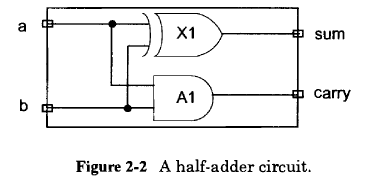
Ejemplos de modelos en SystemC

Circuito medio-sumador



La Figura 2-2 muestra un circuito medio-sumador. Su modelo en SystemC se muestra a continuación.

// File: half\_adder.h // Line 1

#include <systemc.h> // Line 2

SC\_MODULE (half\_adder) { // Line 3

sc\_in<bool> a, b; // Line 4

sc\_out<bool> sum, carry; // Line 5

void prc\_half\_adder (); // Line 6

SC\_CTOR (half\_adder) { // Line 7

SC\_METHOD (prc\_half\_adder);// Line 8

sensitive << a << b; // Line 9

} // Line 10

}; // Line 11

// File: half\_adder.cpp // Line 1

#include "half\_adder.h" // Line 2

void half\_adder::prc\_half\_adder () { // Line 3

sum = a ^ b; // Line 4

carry = a & b; // Line 5

}

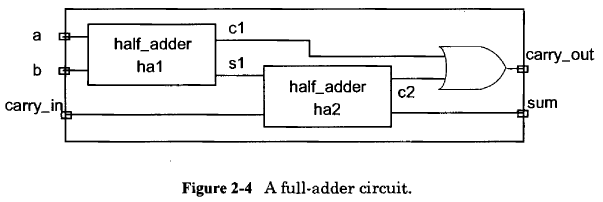
La palabra clave SC\_MODULE inicia la declaración de un módulo de SystemC. El nombre del módulo es especificado como half\_adder. El módulo tiene dos puertos puertos de entrada: a y b. Los puertos son declarados como de tipo bool (bool es un tipo estándar de C++). El módulo tiene dos puertos de salida: sum y carry. Estos también son de tipo bool.

El bloque SC\_CTOR en la línea 7 declara los procesos, y el tipo de procesos usados para describir el comportamiento del módulo. En el módulo half\_adder, el bloque SC\_CTOR declara un proceso de tipo SC\_METHOD. El nombre asociado con el bloque SC\_CTOR debe ser idéntico al nombre del módulo. Un proceso SC\_METHOD es sensitivo a un conjunto de señales especificadas y puertos y no se puede suspender con una sentencia wait; las sentencias wait no están permitidas en los procesos SC\_METHOD. El otro tipo de procesos, SC\_THREAD se describe con detalle en el capítulo 9 del libro [Bashker]. El nombre del proceso se especifica en la declaración del SC\_METHOD. En este caso, el nombre del proceso es prc\_half\_adder. Este está declarado en la línea 6. El proceso, el cual es una función miembro, debe devolver void y no tener argumentos. La sentencia sensitive que aparece en la línea 9 se usa para especificar el conjunto de señales y puertos a los que el proceso SC\_METHOD es sensitivo. Esta sentencia muestra que el proceso prc\_half\_adder es sensitivo a los puertos de entrada a y b. Esto implica que siempre que hay un cambio de valor en los puertos a o b, el proceso prc\_half\_adder es ejecutado completo, y cuando termina espera a que ocurra otro evento de su lista de sensitividad. Note que la declaración de SC\_MODULE termina en un punto y coma.

El segundo archivo half\_adder.cpp contiene la definición del proceso. Las dos sentencias de asignación en las líneas 4 y 5 calculan los valores de los puertos de salida sum y carry. Una asignación a un puerto siempre ocurre después de un retardo delta. Lo mismo es cierto para una asignación de señal. Lo que esto significa es, si el puerto a tiene un cambio en el tiempo 5ns, entonces sum y carry, obtienen sus nuevos valores actualizados en el tiempo %FontSize=12
%TeXFontSize=12
\documentclass{article}
\pagestyle{empty}
\begin{document}
\[
5+\Delta
\]
\end{document} ns.

Describiendo Jerarquía

La jerarquía de un diseño se puede especificar usando la misma construcción SC\_MODULE. En la Figura 2-4 se presenta un ejemplo de un circuito sumador completo (full\_adder), que instancia el circuito half\_adder descrito en una sección previa.



// File: full\_adder.h

/\*\* REF. Page 21, Chapter 2 Getting Started,

\* 2.4 Verifying the Functionality

\* J. Bhasker - A SystemC Primer-Star Galaxy Pub (2002).djvu

\*/

#ifndef FULL\_ADDER\_H\_INCLUDED

#define FULL\_ADDER\_H\_INCLUDED

#include "half\_adder.h"

SC\_MODULE (full\_adder) {

sc\_in<bool> a, b, carry\_in;

sc\_out<bool> sum, carry\_out;

sc\_signal<bool> c1, s1, c2;

void prc\_or ();

half\_adder \*ha1\_ptr, \*ha2\_ptr;

SC\_CTOR (full\_adder) {

ha1\_ptr = new half\_adder ("ha1");

// Named association:

ha1\_ptr->a (a);

ha1\_ptr->b (b);

ha1\_ptr->sum (s1);

ha1\_ptr->carry (c1);

ha2\_ptr = new half\_adder ("ha2");

// Positional association:

(\*ha2\_ptr) (s1, carry\_in, sum, c2);

SC\_METHOD (prc\_or);

sensitive << c1 << c2;

}

// A destructor:

~full\_adder() {

delete hal\_ptr;

delete ha2\_ptr;

}

};

#endif // FULL\_ADDER\_H\_INCLUDED

// File: full\_adder.cpp

/\*\* REF. Page 22, Chapter 2 Getting Started,

\* 2.4 Verifying the Functionality

\* J. Bhasker - A SystemC Primer-Star Galaxy Pub (2002).djvu

\*/

#include "full\_adder.h"

void full\_adder::prc\_or () {

carry\_out = c1 | c2;

}

/\*\*

\* Cuando el tama\~no de este archivo era menor a 512 (es

\* decir, menos de 512 bytes), en las propiedades del archivo

\* en Windows 10, el tama\~no del archivo en disco aparec\'ia

\* como de 0 bytes. Y al parecer esto es la causa de que en

\* en el CodeBlocks lanzado desde la App Debian el archivo

\* se abre "en blanco". Si el tam

\*/

El módulo full\_adder tiene tres puertos de entrada y dos puertos de salida. Todos son de tipo bool. Antes de la declaración de este módulo, se pone una directiva (#include “half\_adder.h”) que incluye la declaración del módulo half\_adder. El archivo systemc.h es incluido automáticamente a través de la inclusión del archivo half\_adder.h.

La línea:

sc\_signal<bool> c1, s1, c2;

declara las señales que son locales al módulo. Además, son declaradas como de tipo bool. Las señales son usadas para comunicación entre y contra procesos e instancias de módulo.

La línea:

half\_adder \*ha1\_ptr, \*ha2\_ptr;

declara dos apuntadores a módulo half\_adder, uno para cada instancia.

El bloque SC\_CTOR para el módulo full\_adder contiene dos instanciaciones de módulo y una declaración de proceso SC\_METHOD. La primera instanciación

ha1\_ptr = new half\_adder ("ha1");

crea una nueva instancia de módulo half\_adder con nombre de instancia ha1. El apuntador devuelto es guardado en ha1\_ptr el cual es usado para conectar señales externas o puertos a los puertos de la instancia. Hay dos formas de especificar tales interconexiones:

1. Usando asociación posicional.
2. Usando asociación nombrada.

En la instancia ha1 se usa asociación nombrada para la interconexión. El puerto a de la instancia ha1 es conectado al puerto a del módulo full\_adder. El puerto b de ha1 es conectado al puerto b del módulo full\_adder. El puerto sum de ha1 es conectado a la señal s1 mientras que el puerto carry es conectado a la señal c1.

La segunda instanciación de módulo half\_adder tiene nombre de instancia ha2 y un apuntador ha2\_ptr. En este caso, se usa asociación posicional para especificar la interconexión. El orden en el cual los puertos de instancia son conectados en la asociación son muy importantes. Las conexiones son hechas en el orden en que los puertos son declarados en el módulo. El puerto a de la instancia ha2 es conectado a la señal s1, el puerto b de ha2 es conectado al puerto carry\_in de full\_adder, y así sucesivamente.

El bloque SC\_CTOR también declara un proceso SC\_METHOD llamado prc\_or que es sensitivo a las señales c1 y c2, cuyo comportamiento es descrito en full\_adder.cpp. Este proceso calcula la lógica de una compuerta or de las dos señales carry intermedias para crear la salida carry\_out.

Este módulo tiene un destructor. El destructor borra la memoria que fue creada usando el operador new en el bloque SC\_CTOR (constructor). Esto es importante para evitar fugas de memoria (memory leaks). El siguiente es el formato de cómo se v un destructor en nuestro contexto.

~module\_name () {

delete ptrl;

delete ptr2;

. . .

// ptrl, ptr2, ... are all pointers that were allocated

// memory in the SC\_CTOR block using the new operator.

}

Note que el destructor se requiere solamente cuando se adquiere memoria a través del operador new en el bloque SC\_CTOR.

Verificación de la funcionalidad

Ahora que ya tenemos un modelo escrito en SystemC, ¿cóm se prueba la funcionalidad del módulo? Bueno, SystemC proporciona un framework y un conjunto de funciones para lograr esta tarea. Ese incluye generación de reloj y trazado de formas de onda.

Los aspectos de pruebas descritos en esta sección no son sintetizables, esto es, no son parte de SystemC RTL. La mayoría de las características que no son parte de SystemC RTL presentadas en esta sección se describen con mucho más detalle en el capítuo 8 del libro [Bashker].

Observemos nuevamente el módulo full\_adder. Digamos que queremos poner a prueba este módulo para todos los posibles valores de patrones de entrada. Cada patrón es aplicado cada 5ns, y las salidas del módulo son registradas cada vez que hay un cambio en las entradas o salidas del módulo full\_adder. Aquí se presenta un banco de pruebas que consigue esta tarea.

// File: driver.h

/\*\* REF. Page 24, Chapter 2 Getting Started,

\* 2.4 Verifying the Functionality

\* J. Bhasker - A SystemC Primer-Star Galaxy Pub (2002).djvu

\*/

#ifndef DRIVER\_H\_INCLUDED

#define DRIVER\_H\_INCLUDED

#include <systemc.h> /\* add -I<path to your systemc> in your

make file, or

Project->Build Options->Search Directories \*/

SC\_MODULE (driver) {

sc\_out<bool> d\_a, d\_b, d\_cin;

void prc\_driver ();

SC\_CTOR (driver) {

SC\_THREAD (prc\_driver);

}

};

#endif // DRIVER\_H\_INCLUDED

/\*\*

\* Cuando el tama\~no de esta archivo era menor a 512 (es

\* decir, menos de 512 bytes), en las propiedades del archivo

\* en Windows 10, el tama\~no del archivo en disco aparec\'ia

\* como de 0 bytes. Y al parecer esto es la causa de que en

\* en el CodeBlocks lanzado desde la App Debian el archivo

\* se abre "en blanco".

\*/

// File: driver.cpp

/\*\* REF. Page 24, Chapter 2 Getting Started,

\* 2.4 Verifying the Functionality

\* J. Bhasker - A SystemC Primer-Star Galaxy Pub (2002).djvu

\*/

#include "driver.h"

void driver::prc\_driver (){

sc\_uint<3> pattern;

pattern = 0;

while (1) {

d\_a = pattern[0];

d\_b = pattern[1];

d\_cin = pattern[2];

wait (5, SC\_NS);

pattern++;

}

}

/\*\*

\* Cuando el tama\~no de esta archivo era menor a 512 (es

\* decir, menos de 512 bytes), en las propiedades del archivo

\* en Windows 10, el tama\~no del archivo en disco aparec\'ia

\* como de 0 bytes. Y al parecer esto es la causa de que en

\* en el CodeBlocks lanzado desde la App Debian el archivo

\* se abre "en blanco".

\*/

// File: monitor.h

/\*\* REF. Page 25, Chapter 2 Getting Started,

\* 2.4 Verifying the Functionality

\* J. Bhasker - A SystemC Primer-Star Galaxy Pub (2002).djvu

\*/

#ifndef MONITOR\_H\_INCLUDED

#define MONITOR\_H\_INCLUDED

#include <systemc.h>

SC\_MODULE (monitor) {

sc\_in<bool> m\_a, m\_b, m\_cin, m\_sum, m\_cout;

void prc\_monitor ();

SC\_CTOR (monitor) {

SC\_METHOD (prc\_monitor);

sensitive << m\_a << m\_b << m\_cin << m\_sum<< m\_cout;

}

};

#endif // MONITOR\_H\_INCLUDED

/\*\*

\* Cuando el tama\~no de esta archivo era menor a 512 (es

\* decir, menos de 512 bytes), en las propiedades del archivo

\* en Windows 10, el tama\~no del archivo en disco aparec\'ia

\* como de 0 bytes. Y al parecer esto es la causa de que en

\* en el CodeBlocks lanzado desde la App Debian el archivo

\* se abre "en blanco".

\*/

// File: monitor.cpp

/\*\* REF. Page 25, Chapter 2 Getting Started,

\* 2.4 Verifying the Functionality

\* J. Bhasker - A SystemC Primer-Star Galaxy Pub (2002).djvu

\*/

#include "monitor.h"

void monitor::prc\_monitor () {

cout << "At time" << sc\_time\_stamp() << "::";

cout << "(a, b, carry\_in): ";

cout << m\_a << m\_b << m\_cin;

cout << " (sum, carry\_out): "<< m\_sum

<< m\_cout << endl;

}

/\*\*

\* Cuando el tama\~no de esta archivo era menor a 512 (es

\* decir, menos de 512 bytes), en las propiedades del archivo

\* en Windows 10, el tama\~no del archivo en disco aparec\'ia

\* como de 0 bytes. Y al parecer esto es la causa de que en

\* en el CodeBlocks lanzado desde la App Debian el archivo

\* se abre "en blanco".

\*/

// File: full\_adder\_main.cpp

/\*\* REF. Page 26, Chapter 2 Getting Started,

\* 2.4 Verifying the Functionality

\* J. Bhasker - A SystemC Primer-Star Galaxy Pub (2002).djvu

\*/

#include "driver.h"

#include "monitor.h"

#include "full\_adder.h"

int sc\_main(int ar9c, char\* ar9v[]) {

sc\_signal<bool> t\_a, t\_b, t\_cin, t\_sum, t\_cout;

full\_adder f1 ("FullAdderWithHalfAdder");

// Connect using positional association:

f1 << t\_a << t\_b << t\_cin << t\_sum << t\_cout;

driver d1 ( "GenerateWaveforms" );

// Connect using named association:

d1.d\_a (t\_a);

d1.d\_b (t\_b);

d1.d\_cin (t\_cin);

monitor mo1 ("MonitorWaveforms");

mo1 << t\_a << t\_b << t\_cin << t\_sum << t\_cout;

sc\_start(100, SC\_NS);

return(0);

}/\*end main()\*/

Para simular cualquier modelo SystemC, primero se escribe un banco de pruebas como una función llamada sc\_main(). La función toma dos argumentos: argc, la cuenta del número de argumentos de la línea de comandos, y argv, un arreglo que contiene los argumentos.

El módulo full\_adder se pone a prueba escribiendo un módulo driver y un módulo monitor. El módulo driver genera patrones patrones de entrada, uno cada 5ns. El módulo monitor despliega el valor de todos los puertos de full\_adder cada vez que alguno de ellos cambia de valor.

Consideremos primero el módulo driver. El módulo driver tiene tres puertos de salida y no tiene puertos de entrada, El bloque SC\_CTOR para este módulo declara un proceso de tipo SC\_THREAD. Un proceso SC\_THREAD tiene la capacidad de suspenderse a símismo debido a sentencias wait. Una sentencia wait puede esperar por algún tiempo o esperar a que ocurran ciertos eventos, o podría esperar por una combinación de ambas cosas. El proceso SC\_THREAD prc\_driver define una variable local pattern la cual es un entero sin signo de tres bits. El ciclo while itera, asigna cada patrón a los puertos de salida y durante cada iteración el patrón es incrementado. pattern es un ejemplo de una variable, en contraste con una señal o un puerto, y esta (la variable) o exibe el comportamiento retardo delta. En otras palabras, la asignación a una variable ocurre instantáneamente.

SystemC permite hacer selección de bit en una variable de tipo sc\_uint a través del operador [ ]. Por ejemplo, pattern[0] se refiere al bit en la posición 0 del entero sin signo pattern. Las tres sentencias de asignación causan que el bit en la posición 0 de pattern sea asignado al puerto d\_a, el bit en la posición 1de pattern sea asignado al puerto d\_b, y que el bit en la posición 2 de pattern sea asignado al puerto d\_cin. La penúltima sentencia en el ciclo while causa que el proceso thread se suspenda por 5ns. SC\_NS es la unidad de medida de tiempo para el parámetro 5.

El módulo monitor tiene solamente dos puertos de entrada. Este monitorea todas las entradas y salidas de la instancia del módulo full\_adder. Este es modelado usando un proceso SC\_METHOD tal que cada vez que hay un cambio en los valores de sus puertos de entrada, se llama al proceso prc\_monitor para imprimir los valores de todas las entradas de ese módulo. La funció predefinida de SystemC sc\_time\_stamp() devuelve el tiempo de simulación actual.

La función sc\_main() es donde todos los componentes del banco de pruebas son puestos juntos. En el archivo donde está la función sc\_main() también se incluyen todos los archivos de cabecera relevantes para los módulos que están siendo puestos a prueba. En nuestro caso, estos son full\_adder.h, monitor.h, y driver.h. Hay algunas señales locales tales como t\_a y t\_b que son declaradas dentro de la función sc\_main() como de tipo bool. Estas señales son usadas para interconectar el driver, el monitor, el diseño bajo prueba (el módulo full\_adder), y el banco de pruebas.

La línea:

full\_adder f1 ("FullAdderWithHalfAdder");

instancia el módulo full\_adder en el banco de pruebas. Note que en sc\_main() hemos usado un mecanismo de instanciación diferente del que usamos en SC\_MODULE. El nombre de la instancia es f1. Además, esta instancia es asociada con señales usando asociación posicional (también se habría podido usar asociación por nombre). La línea:

f1 << t\_a << t\_b << t\_cin << t\_sum << t\_cout;

asocia los puertos del full\_adder con las señales usando asociación posicional. El primer puerto del módulo full\_adder, a, es conectado a t\_a, el segundo puerto de full\_adder, b, es concetado a t\_b, y así sucesivamente. Una vez más, note que la forma en que las interconexiones están especificadas en la función sc\_main() es diferente de la forma en que las interconexiones están especificadas cuando se describe la jerarquía en un módulo. Esas diferencias en la sintaxis de especificación de jerarquía en sc\_main() y dentro de una jerarquía de módulo es causada por el hecho de que dentro de sc\_main(), un módulo se declara como un objeto (f1 es un objeto de tipo full\_adder), mientras que en una jerarquía de módulo, se declara un apuntador por cada instancia del módulo y por lo tanto cuando se usa tiene que ser derreferenciado.

Las instanciaciones de driver y de monitor aparecen después de la instanciación de full\_adder. Los puertos de la instancia de driver son asociados usando asociación nombrada.

d1.d\_a (t\_a);

d1.d\_b (t\_b);

d1.d\_cin (t\_cin);

El puerto d\_a de driver es conectado a la señal t\_a, el puerto d\_b de driver es conectado a la señal t\_b, y así sucesivamente. La instancia monitor es interconectada usando asociación posicional.

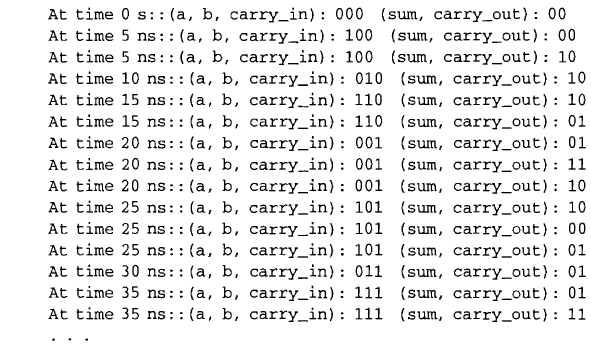
La sentencia:

sc\_start(100, SC\_NS);

comienza la simulación y corre por 100ns.

Para simular el banco de pruebas se debe crear un simulador ejecutable. El apéndice A de [Bashker] da detalles sobre cómo conseguir esto. Se puede crear un archivo Makefile y especificar los siguientes archivos fuente: half\_adder.cpp, full\_adder.cpp, driver.cpp, monitor.cpp, full\_adder\_main.cpp. Después de construir el ejecutable, el banco de pruebas se ejecuta invocando el ejecutable.

La salida producida ejecutando el banco de pruebas es:



Note que la salida presenta algunas líneas consecutivas con el mismo tiempo. Esto es porque estamos imprimiendo valores cada vez que hay un cambio de valor en algún puerto de entrada o de salida. Tales cambios de valores pueden ocurrir después de uno o más retardos delta en un tiempo de simulación particular.

Si usted quiere que los valores sean impresos cuando algún puerto de salida cambia de valor, entonces cambie la lista de sensibilidad en el proceso monitor a:

sensitive << m\_sum << m\_cout;

En el capítulo 8 de [Bashker] se muestra como escribir archivos VCD, leer y escribir patrones desde y hacia archivos de texto, y generación de reloj.

REF.

J. Bashker (2002). A SystemC Primer (1/a edición). Star Galaxy Publishing.

J. Bhasker - A SystemC Primer-Star Galaxy Pub (2002).djvu