

Facultad de Informática

Universidad Complutense de Madrid

### LABORATORIO DE FUNDAMENTOS DE COMPUTADORES I

# PRÁCTICA 2:

### DISEÑO Y SIMULACIÓN DE UN SUMADOR BINARIO DE 2 BITS MEDIANTE VHDL

El objetivo de esta práctica es tomar contacto con la herramienta de simulación de circuitos integrados, ModelSim de Mentor Graphics y con el lenguaje de descripción de hardware VHDL. Para ello se va a implementar y simular un sumador binario de 2 bits.

## **DISEÑO CON VHDL**

Un diseño VHDL tiene dos partes:

- Entity.
- Architecture.

La entity describe la interfaz del módulo, es decir, define las entradas, las salidas y sus tipos. En este laboratorio sólo estudiaremos dos tipos, a saber:

- Tipo bit
- Tipo bit\_vector (n-1 downto 0) que define un vector de n bits.

En la siguiente figura se puede ver la caja negra que describe el interfaz del sumador total que vamos a implementar inicialmente



La declaración de entidad para esta caja negra es la siguiente:

ENTITY sumador IS PORT(op1, op2 : IN bit; c\_ent : IN bit; sum : OUT bit; c\_sal : OUT bit); END sumador;

Como se puede apreciar, el módulo tiene:

- Tres entradas de datos de tipo bit.
- Dos salidas, c\_sal y sum de tipo bit.

Por otro lado, la arquitectura describe la funcionalidad del módulo. Una misma entidad puede tener muchas arquitecturas diferentes. En este curso nos centraremos en dos tipos de arquitecturas:

- Descripción de la estructura del módulo mediante instancias de componentes previamente diseñadas y conectadas entre sí mediante señales.
- Descripción del comportamiento del módulo mediante expresiones lógicas.

En esta práctica vamos a realizar una descripción estructural de la arquitectura del sumador total utilizando las puertas lógicas and2, or2 y xor2 incluidas en el archivo sumador.vhd que se proporciona al alumno. En concreto ese archivo implementa la siguiente red de puertas lógicas:



A las conexiones internas de esta red las vamos a nombrar de la siguiente manera:

- sal\_and1: la conexión entre and1 y or
- sal\_and2: conexión entre and2 y or
- sal\_xor1 la conexión entre xor1 y xor2 y and2

Visto lo anterior, la arquitectura estructural del sumador total es la siguiente:

ENTITY full\_adder IS PORT(ai, bi : IN bit; ci : IN bit; si : OUT bit; co : OUT bit); END full\_adder; --USE work.ALL; ARCHITECTURE puertas OF full\_adder IS --declaración de componentes COMPONENT or2 PORT(i1, i2 : IN bit; o : OUT bit); END COMPONENT; COMPONENT and2 PORT(i1, i2 : IN bit; o : OUT bit); END COMPONENT: COMPONENT xor2 PORT(i1, i2 : IN bit; o : OUT bit); END COMPONENT: --declaración de señales internas SIGNAL sal\_and1, sal\_and2, sal\_xor1: bit; --empieza el cuerpo de la arquitectura BEGIN i\_and1 : and2 PORT MAP(ai, bi, sal\_and1); i\_xor1 : xor2 PORT MAP(ai, bi, sal\_xor1); i\_and2 : and2 PORT MAP(sal\_and1, ci, sal\_and2); i\_xor2 : xor2 PORT MAP(sal\_xor1, ci, si); i\_or : or2 PORT MAP(sal\_and1, sal\_and2, co);

END puertas;

En este fragmento de código se pueden observar los siguientes elementos:

Después de la cabecera de la arquitectura aparece la declaración de los componentes que se van a utilizar en el diseño. La declaración de componentes sirve para establecer el tipo de

diseños que se van a utilizar como elemento de construcción de otro diseño. En nuestro caso se declaran tres componentes la puerta lógica or2, la and2 y la xor2. Por ejemplo, el fragmento de código:

COMPONENT or2 PORT(i1, i2 : IN bit; o : OUT bit); END COMPONENT;

declara el componente or2, definido por dos entradas y una salida, de tipo bit.

A continuación, se declaran las señales internas que conectan las diferentes puertas lógicas entre sí, en concreto se declaran las señales sal\_and1, sal\_and2 y al\_xor1.

SIGNAL sal\_and1, sal\_and2, sal\_xor1 : bit;

Ya dentro del cuerpo de la arquitectura aparecen las instancias de las componentes que vamos a utilizar. Como hemos visto un componente es el tipo de diseño que utilizamos como elemento de construcción, y una instancia es el elemento de construcción propiamente dicho. Si nos fijamos en la figura que contiene la red de puertas lógicas que implementa el sumador total vemos que esta necesita puertas de tipo and2, or2 y xor2. Esto son los tipos de componente que tiene que utilizar. En concreto necesita dos puertas and, dos puertas xor y una puerta and. Estos son las instancias de las componentes que se pueden ver en el fragmento de código.

Nota: Los componentes deben estar implementados previamente a su uso. En este laboratorio se le proporcionaran al alumno en el archivo full\_adder.

# MODELSIM

#### DESCARGA E INSTALACIÓN

El ModelSim es un simulador de lenguajes de descripción de hardware. Nos podemos descargar una versión gratuita del mismo, llamada ModelSim PE Student Edition, del siguiente enlace:

https://www.mentor.com/company/higher\_ed/modelsim-student-edition

cuando os lo descarguéis os enviarán un correo con una descripción detallada de todos los pasos que tenéis que seguir para instalar la herramienta

#### **CREAR UN PROYECTO EN MODELSIM**

- 1. Lo primero es crear un directorio que vamos a llamar fc1\_practica2.
- 2. En este directorio copiamos el archivo full\_adder.vhd proporcionado al alumno
- 3. abrimos el ModelSim, y cerramos la ventana de bienvenida

📓 - 🚘 🔲 🤹 🍈 l 🐰 🐚 🏦 🕫 🌣	·   🔿 - 🛤 🗄		100 📥 🛍 🛍 🛍 🕷 🕴	n n 🔺	
Library	A Type	Path			_
din foatfiylib	Library	SMODEL_TECH/ /floatfixlib			
Jil ieee	Library	\$MODEL_TECH//eee	M IMPORTANT Information		×
lieee env (empty)	Library	\$MODEL TECH//ieee env			
In infact	Library	\$MODEL_TECH//infact	ModelSim	Welcome to version 10.4	a
h mc2_lib (empty)	Library	\$MODEL_TECH//mc2_lib			
ngc_ams (empty)	Library	\$MODEL_TECH//mgc_ams	The ModelSim PE St	tudent Edition is designed with yo	ur 🖆
nodelsim_lib	Library	\$MODEL_TECH//modelsim_lib	needs in mind and pr	ovides an opportunity for you to	
ntiAvm	Library	\$MODEL_TECH//avm	make use of and lear	n more about VHDL or Verilog	
ntiOvm	Library	\$MODEL_TECH//ovm-2.1.2	simulation outside of	school	
ntiPA	Library	\$MODEL_TECH//pa_lib	simulation outside of	senool.	
ntiRnm	Library	\$MODEL_TECH//rnm	There is no support r	workided with this product but we	
ntiUPF	Library	\$MODEL_TECH//upf_lib	There is no support p	browneed with this product but we	
ntiUvm	Library	\$MODEL_TECH//uvm-1.1d	have included a full d	locumentation set and a tutorial to	
osvvm	Library	\$MODEL_TECH//osvvm	help you get started.	Select Help > ModelSim PDF	
t std	Library	\$MODEL_TECH//std	Bookcase to access t	the documentation.	
t std_developerskit	Library	\$MODEL_TECH//std_developerskit			
sv_std	Library	\$MODEL_TECH//sv_std			
synopsys	Library	\$MODEL_TECH//synopsys			
l verilog	Library	\$MODEL_TECH//verilog			
t vhdlopt_lib	Library	\$MODEL_TECH//vhdlopt_lib			
vital2000	Library	\$MODEL_TECH//vital2000			
Library 🗙 🖺 Files 🗶 🏙 Project 🛛					
Transcript					
Reading C:/Modeltech pe edu 10.4	a/tcl/vsim/p	ref.to]	-		
acading official percenting	a, cor, vorm, p.				
			<b>▲</b>		▶
			Dep't show this diplace people	Select Jumpstart to Lumpstart	Class

### 4. Seleccionamos File→New→Project...

м	ModelS	Sim PE S	Student Edi	tio	on 10.4a		
File	Edit	View	Compile	S	Simulate	Add	Project
N	ew			۲	Folder	r	ģ
0	pen				Sourc	e	<u> </u>
Lo	bad			۲	Projec	ct	
C	lose Pro	ject			Librar	y	I
In	nport			۲	Debug	g Archiv	e
E	xport			•	V	HDL	
Si	ave		Ctrl+S		v	HDL	
Si	ave As,						
R	eport						
0	hange [	Directory	/				
	se Sour	ce					
	burce D	rectory					
D	atasets						
E	nvironm	ent		F			
Pi	age Set	up			L		
Pi	rint						
Pi	rint Pos	tscript					
R	ecent D	irectorie	s	Þ			
R	ecent P	rojects		۲			
C	lose Wir	ndow					
Q	uit				<u> </u>		

5. Se rellena la ventana que aparece con el nombre del proyecto (fc1\_practica2) y se busca el directorio fc1\_practica2 que hemos creado con anterioridad, y pulsas OK



6. De la ventana que aparece seleccionas la opción Add Existing File



7. Seleccionar con el browser el archivo full\_adder.vhd que has almacenado con anterioridad en el directorios fc1\_practica2

Add file to Project	×
File Name C:/A_REPOSITORIO/docencia/	laboratorio_FC1/fc1 Browse
Add file as type	Folder
Reference from current location	$\ensuremath{\mathbb{C}}$ Copy to project directory
	OK Cancel

8. En la ventana proyecto aparece el archivo full\_adder.vhd con una interrogación azul, lo que indica que todavía no se ha compilado

M	/lodel	Sim PE S	tude	nt Edi	tion 10	4a						
File	Edit	View	Con	npile	Simu	ate	Add		Proje	ct	Tools	Layout
	- 🖻	93	Ċ	Å	ùa 🗌	8 20	20		Ø •	М	€- €-	
🛗 Pr	oject -	C:/A_RE	POSI	TORIC	/docen	tia/lał	oorato	orio	_FC1/	fc1	practi	ca2/fc1_pra
▼ Nam	ne			St	atus	T	/pe			<u>م 0</u>	rder	
	full_	adder.vh	d	?	,	VI	HDL			0		

9. Para compilar, seleccionar Compile  $\rightarrow$  Compile Selected



Si la compilación ha sido correcta la interrogación se ve sustituida por una v verde



Además, seleccionando la pestaña Library



Se abre una ventana en la que aparece entre otras la librería work en la que se incluyen las entidades vhdl compiladas. Darse cuenta que en work aparecen las cuatro entidades contenidas en el archivo full\_adder.vhd.

ModelSim PE Student Edition 10.4a
File Edit View Compile Simulate
🖻 • 🗃 🔛 🦈 🍈   🐰 🐚 🍭 🖆
Library
▼ Name
- Mu work
u verilog
synopsys
⊕std_developerskit
std
H → M mtiUPF
👖 Library 🗙 📓 Files 🗙 🛗 Project 🗙

10. Para simular el archivo seleccionar Simulate→Start Simulation



Aparece la ventana Start Simulation, despliega el contenido del directorio work y selecciona el archivo full\_adder y ok

reagen more vering contained		
Name	∇ Type	Path
- work	Library	work
<u> </u> <u> </u> <u> </u> <u> </u> xor2	Entity	C:/A_REPOSITORI
<u> </u> <u> </u> <u> </u> or2	Entity	C:/A_REPOSITORI
	Entity	C:/A_REPOSITORI
🔁 🕒 and2	Entity	C:/A_REPOSITORI
→ /// vital2000	Library	\$MODEL_TECH//
	Library	\$MODEL_TECH//
	Library	\$MODEL_TECH//
+	Library	\$MODEL_TECH//:
+	Library	\$MODEL_TECH//
		•
Design Unit(s)		Resolution
work full addar		default

En la ventana transcript aparece la siguiente información indicando que se han cargado correctamente todas las entidades necesarias del diseño



Además, se abre una nueva ventana bajo la pestaña Sim

ModelSim PE St	udent Edition 10.4a	_		×
File Edit View	Compile Simulate Add Objects Tools Layout Bookmarks Window Help			
🛛 🖻 🕶 🖬 🖏 i	æ   X № @ £2.2   Ø - <b>A E</b>			
<u>s</u>   🕇 🖛 🖦	Ef 100 ns 🔶 EL EL EL 🗱 🍩   🜇 🚳			
🔊 sim - Default 💷	::::::::::::::::::::::::::::::::::::::		:	+ 🗗 🗙
▼ Instance	△ Design unit 🛛 🕅 Name		14 🗖 N	ow 🄊 🕨
full_adder	full_adder(puertas) 4/2 ai			
🔁 🗾 i_and1	and2(comportamiento)			
i_xor1	xor2(comportamiento)			
+- i xor2	xor2(comportamiento)			
	or2(comportamiento)			
🗾 standard	standard 🥠 sal_xor1			
	🐴 si			
•	• •			
👖 Library 🗙 📓 File	es 🛪 🏙 Project 🛪 🟭 sim 🛪 🔹 🔹			Þ
	Project : fc1_practica2 Now: 0 ns Delta: 0		sim:/full_a	dder

En la parte izquierda de esta ventana aparecen la jerarquia del diseño con sus instancias, en este caso de full\_adder cuelgan las instancias i\_and1, i\_and2,..etc. en la parte de la derecha aparece la ventana objetos que incluye todas la entradas, salidas y señales internas del diseño. En ocasiones esta ventana no se abre automáticamente, para abrila seleccionar view→Objects

11. Selección de señales que se quieren simular. Selecciona todas las señales que se quieren simular, pincha derecha y selecciona add wave

술 Objects			*****	+ a ×
▼ Name			△ Value	년 🗖 Now 🌶 🕨
👍 ai			0	
🥠 bi			0	
🔔 ci			0	
<u> </u>			0	
sal_	_and1		0	
sal_	_and2		0	
	2001		0	
<b>•</b>	View Declaration			
	View Memory Conter	nts		
	Add Wave	CH-LUW		
	Add Wave New	Ctri+w		
	Add Wave New			
	Add wave to	61.L.B		
	Add Datanow	Ctri+D		
_	Add to			
	UPF			
	Сору	Ctrl+C		
	Find	Ctrl+F		
	Insert Breakpoint			
	Toggle Coverage		•	
	Modify			
	Radix			
	Show			

Como resultado se abre la venta de ondas en las que se van a observar los cronogramas de la simulación



12. Añadir señales de entrada al simulador. En nuestro caso las señales de entrada al sumador son ai, bi, ci, por lo tanto debemos dar valores a estas entradas para ver si el diseño es correcto. Por ejemplo vamos a dar los valores 1,1,1 a estas tres entradas y vamos a comprobar en el cronograma que las dos salidas si y co se ponen a 1. Para ello selecciona con la derecha la señal ai y selecciona force y en value pon un uno

Force Selected Signal	$\times$
Signal Name: sim:/full_adder/ai	-1
Value: 1	-
_Kind	
Delay For: 0	
Cancel After:	
OK Canc	el

Y haz los mismo para bi y ci y selecciona el icono run



Y en la ventana aparece

Ī	😰 Wave - Default 🚞 🔤						
	🖹 • 🚔 🖬 % 🚭   🐰	🖻 🛍 ቧ 🚉	o • 🏘 🖶			s 🗱 🖽	
	] <b>3+ - →E - ⊡-</b> Search: [		🛨 (1) 🖏 👘	,			
	<b>∲</b> 1 •	Msgs					
	₄/full_adder/ai	1					
	↓/full_adder/bi	1					
	₄ /full_adder/ci	1					
	👍 /full_adder/co	1					
	🔶 /full_adder/si	1					
	/full_adder/sal_and1	1					
	/full_adder/sal_and2	1					
	/full_adder/sal_xor1	0					

Se puede ver que si y co valen 1 cuando ai, bi y ci valen 1.

13. Generar relojes para la simulación. Existe una forma de generar todas las posibles entradas del sumador y es utilizando relojes. Pinchar la entrada ai con la derecha y seleccionar clock



Aparece la ventana

M D	efine Clock X
	Clock Name
þ	im:/full_adder/ai
0	Duty
F	Cancel
	Logic Values High: 1 Low: 0
	First Edge
	OK Cancel

Deja todo como está y selecciona ok. Haz lo mismo para bi y si pero cambiando el periodo, en el primer caso a 200 y en el segundo a 400.

FE Wave						
File Edit View Add Format Tools Bookmarks Window Help						
ga Wave - Default						
▐▖▖▖▐▝▖▝▋▏▓▐▆▝▋▔▁░▕○▖▟▌▐▎▁▏░▓▓▓▓▋▓▋▁▙▞▕▚▆▐▓▎⊌▁▓▖						
💁 🕇 🖛 🖦   Ef 🔢 🖬 💷 E	: 🖩 👿 💿 👔 🗃 🌑 📄 🗍 🛱 두 드 구 또 한 한 한 한 한 한 한 한 한 한 한 한 한 한 한 한 한 한					
34 - →E - 30-   Search:						
Asgs Msgs						
🍫 /full_adder/ai 0						
₄ /full_adder/bi 1						
₄ /full_adder/ci 1						
/full_adder/si						
/full_adder/co						
/full_adder/sal_and1 0						
/full_adder/sal_and2 0						
/full_adder/sal_xor1 1						

#### **DESARROLLO DE LA PRÁCTICA**

- 1. Crear un proyecto que incluya el fichero full\_adder.vhd proporcionado al alumno, compilarlo y simularlo comprobando que el sumador funciona correctamente.
- 2. Modificar el archivo full\_adder de manera que implemente un sumador binario de dos bits mediante una arquitectura estructural que utilice que utilice el full\_adder.vhd del apartado anterior como componente, siguiendo la estructura explicada en las trasparencias.

