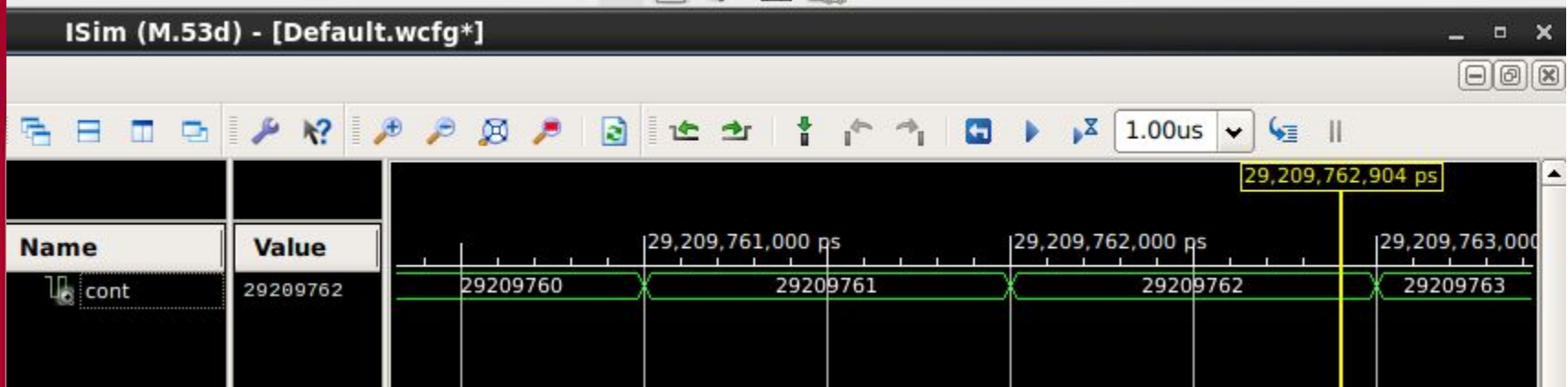
```
    wait for 1 ns;
```

```
    cont <= cont +1;
```

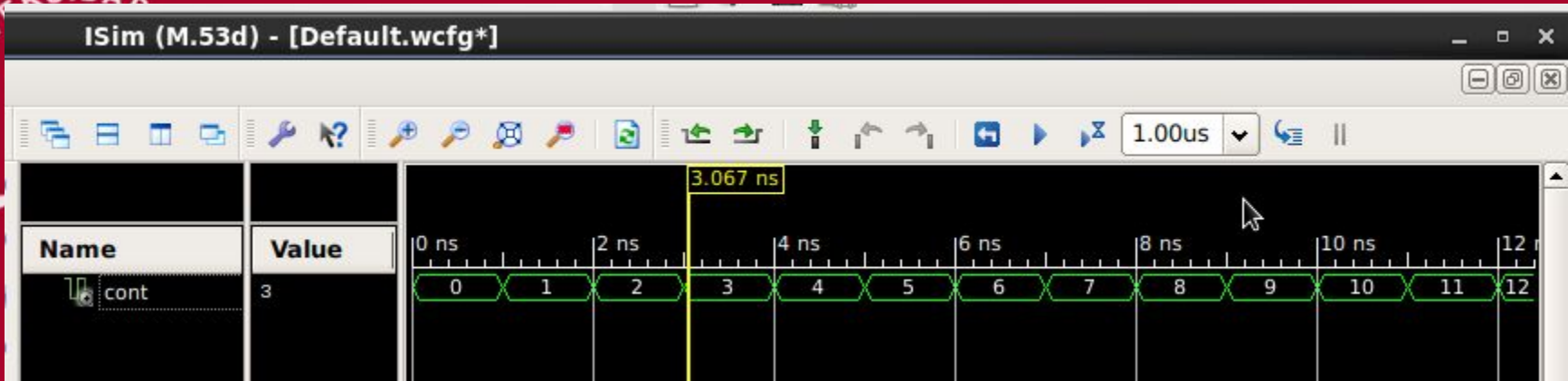
```
end process;
```



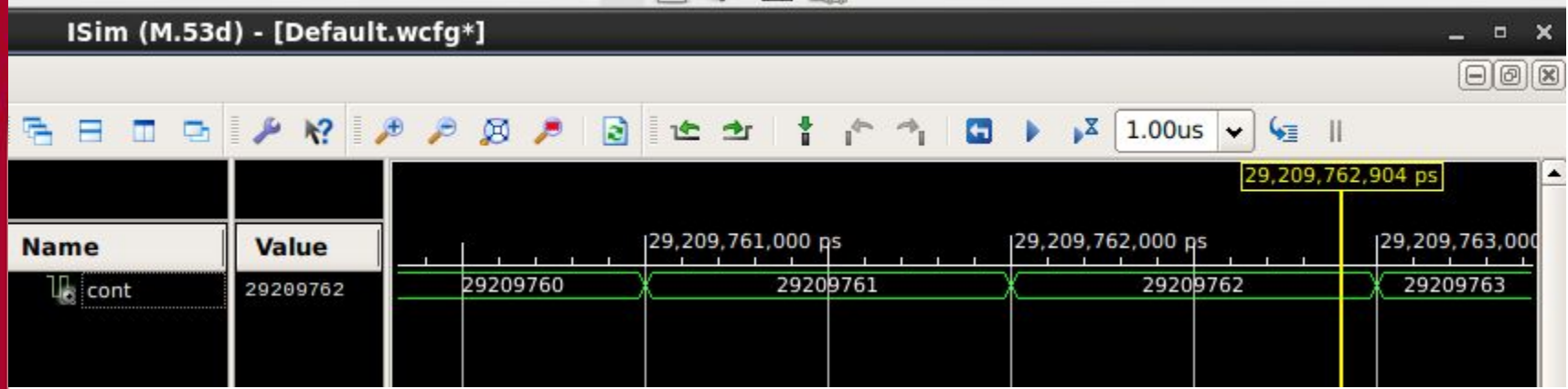
Signal cont : Integer Range (0 to 12)



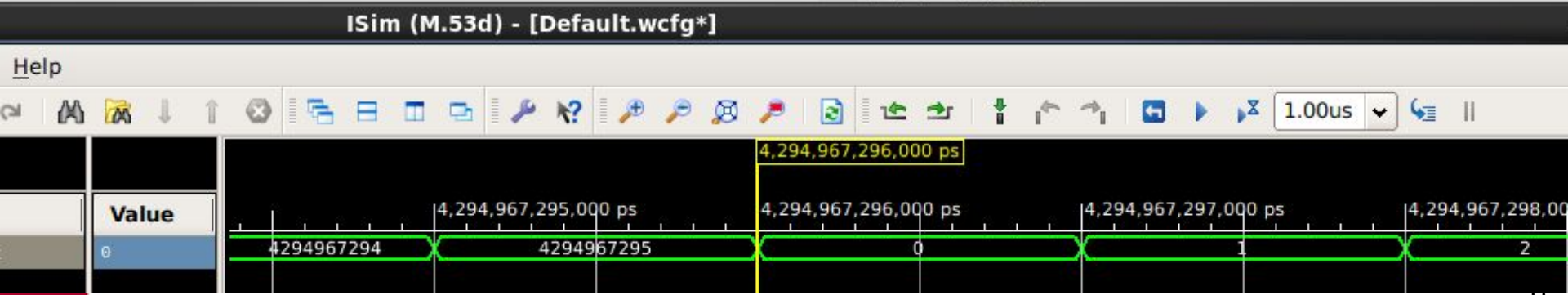
```
begin  
    wait for 1 ns;  
    cont <= cont +1;  
end process;
```

Signal cont: Integer Range (0 to 12)



hexin



VHDL para procesamiento de señal

- Integer
- Boolean
- Sobre el punto flotante
- Punto fijo
- Tipos complejos
- Planteando interfaces
- Filtros y etapas
- FIFOs y memorias

Realmente, es un tipo enumerado

-- predefinido en VHDL:

```
type Boolean is (false, true);
```

Puede ser interesante en GENERICS, para parametrizar componentes

Para señales binarias de I/O, usad `std_logic`

Ejemplo

```
if (my_std_logic = '1') then
```

VS

```
if ( condition ) then
```

La expresión dentro del if ha de ser un boolean

VHDL para procesamiento de señal

- Integer
- Boolean
- Sobre el punto flotante
- Punto fijo
- Tipos complejos
- Planteando interfaces
- Filtros y etapas
- FIFOs y memorias

Sólo cuando realmente se necesite

El float es un tradeoff entre precisión y rango

(es el 'google maps' de los números :)

Tiene sus limitaciones:

- Pérdida de precisión
- Ocupación de recursos
- Operaciones más lentas
- Necesidad de convertir desde/a otros tipos

Pérdida de precisión

- ¿Qué pasa si sumamos mil números flotantes?

Veamos un ejemplo (en C)

```
#include <stdio.h> // for printf
#include <time.h> // for time
#include <stdlib.h> // for rand

#define MAX 10e10 // max rand number range

int main (void)
{
float table [1000];
float accfwd = 0;
float accrev = 0;

srand(time(NULL)); // set random seed to time

int i;

// Initialize table with random floats
for (i=0; i<1000; i++)
{
table[i] = ((float)rand()/((float)(RAND_MAX))) * MAX;
}
```



```
// Sum from 0 to 999
for (i=0; i<1000; i++)
{
    accfwd += table[i];
}

// Sum from 999 to 0
for (i=999; i>= 0; i--)
{
    accrev += table[i];
}

// Compare and print
printf ("sum (fwd): %f\n", accfwd);
printf ("sum (rev): %f\n", accrev);
printf ("difference: %f\n", accfwd-accrev);

return 1;
}
```

Pérdida de precisión

Al ejecutar el programa:

```
$ ./a.out
```

```
sum (fwd): 50039409868800.000000
```

```
sum (rev): 50039435034624.000000
```

```
difference: -25165824.000000
```

Pérdida de precisión

Al ejecutar el programa:

```
$ ./a.out
```

```
sum (fwd): 50039409868800.000000
```

```
sum (rev): 50039435034624.000000
```

```
difference: -25165824.000000
```

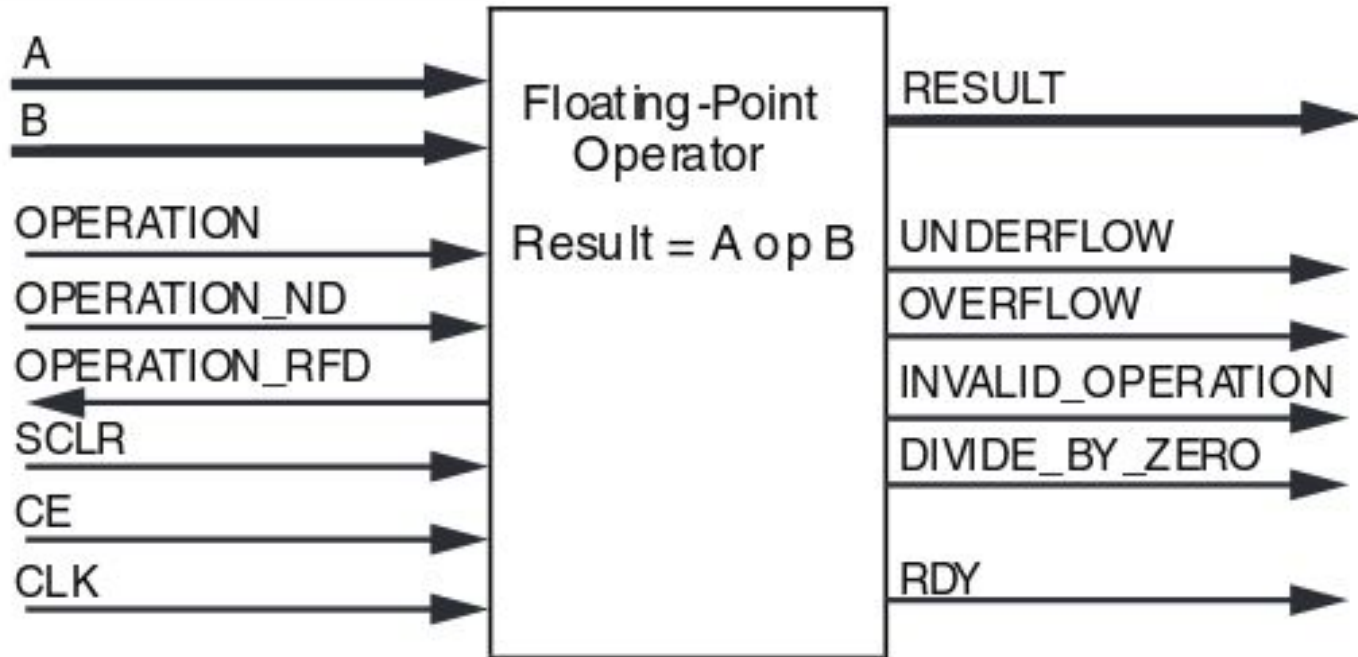
¿De verdad lo necesitas?

Con 32 bits (fixed) tienes hasta $4\text{G} \sim 4 \cdot 10^9$
¡nueve órdenes de magnitud!

Con 64 (fixed) bits tienes hasta $1.89 \cdot 10^{19}$!

En aplicaciones típicas, con eso suele ser
más que suficiente

Xilinx floating-point operator:



DS335_01_021508

Figure 1: Block Diagram of Generic Floating-Point Binary Operator Core

Fuente: Xilinx (DS335)

VHDL para procesado de señal

- Integer
- Boolean
- Sobre el punto flotante
- Punto fijo
- Tipos complejos
- Planteando interfaces
- Filtros y etapas
- FIFOs y memorias

Varias formas:

Usando `std_logic_vector` + librerías
`ieee.std_logic_[un]signed`

Usando librería `numeric_std` + tipos `signed`
y `unsigned`

Usando librerías específicas (por ejemplo: el
`package ieee_proposed.fixed_pkg`)

No soportadas por todos los fabricantes!

Varias formas:

Usando `std_logic_vector` + librerías
`ieee.std_logic_[un]signed`

Usando librería `numeric_std` + tipos `signed`
y `unsigned`

Usando librerías específicas (por ejemplo: el
package `ieee_proposed.fixed_pkg`)

No soportadas por todos los fabricantes!

Al final es lo mismo!

Vector de bits \rightarrow nosotros decidimos qué interpretamos como parte entera y qué interpretamos como parte decimal

La lógica para sumar o multiplicar es la misma independientemente de dónde interpretas que está la coma!

Ejemplo (suma)

	4,0	4,1	4,2
0111	7	3.5	1.75
+ 0001	1	0.5	0.25
= 1000	8	4.0	2.00

Notación: $(x, y) = (\text{bits_totales}, \text{bits_parte_decimal})$

Mismos decimales en el resultado que en los operandos

Ejemplo (mult)

	4,0	4,1	4,2
	7	3.5	1.75
*	2	1	0.5
=	14	3.5	0.8750
	8,0	8,2	8,4

$$\text{ent}(\text{res}) = \text{ent}(a) + \text{ent}(b)$$

$$\text{dec}(\text{res}) = \text{dec}(a) + \text{dec}(b)$$

Ejercicio (mult)

	“total, dec”	valor
	4,2	
*	4,1	
=		

¿Dónde tiene el punto el resultado?

¿Con qué bits nos quedamos?

- Cuando realizamos una operación en punto fijo podemos obtener más bits que los operandos
- Si nos descuidamos terminamos sacando 58 bits a la salida
- Hay que estudiar los rangos de los números con que estamos operando
- Y decidir en función de la precisión que necesitemos

Ejercicio:

- a: entero entre 0 y 100
- b: entero entre -5 y 40

¿Cuántos bits necesitamos para $a * b$?

Suma con acarreo

Si a y b son de 8 bits, para hacer

$c \leftarrow a + b$;

c tiene que ser de 8 bits.

Para tener el bit de acarreo (si nos hace falta):

$d \leftarrow \text{resize}(a,9) + \text{resize}(b,9)$;

d es vector de 9 bits

VHDL para procesamiento de señal

- Integer
- Boolean
- Sobre el punto flotante
- Punto fijo
- Tipos complejos
- Planteando interfaces
- Filtros y etapas
- FIFOs y memorias

Parte real + parte imaginaria

Existe una librería que define un tipo complejo, pero que no es sintetizable: tendréis que crearos los vuestros.

Por ejemplo:

```
type complex10 is record
```

```
  re : std_logic_vector (9 downto 0);
```

```
  im : std_logic_vector (9 downto 0);
```

```
end record;
```

VHDL para procesamiento de señal

- Integer
- Boolean
- Sobre el punto flotante
- Punto fijo
- Tipos complejos
- Planteando interfaces
- Filtros y etapas
- FIFOs y memorias

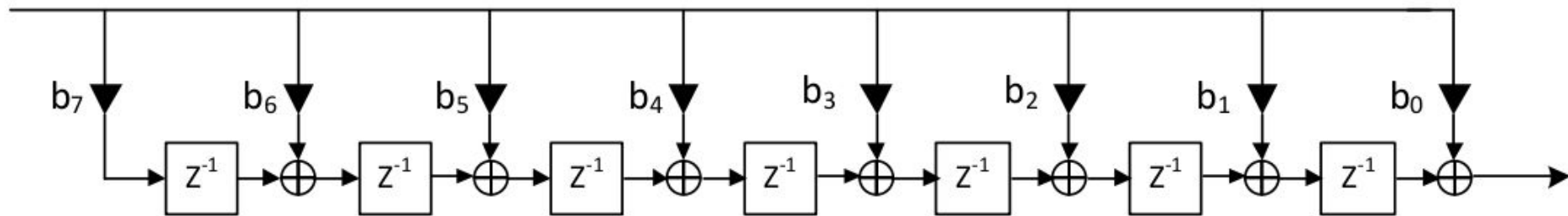
¿Cómo comunicamos nuestros bloques?

- No siempre (en cada ciclo de clk) las entradas tendrán un nuevo dato
- El uso de una señal “data_valid” está altamente recomendado
- Otros interfaces: FIFOs, memorias

VHDL para procesamiento de señal

- Integer
- Boolean
- Sobre el punto flotante
- Punto fijo
- Tipos complejos
- Planteando interfaces
- Filtros y etapas
- FIFOs y memorias

Filtros como conjuntos de etapas (o 'taps')



- 1 etapa es una multiplicación, una suma, un retraso
- Retraso 1 muestra != retraso 1 ciclo
- Si usamos "data_valid" es fácil de arreglar

VHDL para procesamiento de señal

- Integer
- Boolean
- Sobre el punto flotante
- Punto fijo
- Tipos complejos
- Planteando interfaces
- Filtros y etapas
- FIFOs y memorias

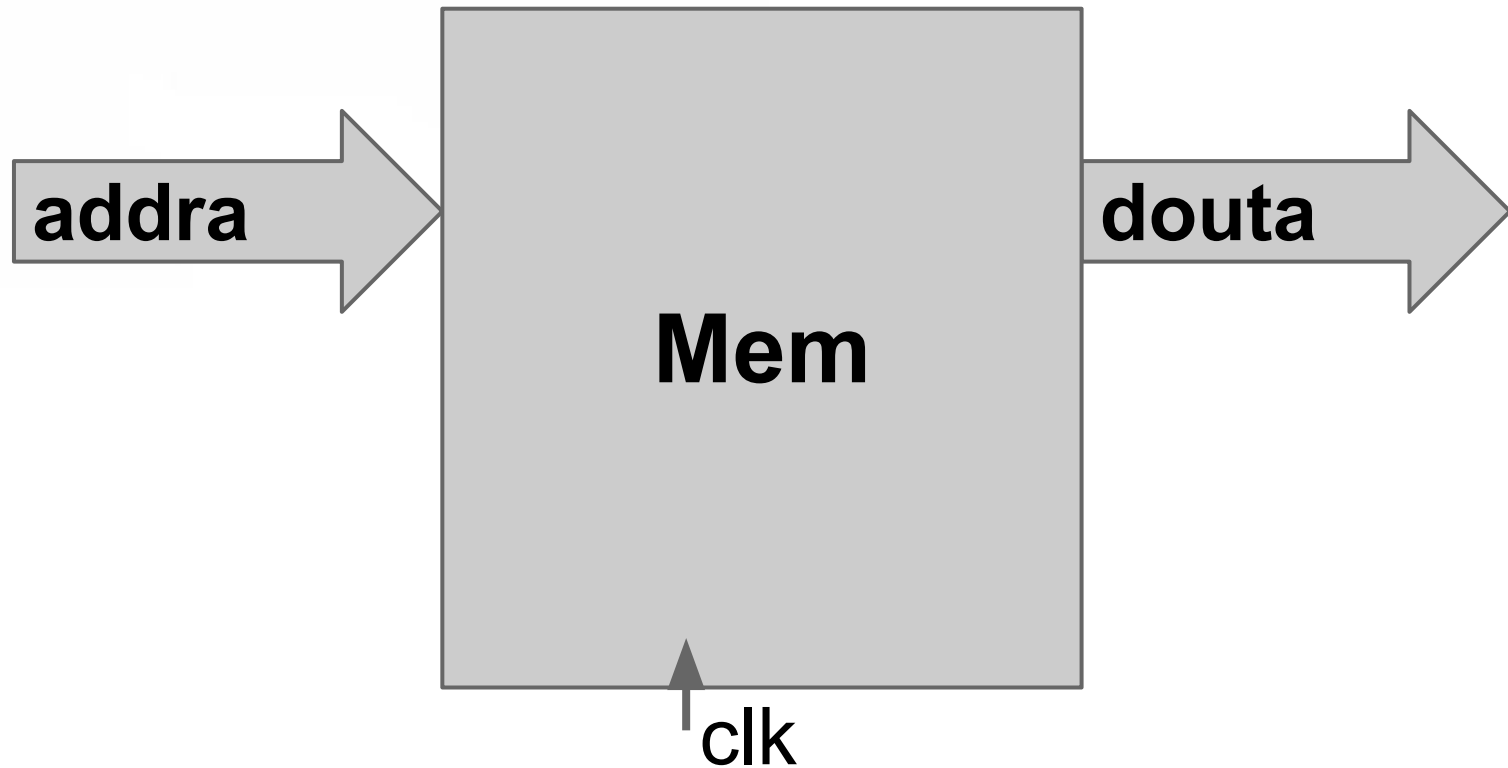
Se generan con Xilinx Core Generator

Utilizan BRAMs (Block RAMs) internas a la
FPGA (no biestables)

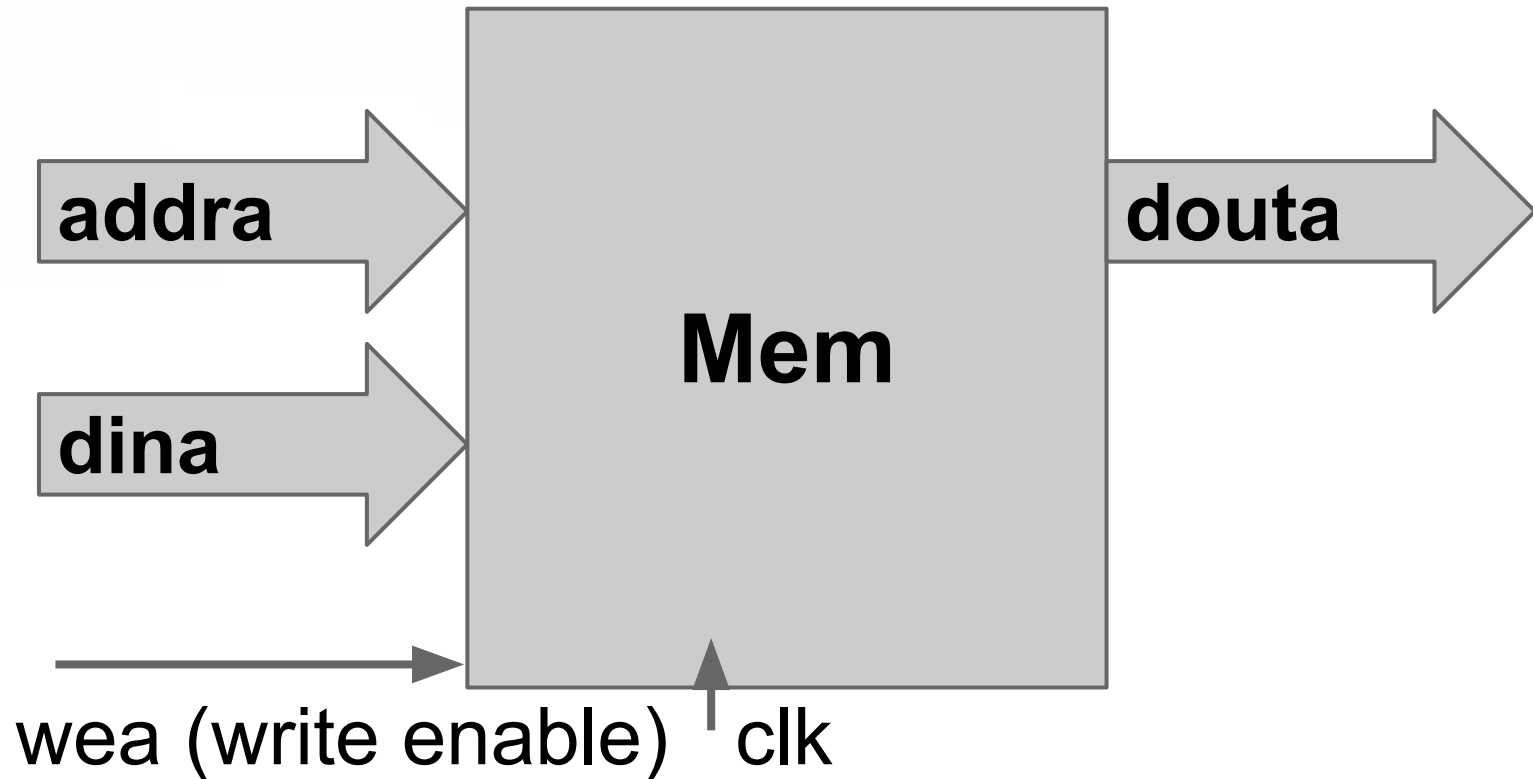
Síncronas (entrada de clk)

¿Habéis hecho alguna?

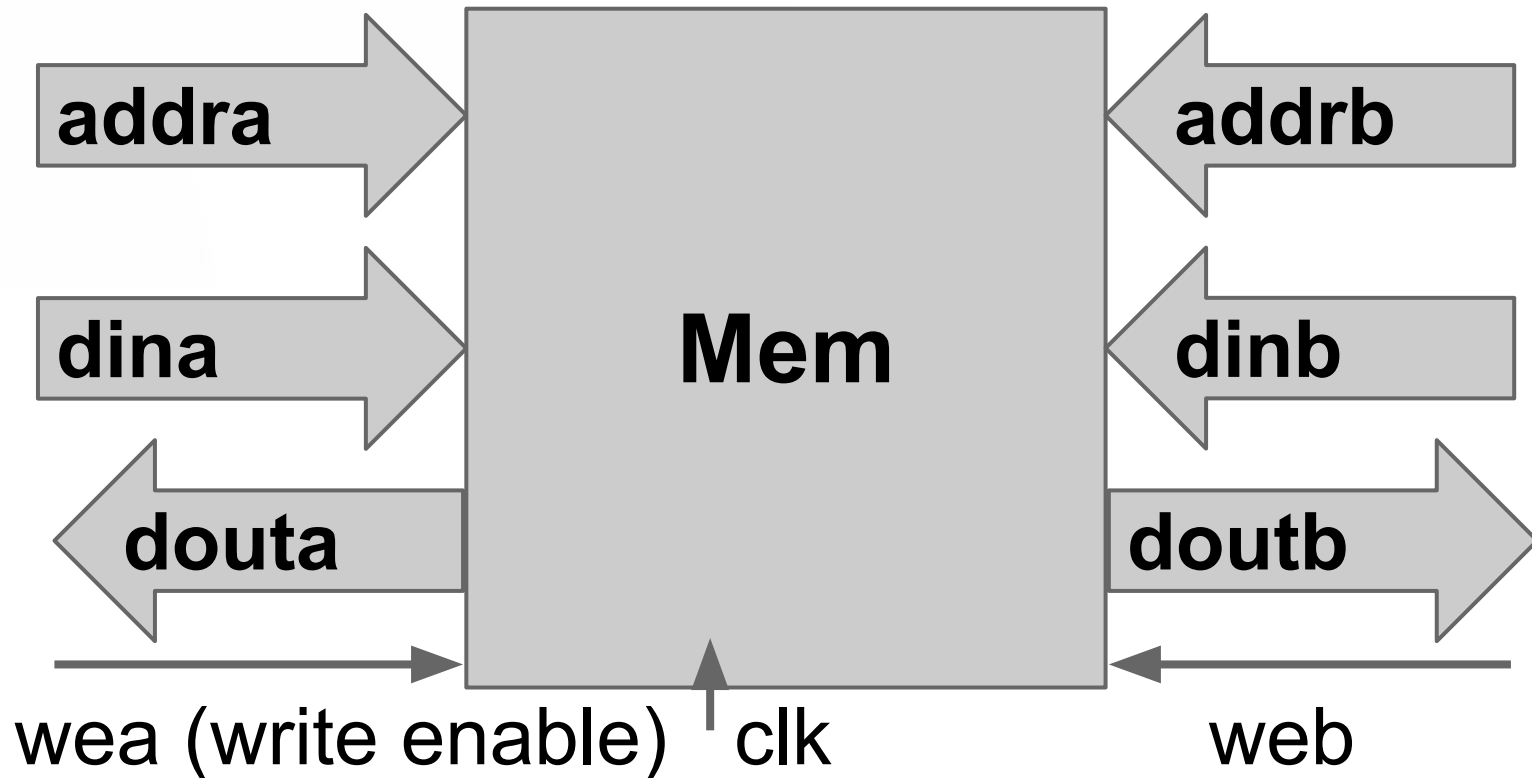
Single port ROM



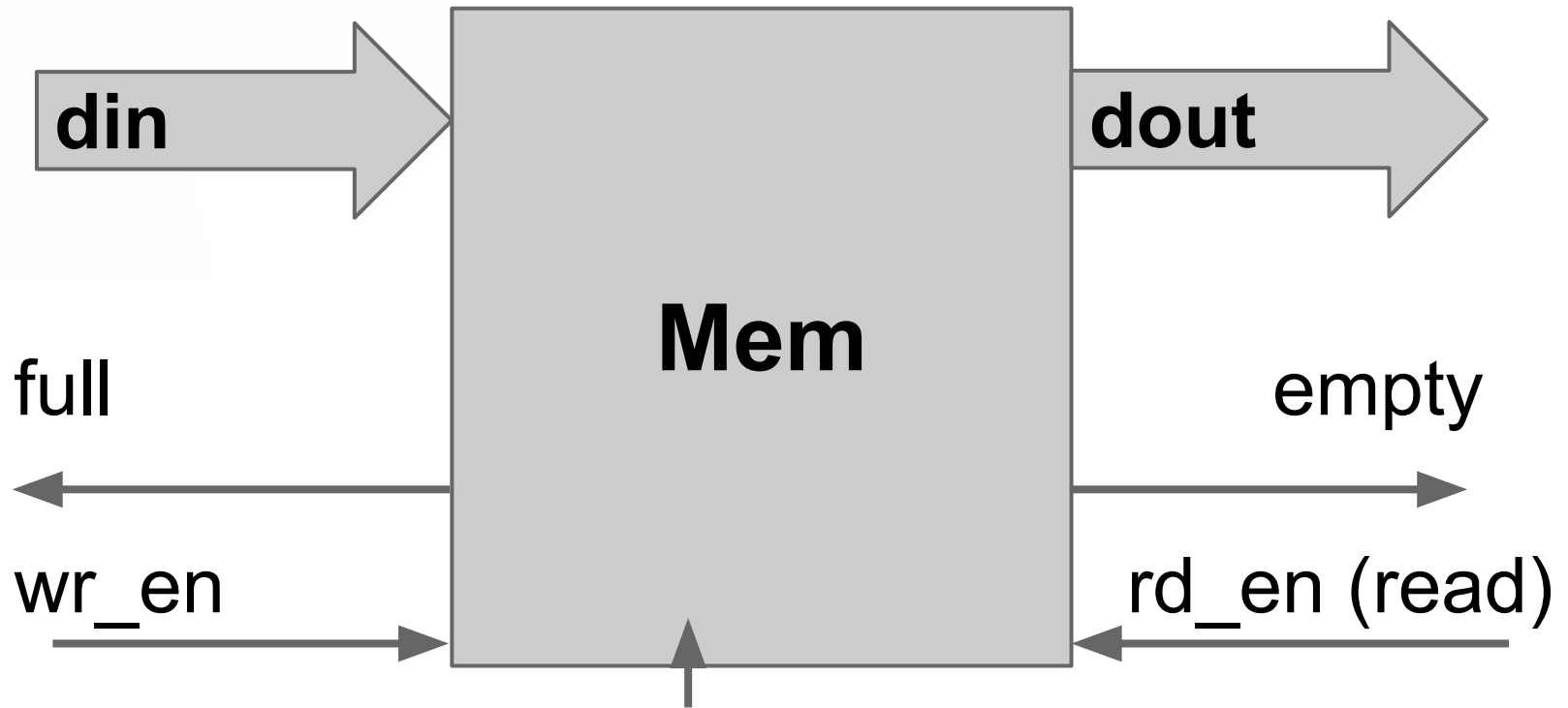
Single port RAM



Dual port RAM



FIFOs



Un consejo final

Simulad cada concepto que no tengáis claro por separado.

Siempre será mucho más sencillo que depurar un circuito complejo cuando lo tengáis hecho.