

Sistemas Electrónicos Digitales

Tema #4 Diseño mediante Lenguajes de Descripción Hardware

Parte 4.7


Índice

- 4.1 Ventajas de los HDL.
- 4.2 Metodología de Diseño.
- 4.3 VHDL. Sintaxis de VHDL.
- 4.4 Codificación de circuitos lógicos en VHDL.
- 4.5 Módulos IP.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70


ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

 UNIVERSIDAD
POLITÉCNICA
DE MADRID

Problemática actual (I)

- La aparición de los sistemas empotrados, los SoC y el codiseño HW-SW supone un desafío en todas las etapas de diseño.
- Es necesario decidir cómo se especifica a nivel de sistema, cómo se hace la partición HW-SW y cómo se verifica el diseño.
- En los diseños de sistemas empotrados actuales el SW se desarrolla en ensamblador o C/C++ y el HW en VHDL o Verilog.
- Esto genera problemas al usarse distintos lenguajes de programación, herramientas incompatibles y flujos de trabajo fragmentados.

 UNIVERSIDAD
POLITÉCNICA
DE MADRID

Problemática actual (II)

- Para enfrentar esta problemática se han desarrollado los denominados “Lenguajes de Descripción a Nivel de Sistema”.
- Permiten:
 - Trabajar a distintos niveles de abstracción sobre el hardware subyacente.
 - Simplificar el particionado hardware/software describiendo ambos en un mismo lenguaje (típicamente basado en C).
 - Mejorar la velocidad de simulación y la capacidad de

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99



Estado actual herramientas

- Las herramientas actuales permiten utilizar estos lenguajes para modelar y simular (incluida la cosimulación HW/SW), pero no para la síntesis directa de un sistema.
- Lo que sí existen son herramientas para generar una descripción HDL (VHDL, Verilog), bajo ciertas restricciones, a partir de ellos.
- De esta forma es posible continuar el proceso de síntesis hasta completarlo con las herramientas de síntesis actuales.



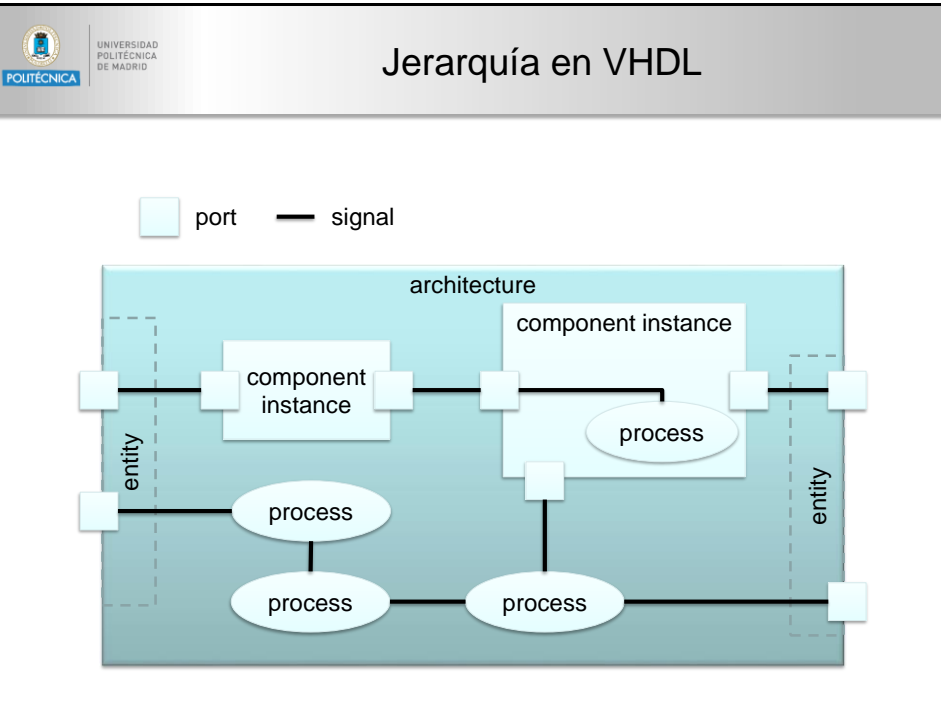
Aportación SystemC

- SystemC es una herramienta de modelado para sistemas empuotrados basada en C++.
- Está pensada para el diseño a nivel de sistema.
- Permite modelar la funcionalidad, la comunicación entre tareas, el SW y el HW a varios niveles de abstracción.
- Responde a la demanda de crear especificaciones ejecutables muy rápidas para validar sistemas empuotrados.
- C/C++ proporciona niveles adecuados de abstracción, integración HW-SW y prestaciones y un leguaje común para el modelado y la interoperabilidad de herramientas de diseño, servicios y módulos IP

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99



UNIVERSIDAD POLITÉCNICA DE MADRID

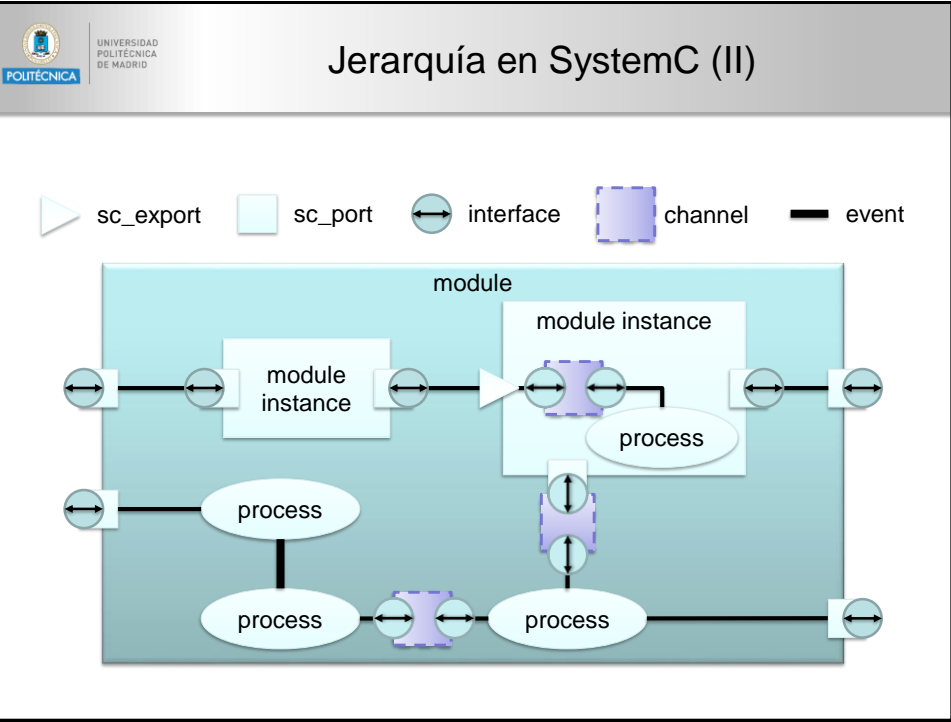
Jerarquía en SystemC (I)

- La jerarquía en SystemC es semejante a la de VHDL.
- Se define instanciando módulos dentro de otros módulos.
- Como en VHDL, los procesos se comunican a través de señales.
- Sin embargo, SystemC posee capacidades de modelado adicional como canales (*channels*), interfaces (*interfaces*) y eventos (*events*).

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99



UNIVERSIDAD POLITÉCNICA DE MADRID

Ejemplo: Biestable D

```

library ieee;
use ieee.std_logic_1164.all;
entity dff is
  port(clk : in std_logic;
        din : in std_logic;
        dout: out std_logic);
end dff;
architecture behavioral of dff is
begin
  p1:process(clk)

```

```

#include <systemc.h>

SC_MODULE(dff) {
  sc_in<sc_logic> clk, din;
  sc_out<sc_logic > dout;

  void p1() {
    if (clk.posedge())
      dout = din;
  }
}

```



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

UNIVERSIDAD POLITÉCNICA DE MADRID

Módulos

- **SC_MODULE**(name) desempeña el papel del par **entity-architecture** en VHDL.
- Contiene los puertos de E/S, funciones, componentes, procesos...
- Cada módulo suele ir en un fichero propio con la extensión '.h'.
- En diseños más complejos la definición de las funciones puede hacerse en un fichero .cpp
- Dado que se trata de clases es necesario un constructor para el que se normalmente se usa el macro **SC_CTOR**(nombre) que no tiene equivalente en VHDL.
- Es en este constructor donde se crean los procesos y se establece la lista de sensibilidad de los mismos.

UNIVERSIDAD POLITÉCNICA DE MADRID

Puertos: clases

Puerto VHDL	Puerto SystemC
in	sc_in
out	sc_out
inout	sc_inout
buffer	sc_inout

- **sc_out** se comporta como *buffer* en VHDL.
- Están basados en plantillas de C++.
- El tipo del puerto se

- Ejemplos:

```
sc_in<sc_logic> reset;
```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70

Cartagena99

VHDL	SystemC	Comentarios
reg <= X"55";	reg = 0x55;	Estilo desaconsejado
reg <= "0101XZ01";	reg.write("0101XZ01");	
q <= '0';	q.write('0');	
q <= '0';	q.write(SC_LOGIC_0)	
clk = '1'	clk == '1'	Estilo desaconsejado
if (clk = '1') then	if (clk.read() == SC_LOGIC_1) {	
dout <= din;	dout = din;	Estilo desaconsejado
dout <= din;	dout.write(din.read());	

VHDL	SystemC
<pre>entity example is port (d: in bit_vector(7 downto 0) := X"8E"; carry: out std_logic := '1'); end;</pre>	<pre>SC_MODULE(example) { sc_in<sc_lv<8>> > d; sc_out<sc_logic > carry; ... SC_CTOR(example) { carry.initialize(0x8E); zero.initialize('1'); ... } };</pre>

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70

Cartagena99

Procesos

- La funcionalidad se describe por medio de procesos.
- Existen 3 tipos:
 - SC_METHOD
 - SC_THREAD
 - SC_CTHREAD
- Los procesos se crean durante la etapa de enlazado o, en terminología de simulación, durante la elaboración (creación estática).
- Los procesos se pueden crear también dinámicamente durante la simulación.
- En simulación, SC_METHOD es más rápido que SC_THREAD.

Procesos: lista de sensibilidad


- Lista de las señales, puertos y eventos que activan el proceso.
- Se pueden usar varios estilos:

```
sensitive << clk; // clk'event (ambos flancos)
sensitive << clk.pos(); // rising_edge(clk)
sensitive << clk.neg(); // falling_edge(clk)
sensitive(clk); // Sólo 1 arg. (desaconsejado)
sensitive << clk << reset; // Varios argumentos
```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70


ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

 UNIVERSIDAD
POLITÉCNICA
DE MADRID

Procesos: SC_METHOD

- Se ejecuta siempre que ocurre un evento en la lista de sensibilidad.
- El proceso se ejecuta por completo antes de devolver el control al planificador de SystemC (el proceso queda suspendido hasta el próximo evento).
- Equivale a un **process** de VHDL con lista de sensibilidad estática.
- Por ello tampoco puede contener **wait()**.

 UNIVERSIDAD
POLITÉCNICA
DE MADRID

Procesos: SC_THREAD

- El planificador lo ejecuta una sola vez.
- El proceso puede suspenderse con una sentencia **wait()**, devolviendo el control al planificador.
- El proceso continúa a la llegada de un nuevo evento que puede especificarse en la sentencia **wait()** (sensibilidad dinámica), la lista de sensibilidad (sensibilidad estática) o ambas.
- Esto último no es posible en VHDL.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Procesos SC_CTHREAD

- SC_CTHREAD: Clocked Thread.
- Parecido a SC_THREAD, pero con diferencias importantes:
 - Es necesario especificar un objeto reloj: al declarar este tipo de proceso es necesario proporcionar, además del nombre de la función, el reloj que activa el proceso.
 - SC_CTHREAD no tiene una lista de sensibilidad como los otros tipos de procesos, se activa en el flanco del reloj especificado.
- Menos flexible y mucho menos usado que los otros tipos.

Procesos: ejemplos

```
#include "systemc.h"
```

```
SC_MODULE(dff) {  
    sc_in<bool> clk;  
    sc_in<bool> din;  
    sc_out<bool> dout;  
};
```

```
void p1() {  
    dout.write(din.read());  
}
```

```
#include "systemc.h"
```

```
SC_MODULE(dff) {  
    sc_in<bool> clk;  
    sc_in<bool> din;  
    sc_out<bool> dout;  
};
```

```
void p1() {  
    while (true) {  
        wait();  
        dout.write(din.read());  
    }  
}
```

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

Procesos: ejemplos

```
#include "systemc.h"

SC_MODULE (dff) {
    sc_in_clk clk;
    sc_in<bool > din;
    sc_out<bool > dout;

    void p1 () {
        while (true) {
            wait();
            dout.write(din.read());
        }
    }

    SC_CTOR(first_counter) {
        SC_CTHREAD(p1, clk.pos());
    }
};
```

Procesos: inicialización

- En VHDL un proceso sólo se ejecuta cuando se produce un evento en su lista de sensibilidad.
- En SystemC, todos los procesos se ejecutan una vez al inicio de la simulación aunque no se haya producido evento en la lista de sensibilidad.
- Esta primera llamada puede utilizarse para la inicialización del proceso.
- Si resulta inconveniente, puede evitarse esta primera

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

Procesos: lugar atributos

- Los atributos del proceso como **sensitive** y **dont_initialize** sólo afectan al proceso precedente.
- Se recomienda indentar para resaltar el proceso al que afectan:

```

SC_CTOR(dff) {
    SC_THREAD(p1);
    dont_initialize();
    sensitive << clk.pos();
    SC_THREAD(p2);
    SC_METHOD(p3);
    sensitive << clk.pos() << reset;
}
    
```

} p1

} p3

Procesos: variables

- Equivaldrían a las usadas en los **process** en VHDL.
- Se declaran dentro del módulo como variables miembro.
- Se usan como cualquier otra variable de C/C++.

```
#include <systemc.h>
```

```
SC_MODULE(counter) {
    sc_in<sc_logic> clk, reset;
    sc_out<sc_bv<4>> > cout;
```

```
    unsigned int count; ←
```

```
void p1() {
    if (reset.read() == 1) count = 0;
    else if (clk.posedge()) ++count;
```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Procesos: sincronización (I)

- Un módulo puede contener:
 - Varios procesos concurrentes y/o
 - Submódulos con sus propios procesos concurrentes.
- Es necesario que estos módulos puedan intercambiar información y para hacerlo deben sincronizarse entre sí.
- El mecanismo más sencillo es el uso de señales, semejantes a las de VHDL.
- En SystemC una señal, **sc_signal**, es una clase basada en plantilla instanciada para un tipo determinado.
- Como en VHDL, la asignación tiene efecto un Δt después (siguiente ciclo de simulación).

Procesos: sincronización (II)

- Cuando se requiere una sincronización:
 - Con un nivel de abstracción mayor.
 - Con una funcionalidad adicional (buffers, recursos compartidos...).
- SystemC usa “canales” que pueden implementar cualquier esquema de comunicación.
- SystemC viene con algunos canales predefinidos y siempre se pueden definir nuevos tipos de canales desde cero o a partir de los predefinidos.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Procesos: ejemplo Δt

```
#include <systemc.h>

SC_MODULE(delta) {
  sc_in<bool > clk;
  sc_in<bool > din;
  sc_out<bool > dout;

  sc_signal<bool > q_s, clk_s;

  void p1() { q_s.write(din.read()); }
  void p2() { dout.write(q_s.read()); }
  void p3() { clk_s.write(clk.read()); }

  SC_CTOR(delta) {
    SC_METHOD(p1);
    sensitive << clk.pos();
    SC_METHOD(p2);
    sensitive << clk_s.posedge_event();
    SC_METHOD(p3);
    sensitive << clk;
  }
};

library ieee;
use ieee.std_logic_1164.all;

entity delta is
  port (clk, din: in std_logic;
        dout : out std_logic);
end delta;

architecture rtl of delta is
  signal q_i, clk_s: std_logic := '0';
begin
  p1: process (clk)
  begin
    if rising_edge(clk) then
      q_i <= not q_i;
    end if;
  end process;

  clk_s <= clk; -- delta delay!

  process (clk_s)
  begin
    if rising_edge(clk_s) then
      dout <= q_s;
    end if;
  end process;
end rtl;
```


Uso directo constructor C++

- **SC_CTOR** es un macro que oculta la sintaxis del constructor de C++ y ciertas inicializaciones.
- Sin embargo hay ocasiones donde es necesario emplear C++ directamente. Por ej.: cuando se quieren pasar parámetros adicionales al constructor.
- En ese caso no se puede usar SC_CTOR (sólo acepta un parámetro, el nombre del módulo).
- Si ocurre esto y se usan procesos (lo habitual) se debe

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70

Cartagena99


 UNIVERSIDAD POLITÉCNICA DE MADRID

SC_HAS_PROCESS

```

SC_MODULE(count) {
  sc_in<bool> clk;
  sc_out<int> dout;

  int cnt;


  void p1() { dout.write(cnt++); }

  count(sc_module_name _n, int _cnt) : sc_module(_n), cnt(_cnt) {
    SC_METHOD(p1);
    sensitive << clk.pos();
    dont_initialize();
  }

  count(sc_module_name _n) : sc_module(_n) {
    cnt=8;
    SC_METHOD(p1);
    sensitive << clk.pos();
    dont_initialize();
  }

  SC_HAS_PROCESS(count);
};

count DUT1("dut1"); // Using default constructor, start counting at 8
count DUT2("dut2", 32); // Instantiate Device Under Test, start counting at 32
  
```


 UNIVERSIDAD POLITÉCNICA DE MADRID

Tipos de SystemC

Tipo VHDL	Tipo SystemC	Comentarios
bit	sc_bit	Mejor usar tipo nativo bool de C++
bit_vector	sc_bv	Más rápido que sc_lv en simulación
std_ulogic	sc_logic	Sólo 4 valores: 'X','Z','0' and '1'
std_ulogic_vector	sc_lv	Sólo 4 valores, 'X','Z','0' and '1'
std_logic	sc_logic_resolved	Con función de resolución para 'X','Z','0' y '1'
std_logic_vector	sc_signal_rv	Con función de resolución para 'X','Z','0' y '1'
boolean	bool	Recomendado en lugar de sc_bit



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

VHDL	SystemC
and, or, xor, not	&, , ^, ~
srl, sll	>>, <<
<=	=
:=	=
=, /=	==, !=
<, <=, >, >=	<, <=, >, >=
+, -, *, /, mod, rem	+, -, *, /, %
	++, --

Operación	VHDL	SystemC
Selección de bit	count(3)	count[3]
Selección grupo bits	count(5 downto 2)	count(5, 2) count.range(5, 2)
Concatenación	count(1) & count(3 downto 2)	(count[1], count(3, 2)) .concat(count[1], count(3, 2))
Attributes	count'LEFT	No tienen equivalente

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70

Cartagena99

SystemC: operaciones aritméticas (I)

- Los tipos enteros (**sc_int**, **sc_uint**, **sc_bigint**, **sc_biguint**) admiten los operadores enteros de C: **+**, **-**, *****, **/** y **%**.
- Los tipos de vectores (**sc_bv**, **sc_lv**) no admiten ninguna operación aritmética.
- Para operar sobre ellos se debe:
 1. Convertir a entero.
 2. Operar.
 3. Reconvertir a vector.

SystemC: operaciones aritméticas (II)

VHDL

```
-- Viejo estilo: desaconsejado (deprecated)  
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_unsigned.all;  
signal counter: std_logic_vector(7 downto 0);  
counter <= counter + 1;
```

```
-- Nuevo estilo: aconsejado (recommended)  
.....
```

SystemC

```
#include <systemc.h>  
sc_signal<sc_bv<8>> counter;  
counter.write(counter.read().to_uint() + 1);
```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Ejemplo: contador

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity cntnr is
  port(reset, clk : in std_logic;
        q: out std_logic_vector(3 downto 0);
end cntnr;
architecture behavioral of cntnr is
  signal q_i: std_logic_vector(q'range);
begin
  p1:process(reset, clk)
  begin
    if reset = '1' then
      q_i <= (others => '0');
    elsif rising_edge(clk) then
      q_i <= std_logic_vector(unsigned(q_i) + 1);
    end if;
  end process;
end behavioral;

#include <systemc.h>

SC_MODULE(cntnr) {
  sc_in<sc_logic> reset, clk;
  sc_out<sc_bv<4>> > q;

  void p1() {
    if (reset.read() == SC_LOGIC_1)
      q.write(0);
    else if (clk.posedge())
      q.write(q.read().to_uint() + 1);
  }

  SC_CTOR(cntnr) {
    SC_METHOD(p1);
    sensitive(reset, clk);
  }
};
  
```

SystemC: conversiones de tipo

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

port (
  dout : out std_logic_vector(3 downto 0)
  ...
);

signal cnt_i : integer range 0 to 15;

  sc_out<sc_lv<4>> > dout;
  sc_signal<sc_uint<4>> > cnt_i;


  // Convierte sc_lv a sc_uint
  cnt_i.write(din.read().to_uint());

  // Convierte sc_uint a sc_lv
  dout.write(
    static_cast<sc_lv<4>> >(cnt_i.read()));
  
```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70


ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70




 UNIVERSIDAD
POLITÉCNICA
DE MADRID

SC_MAIN

- Punto de entrada para la simulación de una descripción en SystemC.
- Se correspondería en VHDL con un *testbench*.
- Al ser una función no hay constructor y, por tanto, no se pueden usar procesos.
- En ella se instancia el módulo de nivel superior (*top level module*) y se arranca y para la simulación (con **sc_start()** y **sc_stop()**).
- Es posible pasar parámetros a través de la línea de comandos.


 UNIVERSIDAD
POLITÉCNICA
DE MADRID

Simulación: registro (I)

- Es posible crear un fichero en formato VCD en el que registrar las señales para su visionado posterior.
- Generación de un fichero VCD:
 - Declarar puntero al tipo de archivo **sc_trace_file()**.
 - Crear fichero VCD con **sc_create_vcd_trace_file()**.
 - Establecer la unidad de tiempo con **set_time_unit()**.
 - Añadir las señales de interés con **sc_trace()**.
 - Ejecutar la simulación.
 - Cerrar el archivo con **sc_close_vcd_trace_file()**.

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99



Simulación: registro (II)

- Para registrar señales en niveles inferiores de la jerarquía se debe proporcionar la “ruta” usando '.' como separador.
- Es posible trazar los ciclos delta en un archivo VCD asignando *true* a la variable **delta_cycles**.
- También se pueden añadir comentarios con **write_comment()**.




Simulación: aserciones

- Son de gran ayuda durante la depuración.
- Comprueban que se cumpla una condición y cuando no lo hace se activan.
- Permiten automatizar la verificación del funcionamiento.
- En SystemC se usa **sc_assert()**.
- También puede usarse el **assert()** de C/C++ si bien el mensaje presentado no es exactamente igual.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99



UNIVERSIDAD
POLITÉCNICA
DE MADRID

Ejemplo: DUT

```

library ieee;
use ieee.std_logic_1164.all;
entity dff is
    port(clk : in std_logic;
          din : in std_logic;
          dout: out std_logic);
end dff;
architecture behavioral of dff is
begin
    p1:process(clk)
    begin
        if rising_edge(clk) then
            dout <= din;
        end if;
    end process;
end behavioral;


#include <systemc.h>

SC_MODULE(dff) {
    sc_in<sc_logic> clk, din;
    sc_out<sc_logic > dout;

    void p1() {
        if (clk.posedge())
            dout = din;
        }

    SC_CTOR(dff) {
        SC_METHOD(p1);
        sensitive(clk);
    }
};

```



UNIVERSIDAD
POLITÉCNICA
DE MADRID

Ejemplo: Testbench

```

#include <systemc.h>
#include "dff.h"
int sc_main(int argc, char* argv[])
{
    sc_signal<bool> din, dout;

    sc_clock clk("clk", 10, SC_NS, 0.5);

    dff DUT("dff");
    DUT.din(din);

    sc_trace(fp, clk, "clk");
    sc_trace(fp, din, "din");
    sc_trace(fp, dout, "dout");


    sc_start(31, SC_NS);
    din = true;
    sc_start(31, SC_NS);
    din = false;
    sc_start(31, SC_NS);
};

```

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**


**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

 UNIVERSIDAD
POLITÉCNICA
DE MADRID

SystemC: Interfaces

- Una interfaz proporciona un conjunto de métodos u operaciones, pero no los implementa:
 - Son clases abstractas.
 - Todos los métodos de una interfaz son funciones virtuales puras.
- Se utilizan para declarar los métodos que debe implementar un canal.
- Son útiles cuando se modela la comunicación entre las distintas capas de un sistema.
- Todos las interfaces descienden de `sc_interface`.

 UNIVERSIDAD
POLITÉCNICA
DE MADRID

SystemC: Canales

- Un canal implementa los métodos de uno o más interfaces y sirve de contenedor para la funcionalidad de la comunicación.
- Diferentes canales pueden implementar una misma interfaz de diferente forma.
- Un canal puede implementar más de una interfaz.
- Un canal puede conectarse a más de dos módulos.
- Se distingue entre canales primitivos (*primitive channels*)

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Canales Primitivos (I)

- Un canal es primitivo cuando no contiene ninguna jerarquía o proceso.
- Todos los canales primitivos descienden de **sc_prim_channel**.
- SystemC incorpora varios canales de este tipo.
- Los más simples son:
 - sc_signal
 - sc_mutex
 - sc_semaphore
 - sc_fifo

Canales Primitivos (II)

- **sc_signal**: es el canal más clásico y más simple.
- **sc_mutex**:
 - Permite a múltiples hilos de un programa compartir un recurso común sin entrar en colisión.
 - Cualquier proceso que necesite el recurso debe adquirirlo bloqueando el objeto mutex (*lock*) para obtener acceso exclusivo al recurso. Cuando ya no necesite el recurso, debe liberarlo desbloqueando el objeto mutex (*unlock*).
 - Ejemplo: arbitraje de un *bus* compartido.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Canales Primitivos (III)

- **sc_semaphore:**
 - Se usa para gestionar el caso en que existen más de una copia de un recurso o un recurso puede estar en posesión de más de un dueño.
 - Cuando se crea un semáforo es necesario especificar cuantas copias del recurso están disponibles.
 - Un mutex es un semáforo con una cuenta de uno.
 - Para acceder a los recursos gestionados por semáforo es necesario esperar a que uno quede libre, tomar posesión de él, y, después, cuando se ya no sea necesario, notificarlo (de forma semejante como se hace con mutex).

Canales Primitivos (IV)

- **sc_fifo:**
 - Fifo (*first in, first out*) es probablemente el canal más popular para el modelado a nivel de arquitectura.
 - Por defecto, **sc_fifo** tiene una profundidad de 16 elementos.
 - Es necesario especificar el tipo de dato que almacena el canal fifo.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Canales Primitivos: evaluar-actualizar

- Un canal primitivo debe soportar el mecanismo de acceso evaluar-actualizar (*evaluate-update*):
 - El mecanismo evaluar-actualizar está diseñado para simular la concurrencia.
 - Cuando ocurren acciones simultáneas sobre un canal (p.ej.: lectura/escritura concurrente en un canal), éstas deben serializarse en la práctica.
 - La actualización del valor de un canal sólo se hará efectiva después de que todos los procesos activos actualmente finalicen su ejecución o lleguen a un punto de sincronización.

Canales Jerárquicos

- Pueden tener una estructura, contener procesos y acceder directamente a otros canales.
- Todos derivan de **sc_channel**.
- **sc_channel** es simplemente una redefinición de **sc_module**, así que, desde el punto de vista del lenguaje, un canal jerárquico no es otra cosa que un módulo.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Uso canales primitivos y jerárquicos

- Usar canales primitivos siempre que:
 - Se necesite usar un esquema de comunicación petición-actualización (*request-update*).
 - Los canales sean atómicos y no puedan descomponerse en partes más pequeñas.
 - La velocidad sea absolutamente crucial (reduce con frecuencia el número total de ciclos delta).
 - No tenga sentido intentar construir un canal a partir de procesos y otros canales.
- Usar canales jerárquicos sólo cuando:
 - Los canales sean verdaderamente jerárquicos y los usuarios quieran tener la posibilidad de explorar la estructura interna.
 - Los canales contengan procesos.
 - Los canales contengan otros canales.
- Ambos tipos se pueden refinar. El refinamiento tiene que ver con las interfaces, no con si un canal es primitivo o jerárquico.

Canales: Conexión (I)

- Un módulo y sus procesos acceden a la interfaz de un canal a través de un puerto.
- Cada tipo de puerto asume una cierta interfaz. Un canal no puede conectarse a un puerto si éste no implementa la interfaz del puerto:
`sc_port<interface<type>, N > p;`
(N = número de canales que pueden conectarse al puerto)
- Todos los puertos descienden de **sc_port**.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Canales: conexión (II)

- Capacidad multi-puerto (*multi-port capability*): es posible conectar un puerto a múltiples canales.
- Acceso a canales sin puerto (*port-less channel access*):
 - Para la comunicación entre módulos se deben usar puertos para conectar los módulos a los canales.
 - Para la comunicación intramódulo, se permite acceder directamente a los canales sin usar puertos.
 - Para ello se llama directamente a los métodos de la interfaz del canal.

Canales: Conexión (III)

- Los módulos se conectan a los canales **después** de haber instanciado los módulos y los canales.
- Hay dos formas de asociar los puertos a canales:
 - Por nombre (recomendado).
 - Por posición (desaconsejado).

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Canales: Conexión (IV)

VHDL	SystemC	Observaciones
<pre>dff1: dff port map (clk => clk, reset => reset, din => din, dout => q1_s);</pre>	<pre>dff1.clk(clk); dff1.reset(reset); dff1.din(din); dff1.dout(q1_s);</pre>	Por nombre, estilo recomendado
<pre>dff1: dff port map (clk, reset, din , q1_s);</pre>	<pre>dff1 << clk << din << dout;</pre>	Por posición, estilo desaconsejado
<pre>dff1: dff port map (clk, reset, din , q1_s);</pre>	<pre>dff1(clk); dff1(reset); dff1(din); dff1(q1_s);</pre>	Por posición, estilo desaconsejado


SystemC: Eventos (I)

- Se emplean para sincronizar procesos.
- Se declaran con el tipo **sc_event**:
sc_event e;
- Se activan con **notify()**.
- Un evento en SystemC es el resultado de dicha activación y no tiene valor o duración sino que ocurre en un instante dado de tiempo y puede disparar uno o varios procesos.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70


ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70

Cartagena99

 UNIVERSIDAD
POLITÉCNICA
DE MADRID

SystemC: Eventos (II)

- La notificación puede ser inmediata o diferida:
e.notify(); // Notificación inmediata
e.notify(10,SC_MS); // Notif. diferida 10ms
- Una nueva notificación reemplaza cualquier notificación diferida pendiente.
- Si resulta conveniente, una notificación diferida puede cancelarse:
e.cancel(); // Cancela la última notificación

 UNIVERSIDAD
POLITÉCNICA
DE MADRID

SystemC: Eventos (III)

- Gracias a ellos se tienen dos posibilidades para suspender la ejecución de un proceso:
 - Esperar una cantidad fija de tiempo: **wait**(10, SC_NS);
 - Esperar hasta la activación de un evento: **wait**(e);
- Sólo las sentencias **wait** y la activación de eventos hacen avanzar el tiempo simulado.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Ejemplo (I): objetivos

- Mostrar el desarrollo de un canal y sus interfaces:
 - fifo, read_if, write_if
- Mostrar el uso de eventos:
 - fifo#write_event, fifo#read_event
- Mostrar como crear un componente con subcomponentes:
 - top
- Mostrar el uso de constructores con parámetros adicionales:
 - top
- Mostrar como puede utilizarse SystemC para sacar conclusiones muy al principio del proceso de desarrollo.

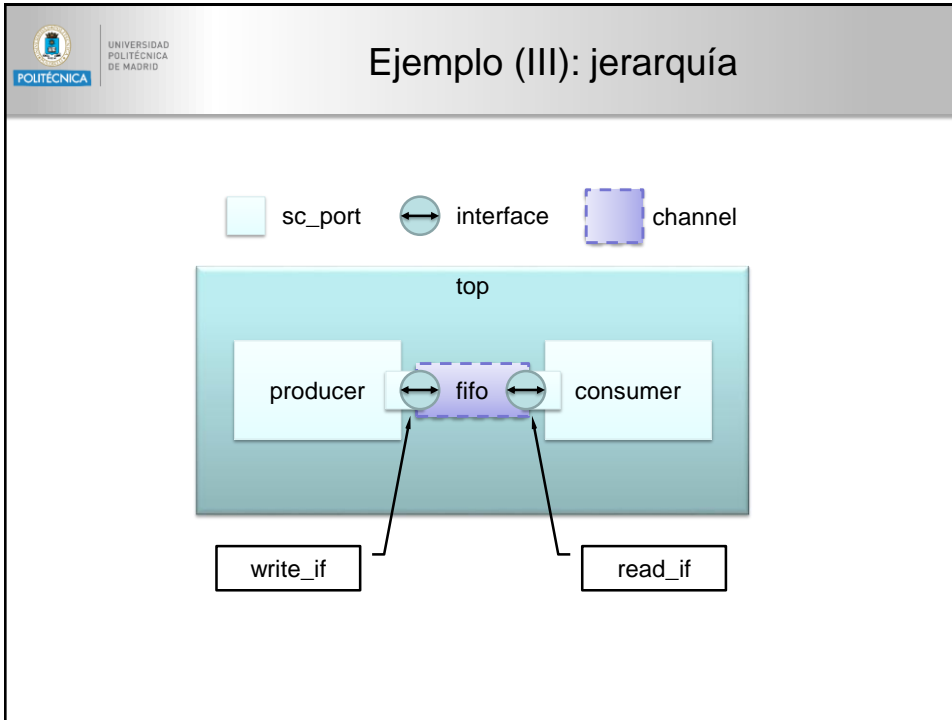
Ejemplo (II): especificación del problema

- Caso: interconexión de 2 sistemas
 - Un productor de datos (**producer**):
 - Emite mensajes a un ritmo fijo de 1 mensaje / μ s.
 - Los mensajes tienen una longitud aleatoria entre 1 y 19 caracteres.
 - Un consumidor de datos (**consumer**):
 - Procesa caracteres a un ritmo fijo de 1 carácter / 100 ns.
- Problema:
 - Si el mensaje tiene más de 10 caracteres...
 - No es posible procesarlo en 1 μ s: el productor debe esperar.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99



UNIVERSIDAD POLITÉCNICA DE MADRID

Ejemplo (IV): interfaces del canal

```

// Interfaz para escribir en el canal
class write_if : virtual public sc_interface
{
public:
    virtual void write(char) = 0; // Escribe carácter en la memoria
    virtual void reset() = 0;    // Borra memoria
};

// Interfaz para leer del canal
class read_if : virtual public sc_interface
{
public:
    virtual void read(char &) = 0; // Lee un carácter de la memoria

```



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Ejemplo (Va): realización interfaz (canal)

```
class fifo : public sc_channel, public write_if, public read_if
{
    char *data;
    int num_elements, first;
    sc_event write_event, read_event;
    int size, num_read, max_used, average;
    sc_time last_time;

    void compute_stats() {
        average += num_elements;
        if (num_elements > max_used)
            max_used = num_elements;
        ++num_read;
    }
}
```

Ejemplo (Vb): realización interfaz (canal)

```
public:
    fifo(sc_module_name name, int size_) : sc_channel(name), size(size_) {
        data = new char[size];
        num_elements = first = 0;
        num_read = max_used = average = 0;
        last_time = SC_ZERO_TIME;
    }

    ~fifo() {
        delete[] data;

        cout << endl << "Fifo size is: " << size << endl;
        cout << "Average fifo fill depth: " <<
```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Ejemplo (Vc): realización interfaz (canal)

```
// write_if interface implementation

void write(char c) {
    if (num_elements == size)
        wait(read_event);
    data[(first + num_elements) % size] = c;
    ++ num_elements;
    write_event.notify();
}

void reset() { num_elements = first = 0; }
```

Ejemplo (Vd): realización interfaz (canal)

```
// read_if interface implementation

void read(char &c){
    last_time = sc_time_stamp();
    if (num_elements == 0)
        wait(write_event);
    compute_stats();
    c = data[first];
    -- num_elements;
    first = (first + 1) % size;
    read_event.notify();
}
```

Cartagena99

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Ejemplo (VI): modelado del productor

```
SC_MODULE(producer) {
    sc_port<write_if> out;

    void p1() {
        const char *data = "0123456789012345678901234567890123456789";
        int total = 100000;
        while (true) {
            int len = 1 + int(19.0 * rand() / RAND_MAX);
            for (int i = 0; i < len; ++i) {
                out->write(data[i]);
                --total;
            }
            if (total <= 0)
                break;
            wait(1, SC_US); // 1 mensaje / us
        }
    }

    SC_CTOR(producer) {
        SC_THREAD(p1);
    }
};
```

Ejemplo (VII): modelado del consumidor

```
SC_MODULE(consumer) {
    sc_port<read_if> in;

    void p1() {
        char c;

        while (true) {
            in->read(c);
            wait(100, SC_NS); // Procesa 1 caracter / 100 ns
        }
    }

    SC_CTOR(consumer) {
```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Ejemplo (VIII): ensamblado del sistema

```
SC_MODULE(top) {
    fifo    fifo_inst;
    producer prod_inst;
    consumer cons_inst;

    top(sc_module_name name, int size) :
        sc_module(name),
        fifo_inst("Fifo", size),
        prod_inst("Producer"),
        cons_inst("Consumer")
    {
        prod_inst.out(fifo_inst);
        cons_inst.in(fifo_inst);
    }
};
```

Ejemplo (IX): *testbench*

```
int sc_main(int argc , char *argv[])
{
    int size = 10;
    if (argc > 1)
        size = atoi(argv[1]);
    if (size < 1)
        size = 1;
    if (size > 100000)
        size = 100000;


    top dut("Top", size);

    sc start();
}
```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99


 UNIVERSIDAD
POLITÉCNICA
DE MADRID

Ejemplo (X): tiempo base

- Resultados sin usar FIFO (capacidad = 1 byte)


```

$ ./top_tb 1

SystemC 2.3.3-Accellera --- Nov 26 2018 18:32:50
Copyright (c) 1996-2018 by all Contributors,
ALL RIGHTS RESERVED

Fifo size is: 1
Average fifo fill depth: 1
Maximum fifo fill depth: 1
Average transfer time per character: 181067 ps
Total characters transferred: 100007
Total time: 18108 us
  
```

- Tasa de procesamiento un 45% inferior a la máxima teórica (181 ns / caracter).


 UNIVERSIDAD
POLITÉCNICA
DE MADRID

Ejemplo (XI): iteración 1

- Resultados con FIFO 1 kbyte:

```

$ ./top_tb 1024


SystemC 2.3.3-Accellera --- Nov 26 2018 18:32:50
Copyright (c) 1996-2018 by all Contributors,
ALL RIGHTS RESERVED

Fifo size is: 1024
Average fifo fill depth: 118.555
Maximum fifo fill depth: 313
Average transfer time per character: 100700 ps
  
```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70

Cartagena99


 UNIVERSIDAD
POLITÉCNICA
DE MADRID

Ejemplo (XII): iteración 2

- Resultados con FIFO 256 bytes:


```

$ ./top_tb 256

SystemC 2.3.3-Accellera --- Nov 26 2018 18:32:50
Copyright (c) 1996-2018 by all Contributors,
ALL RIGHTS RESERVED

Fifo size is: 256
Average fifo fill depth: 108.954
Maximum fifo fill depth: 256
Average transfer time per character: 100756 ps
Total characters transferred: 100007
Total time: 10076300 ns
  
```

- La penalización es casi nula y no desperdiciamos memoria.


 UNIVERSIDAD
POLITÉCNICA
DE MADRID

Ejemplo (XIII): conclusión

- Con:
 - Recursos y tiempo de desarrollo limitados,
 - Un modelo relativamente poco elaborado,
 - Sin dar detalles de la implementación del HW,
 - Sin prototipo físicos...
- Hemos conseguido:
 - Determinar un parámetro crítico (tamaño del buffer) para obtener el máximo rendimiento del sistema.
 - Optimizar el coste del producto usando sólo la memoria

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99