Implementación de componentes de un procesador SPARC en modelos atomicos utilizando CD++

Autores:
Cremona, Pablo L. e-mail: pcremona@dc.uba.ar
Sor, Pablo S. e-mail: psor@dc.uba.ar

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Director:
Dr. Gabriel Wainer e-mail: gabrielw@dc.uba.ar

Materia:
Organización del Computador I – 2° año, 1° cuatrimestre

Junio 2000

Resumen:
La herramienta CD++ sigue el formalismo DEVS de simulación de eventos discretos. Con ella se desarrollan varios componentes de un procesador SPARC utilizando lógica digital.
El objetivo es armar una biblioteca para que, juntos a otros componentes, puedan ser integrados para simular la arquitectura y funcionamiento del procesador completo. En particular en este trabajo se trató de estandarizar la parametrización de los componentes, así también como esquema general en la programación.
En este informe se hace una introducción a DEVS, continuando con la descripción, desarrollo, e implementación de los componentes, algunos de los resultados obtenidos en las pruebas, y conclusiones.

Palabras Clave: sistemas de eventos discretos, DEVS, organización de computadoras, CD++, lógica digital

INTRODUCCIÓN

DEVS (Discrete EVents Systems specification) es un formalismo propuesto por Zeigler [Zei76] para el modelado de sistemas de eventos discretos. La característica de éstos para poder ser tratados mediante este formalismo es que deben poder representarse por variables discretas y tiempo contínuo. El conjunto de estas variables discretas forman los estados del sistema.

DEVS permite simular un sistema mediante modelos jerárquicos. Los modelos básicos son llamados atómicos, y el conjunto de estos forman estructuras más complejas conocidas como modelos acoplados. De esta manera, puede refinarse la simulación tanto como uno quiera, permitiendo reutilización de modelos, fácil mantenimiento, menor tiempo de testeo, y mayor productividad.

Un modelo atómico DEVS se describe como:
$\mathbf{M} = < \mathbf{X}, \mathbf{S}, \mathbf{Y}, \boldsymbol{\delta}_{int}, \boldsymbol{\delta}_{ext}, \boldsymbol{\lambda}, \mathbf{D} >$

donde,
X: conjunto de eventos externos de entrada;
S: conjunto de estados secuenciales;
Y: conjunto de eventos externos generados para salida;
$\boldsymbol{\delta}_{int}$: $\mathbf{S} \rightarrow \mathbf{S}$, función de transición interna, que define los cambios de estado por causa de eventos internos;
$\boldsymbol{\delta}_{ext}$: $\mathbf{Q} \times \mathbf{X} \rightarrow \mathbf{S}$, función de transición externa, que define los cambios de estado por causa de eventos externos; donde el conjunto de eventos totales del sistema es $\mathbf{Q} = \{ (\mathbf{s}, \mathbf{e}) / \mathbf{s} \in \mathbf{S}, \mathbf{e} \in [0, \mathbf{D}(\mathbf{s})] \}$, siendo **e** el tiempo transcurrido desde la última transición de estado con **s** -es decir, la función de transición externa depende del estado actual y del tiempo transcurrido-.
$\boldsymbol{\lambda}$: $\mathbf{S} \rightarrow \mathbf{Y}$, función de salida; y
D: $\mathbf{S} \rightarrow \Re_0^+$, función de duración de un estado -**D**(**s**) es el tiempo que el modelo se queda en el estado **s** si no hay un evento externo-.
Cada modelo puede verse como con puertos de entrada/salida para comunicarse con otros modelos. Los eventos de entrada y salida determinan los valores en esos puertos. Los eventos de entrada externos se reciben por el puerto de entrada, y la especificación del modelo debe definir las propiedades bajo los mismos. Los eventos internos producen cambios de estado, cuyos resultados se envian a través de los puertos de salida.

CD++ [BBW98a] es una herramienta que implementa los conceptos teóricos para el mecanismo de simulación del paradigma DEVS. Utilizando C++, los modelos atómicos se programan e incorporan en la jerarquía de clases. Un lenguaje de especificación permite definir el acoplamiento de los modelos, valores iniciales y los eventos externos de entrada. Al correr la simulación, la herramienta envía a la salida el desarrollo, eventos, y transiciones de la misma. Cuatro tipos de mensajes son generados:
*****: indica un cambio de estado a causa de un evento interno;
X: se produce la llegada de un evento externo -incluye port y valor-;
Y: salida del modelo;
done: indica que el modelo finalizó con su tarea.
Las funciones que deben tenerse en cuenta para programar un modelo son:
externalFunction: (función de transición externa) es activada cuando el valor de alguno de los puertos de entrada cambia. En esta función se programa la reacción del modelo a cambios externos.
outputFunction: (función de salida) aquí se programan las salidas que debe emitir el modelo.
internalFunction: (función de transición interna) aquí se programan la reacción del modelo a cambios internos.

DESCRIPCIÓN DE LOS COMPONENTES A SIMULAR

El diseño de la arquitectura del modelo a simular se basa en la especificación del *Integer Unit* del procesador SPARC de Sun Microsystems. Se trata de la misma arquitectura RISC, a la que se le han simplificado el set de instrucciones y el manejo de la memoria.
Para el desarrollo de los modelos utilizamos CD++ y C++, compilando con GCC 2.8.1.
Los modelos utilizan un tiempo configurable de estabilización de los registros. Es decir, que luego de recibida una entrada, el modelo tarda el tiempo de estabilización en retornar a un estado correcto; emitiendo las salidas necesarias y realizando las acciones pertinentes.

Por último, todos los componentes están inicializados en cero, y están armados de tal forma que el primer evento externo genere una salida cuando exista un cambio de fase.

ALIGNS

Debido a que el bus de direcciones no utiliza los dos bits mas bajos pero la memoria es direccionable a byte, se debe alinear el dato a transmitir por el bus de datos, según su tamaño (byte, half word o word) y los dos bits menos significativos de la dirección (A_1A_0).

Los puertos de entrada y salida del modelo son:

Línea	In	Out	Descripción
OP0..31	X		Operando
SIZE0..1	X		00:Word, 01:Byte 10: Half Word
A0..1	X		Bits más bajos del address
RES0..31		X	Resultado alineado

Comportamiento:

Si Size = Word, Res = Op
Si Size = Half Word, Res = Op << (16 * A_1)
Si Size = Byte, Res = Op << (8 * A_1A_0)

El modelo posee un único parámetro, **response**, que setea el tiempo te transición interna. Este parámetro se repite en todos los modelos que se detallan en este informe. Todos los parametros de todos los modelos descriptos son opcionales, y de no ingresarse los modelos se configuran con valores por defecto.
Para la representación interna del operando OP utilizamos un arreglo de bit (clase usada para representar todos los bits de entrada en todos los atómicos de este informe), y para RES un arreglo o_bits (esta clase en encarga de controlar el valor anterior de un puerto para saber cuando corresponde la generación de un mensaje por parte de un port de salida y al igual que bit, es usada en todos los atómicos, pero para representar los puertos de Out).

La función de transición externa, al igual que en el resto de los atómicos, se encarga de actualizar las variables internas para mantener el valor de los mensajes recibidos y de realizar el **holdIn** según el tiempo configurado y a menos que algún modelo cambie este comportamiento, no se repetirá esta explicación.

La funcion de salida realiza el alineamiento y por medio de la calse o_bit se encarga de decidir si es necesario emitir algún mensaje por los cambios en los puertos de salida.

ALU

Es la Unidad Aritmetico-Lógica y se encarga de realizar: sumas (con y sin carry in), restas (también con y sin carry in), y operaciones logicas (AND, OR, XOR, NAND, NOR y NXOR). Realiza operaciones sin signo, con complemento a dos. Posee dos operandos de entrada de 32bits, y uno de salida, además de bits para el codigo de operación, carry in y salida de flags.

Los puertos de entrada y salida del modelo son:

Línea	In	Out	Descripción
OPA0..31	X		Op A
OPB0..1	X		Op B
CIN	X		Carry in
FCOD0..3	X		Function Code
RES0..31		X	Result
C		X	Carry
N		X	Negative
Z		X	Zero
V		X	Overflow

Las operaciones lógicas dejan C y O en 0

Fcod	Operación	Res
0000	ADD	OpA + OpB
0001	AND	OpA $\wedge$ OpB
0010	OR	OpA $\vee$ OpB
0011	XOR	OpA XOR OpB
0100	SUB	OpA – OpB
0101	ANDN	~(OpA $\wedge$ OpB)
0110	ORN	~(OpA $\vee$ OpB)
0111	XNOR	~(OpA XOR OpB)
1000	ADDX	OpA + OpB + Cin
1100	SUBX	OpA – OpB – Cin

Para las operaciones convertimos los puertos de entrada en enteros y realizamos los cálculos entre enteros (long) guardando los resultados en enteros de 64 bits (long long) provistos por el C++. Luego calculamos los flags según corresponda.

CCLOGIC

Este componente se utiliza para determinar, evaluando los flags, si es preciso bifurcar o no en un salto condicional.

Los puertos del modelo son:

Nombre	In	Out	Descripción
C	X		Carry
N	X		Negative
Z	X		Zero
V	X		Overflow
COND0..3	X		Cond
RES		X	Result

El estado de los ports de entrada se representó mediante obejtos bit, y RES mediante o_bit. La función de transición externa se encarga de recibir los mensajes y actualizar los valores internos. Luego, en la funcion de salida, se calcula el valor correspondiente de RES, según se detalla en la tabla.

Cond	B...	Descripción	Res
0000	N	Never	0
0001	E	Equal	Z
0010	LE	Less or Equal	Z ∨ (N XOR V)
0011	L	Less	N XOR V
0100	LEU	L. or Eq. Uns.	C ∨ Z
0101	CS	Carry Set	C
0110	NEG	Negative	N
0111	VS	Overflow Set	V
1000	A	Always	1
1001	EN	Not Equal	~Z
1010	G	Greater	~ (Z ∨ (N XOR V))
1011	GE	Greater or Equal	~(N XOR V)
1100	GU	Greater Unsigned	~(C ∨ Z)
1101	CC	Carry Clear	~C
1110	POS	Positive	~N
1111	VC	Overflow Clear	~V

CLOCK

Este componente se utiliza como mecanismo de sincronización. El período del reloj se configura mediante el parametro **response** (en realidad este parametro indica la duración de medio período, ya que marca el tiempo que transcurre desde que se envía un bit, hasta que se envía el siguiente con el valor inverso).

El modelo solo posee un puerto:

Nombre	In	Out	Descripción
CLCK		X	Clock

Ya que no posee puertos de entrada, tampoco se codificó una función de transición externa. La funcion de transición interna es la encargada de realizar el holdIn para esperar el tiempo correspondiente, según se especificó en el parámetro **response**. La función de salida simplemente niega el ultimo valor del clock, y envía el mensaje.

CMP

Este componete es utilizado por la unidad de direccionamiento, para verificar que las direcciones no sobrepasen el registro limite. Posee dos registros de entrada de 32 bits, que son los que se comparan, y dos bits de salida, indicando si el resultado de la comparación fue igual, menor o mayor
Los puertes de entrada y salida son:

Nombre	In	Out	Descripción
OPA0..31	X		Op A
OPB0..31	X		Op B
EQ		X	Equal
LW/GT		X	lower/not greater

Los registros de entrada los representamos mediante dos arreglos de la clase bit y los dos puertos de salida mediante o_bit. La función de salida compara los dos arreglos que represetan a los operandos de entrada bit a bit.

CS

Se utiliza para determinar si una direccion de memoria se encuentra dentro de un determinado rango, para poder decidir que modulo de memoria debe contestar el pedido. Posee dos operandos de entrada, uno con la direccion, de 32bits, y el otro de un solo bit, indicando se la direccion es valida. Posee solo un bit de salida.

Los puertos de entrada y salida son:

Nombre	In	Out	Descripción
A0..31	X		Address
AS	X		Address Strobe
CS		X	Chip Select

Además del parámetro **response**, con el que se indica el tiempo de respuesta, posee otros dos parámetros más, **bottom**, y **top**, con los que se indica el rango entre el cual debe estar el address.

CWPLOGIC

La arquitectura tiene 520 registros de uso general, de los cuales los primeros 8 son uso global, y el resto son organizados en ventanas que se superponen. En un instante dado, sólo 32 registros "locales" están disponibles. El esquema se implementa con un arreglo de 8 registros que contiene los registros globales, y otro de 512, donde una ventana de 24 registros aloja los registros de salida, los locales y los de entrada. CWP es la variable que apunta la ventana activa. CWPLogic es el componente que se utiliza para determinar si se intenta acceder a una ventana de registros o a los registros globales. Posee como entrada la ubicación de la ventanta actual (CWP) y el registro buscado, y como salida el numero de registro dentro de la ventanta o global, según corresponda, y un bit indicando cual de estas dos salidas es la valida.
Los puertos de entrada y salida son:

Nombre	In	Out	Descripción
CWP0..4	X		Current Window Pointer
SEL0..3	X		Select
GSEL0..2		X	Global Select
RSEL0..8		X	Register Select
R/G		X	Register/Global

El valor de los puertos de salida es:

Si Sel < 8, GSel = Sel y R = 0

Si no, RSel = (CWP * 16 + Sel - 8) % 512 y R = 1

Para realizar la función de salida, los valores de los puertos se convierten a enteros, y luego se realizan los cálculos como se indica anteriormente. El determinar si un puerto de salida requiere un nuevo mensaje o no, como en el resto de los modelos, se deja en manos de la clase o_bit.

INC4

El procesador trabaja con dos program counters: PC (Program Counter), que contiene la dirección de la próxima instrucción a utilizar y nPC (Next Program Counter), que conserva el próximo valor del PC.
Cada ciclo de instrucción normal termina copiando el nPC al PC y sumándole 4 al nPC. Esto se realiza mediante el componente Inc4

Los puertos de entrada y salida son:

Nombre	In	Out	Descripción
OP0..31	X		Op
RES0..31		X	Result = Op + 4

INC/DEC

Inc/Dec es el componente que se utiliza para desplazar la ventana de registros, incrementando o decrementando el valor de CWP. Éste último es de 5 bits. Opera en forma circular.
Los puertos de entrada y salida del modelo son:

Línea	In	Out	Descripción
OP4..0	X		Operando
FCOD	X		1 = INC; 0 = DEC

RES4..0	X	Resultado

Para la representación interna del operando OP utilizamos un arreglo de bit, y para RES un arreglo o_bits.

IRQLOGIC

El bus de control posee quince lineas de IRQ (IRQ1-IRQ15), a su vez, el procesador, posee en su palabra de estado, 4 bits para guardar el Processore Interrupt Level (PIL), que indica cual es nivel minimo de interrupcion que se debe atender. Este componente se utiliza para determinar si un IRQ debe ser atendido o no (de acuerdo al valor del PIL), y en caso de existir varios IRQs simultáneos, cual será atendido primero.

Los puertos de entrada salida son:

Nombre	In	Out	Descripción	Prioridad	Trap Type
IRQ1..5	X		Interrupt request	32-IRq	0x11-0x1F
PIL0..3	X		Processor interrupt level		
TF		X	Trap found		
TT0..7		X	Trap type		

Si alguna IRQ > PIL está encendida, se selecciona la de mayor prioridad para enviar su Trap Type y se enciende TF. Para lograr esto, en la función de salida, se recorre el arreglo que representa a los puertos de IRQ, mientras sean mayores que el valor de PIL, hasta hallar el de mayor prioridad (encendido).

LATCH

Este componente modela una memoria que contiene un solo registro. Se utiliza principalmente para mantener los palores enviados y recibidos por el bus de datos en el integer Unit.

Los puertos de entrada salida son:

Nombre	In	Out	Descripción
IN0..31	X		
EIN	X		Enable Input
CLEAR	X		
OUT0..31		X	

Si EIN esta en 1, el valor de la entrada (IN), se copia al latch. Si Clear esta en 1, el latch se reseta y pasa a contener el valor 0. El valor del latch siempre esta reflejado en OUT.

MEMORIA

La memoria del modelo es plana, no provee mecanismos de paginación o segmentación. No soporta multiprogramación. La implementación incluye dos registros, BASE y LÍMITE, que determinan el espacio de direcciones disponible para el programa. El direccionamiento es a byte, y la información es guardada según el standard Big-Endian.

La operatoria es la siguiente: cuando el bus necesita una operación con la memoria, prepara todas las señales, datos, dirección, etc.; una vez que está todo listo, envía la señal AS, que la memoria interpreta e identifica como que la dirección que se encuentra en el bus de direcciones es valida. Toma la dirección –A2..A31-, y establece el tipo de operación, lectura o escritura, según lo recibido en RD/WR. Si es lectura, se toman los 4 bytes desde la dirección especificada y se envían por las líneas de datos; si es escritura, se graban únicamente los bytes indicados en el byte select –BSEL0..3, tomados también del bus-. Si la dirección tomada desde el bus es incorrecta, es decir, se intenta apuntar a una posición no disponible por el tamaño de la memoria real, se enciende la señal de error –ERR-. Una vez que está concluido el proceso de búsqueda, lectura/escritura, se informa de ello enviando al bus la señal de data acknowledge –DTACK-.

Los puertos de entrada y salida se muestran en la siguiente tabla:

Línea	In	Out	Descripción
DATA0..31	X	X	Data

A2..31	X		Address
BSEL0..3	X		Byte Select
AS	X		Address Strobe
RD_WR	X		Read or Write
DTACK		X	Data Acknowledge
ERR		X	Error
RESET	X		Reset
DUMP	X		Dump de memoria

Además del parametro **response**, el modelo posee otros tres parámetros: **memsize**, que indica el tamaño de la memoria, **memfile** mediante el cual se elige el mapa de memoria para la inicialización, y **dumpfile**, que indica el archivo sobre el que se realizará el volcado de memoria. El volcado se realizará al recibir un mensaje por el port DUMP. En principio se diseño el componente para que el dump se provocara al finalizar su existencia, pero debió realizarse mediante un port debido a un Bug en el CD++, que no llama los destructores de los modelos.

Se utilizaron arreglos de bit y o_bit para representar los estados de los ports de entrada y salida respectivamente. La memoria se representa con un arreglo de bytes y por lo tanto el máximo estará limitado por la memoria disponible en el sistema donde se lleve a cabo la representación. Si se indicó un valor para el parámetro **memfile**, esta se cargara con el valor de dicho archivo, y en caso de no indicarse un archivo de entrada la memoria no estará inicializada y sus valores serán indeterminados.

Si no se especifica el parametro **memsize** el valor por defecto es 32kb. Si el valor especificado es mayor que el tamaño de la imagen provista, el resto de la memoria no será inicializada, si es menor, se trunca.
Si no se indica un archivo para **dumpfile** no se producirá el vuelco.

MUL/DIV

Es la Unidad Multiplier./Divider y se encarga de realizar: multiplicaciones y divisiones (con y sin signo). Posee tres operandos de entrada de 32bits, (los dos operandos reales, más el registro auxiliar %Y) , y dos de salida (el operando e %Y), además de bits para el codigo de operación, y salida de flags.

Los puertos de entrada y salida del modelo son:

Línea	In	Out	Descripción
OPA0..31	X		Op A
OPB0..31	X		Op B
YIN0..31	X		Y In
YOUT0..31		X	Y Out
FCOD0..1	X		Function Code
RES0..31		X	Result
N		X	Negative
Z		X	Zero
V		X	Overflow
DIV_ZERO		X	Division by zero

El resultado generada es:

FCod		
00	UMUL	YOut * 2^{32} + Res = OpA × OpB (Unsigned)
01	SMUL	YOut * 2^{32} + Res = OpA × OpB (Signed)
10	UDIV	Res = YIn * 2^{32} + OpA / OpB (Unsigned)
11	SDIV	Res = YIn * 2^{32} + OpA / OpB (Signed)

Para realizar las operaciones convertimos los puertos de entrada en enteros y realizamos los calculos entre enteros (long) guardando los resultados en enteros de 64 bits (long long) provistos por el C++. Luego calculamos los flags según corresponda.

MUX

El multiplexor es un componente que, a partir de una señal de selección, toma una de las dos entradas y la envía por la salida. Varios multiplexores de este tipo se encuentran dentro de la arquitectura a simular, formando parte de las unidades aritmético-lógica y de enteros, entre otras.

Las dos entradas están formadas por datos de 32 bits cada una. Para seleccionarlas, se recibe la señal SELA/SELB, que indica cual operando será copiado en la salida.

A continuación se muestra la tabla con los puertos de entrada y salida:

Línea	In	Out	Descripción
A0..31	X		Operando A
B0..31	X		Operando B
SELA_B	X		Select A/B
OUT0..31		X	Salida

REGBLOCK

Es el bloque de registros a utilizarse en los procedimientos (sin incluir los 8 registros globales). Este componente permite almacenar o leer valores sobre cada uno de los registros existentes.
Mediante un arreglo de 512 registros de 32 bits simulamos los registros manejados por el componente.

En el siguiente cuadro pueden verse los distintos ports de entrada y salida :

Línea	In	Out	Descripción
ASEL0..8	X		Select A
BSEL0..8	X		Select B
CSEL0..8	X		Select C
CEN	X		Enable C
RESET	X		Reset
AOUT0..31		X	Output A
BOUT0..31		X	Output B
CIN0..31	X		Input C

Su compotamiento, suponiendo R el bloque de registros, es el siguiente:

AOut = R[ASel]
BOut = R[BSel]
Si CEn = 1, R[CSel] = Cin
Reset deja todos los registros en 0

REGGLOB

Es el bloque de registros globales. Este componente permite almacenar o leer valores sobre cada uno de los registros globales existentes a excepción de %g0, que siempre posee el valor 0. Mediante un arreglo de 8 registros de 32 bits simulamos los registros manejados por el componente.

En el siguiente cuadro pueden verse los distintos ports de entrada y salida :

Línea	In	Out	Descripción
ASEL0..3	X		Select A
BSEL0..3	X		Select B
CSEL0..3	X		Select C
CEN	X		Enable C
RESET	X		Reset
AOUT0..31		X	Output A
BOUT0..31		X	Output B
CIN0..31	X		Input C

Su compotamiento, suponiendo G el bloque de registros, es el siguiente:

AOut = G[ASel]
BOut = G[BSel]
Si CEn = 1 y Csel > 0, G[CSel] = Cin
Reset deja todos los registros en 0

SHIFTER

Forma parte de la ALU de la Sparc, junto con el componente Mul/Div y ALU. Se utiliza para realizar decalajes a izquierda y a derecha (con y sin signo). Para realizar este modulo convertimos la entrada a un entero, y realizamos los decalajes utilizando instrucciones de C++.

Los puertos de entrada y salida son:

Nombre	In	Out	Descripción
OPA0..31	X		Op A
OPB..4	X		Op B
FCOD0..1	X		Function Code
RES0..31		X	Result

Donde Fcod indica:

FCod	Instruc.	Res
01	SLL	OpA << OpB
10	SRL	OpA >> OpB (Unsigned)
11	SRA	OpA >> OpB (Signed)

SIGNEXTN

Este componente extiende el signo de su operando de entrada (de n bits), a 32 bits. Se utiliza para modelar el componente SignExt13 y SignExt22. El tamaño del operando es configurable mediante el parámetro **nbits**. Para su implementación simplemente se copia el bit más significativo que tenga valor uno de la entrada, en todos los bits necesarios hasta completar 32.

Los puertos de entrada y salida del modelo son:

Nombre	In	Out	Descripción
OP0..12	X		Op
RES0..31		X	Result = signextend(Op)

TRAPLOGIC

Este componente es el encargado de determinar cuál es el trap que deberá atenderse. Está basado en un sistema de prioridades.

Consta de una serie de entradas para ciertos traps distinguidos, una entrada que indica que ha ocurrido un trap "común" y 7 bits para indicar el número de trap "común". Como salida tiene un bit que indica si hay algún trap para atender y 8 bits para indicar cuál es el trap (*trap type*).

En el siguiente cuadro pueden verse los distintos ports de entrada y salida (In y Out), prioridades y tipos de los distintos traps existentes:

Línea	In	Out	Descripción	Prioridad	Trap Type
INST_ACC_EXCEP	X		Instruction access exception	5	0x01
ILLEG_INST	X		Illegal instruction	7	0x02
PRIV_INST	X		Privileged instruction	6	0x03
WIN_OVER	X		Window overflow	9	0x05
WIN_UNDER	X		Window underflow	9	0x06
ADDR_NOT_ALIGN	X		Address not aligned	10	0x07
DATA_ACC_EXCEP	X		Data access exception	13	0x09
INST_ACC_ERR	X		Instruction access error	3	0x21
DATA_ACC_ERR	X		Data access error	12	0x29
DIV_ZERO	X		Division by zero	15	0x2A
DATA_ST_ERR	X		Data store error	2	0x2B
TRAP_INST	X		Trap instruction	16	0x80+TN
TN6.. 0	X		Trap number		
TF		X	Trap found		
TT7.. 0		X	Trap type		

De acuerdo al cuadro anterior, el componente que simulamos registra cuál es el trap con mayor prioridad que debe ser atendido y luego del tiempo de preparación emite por los puertos de salida el índice correspondiente.

WIMCHECK

Es el componente de hardware que se encarga de hacer el chequeo que indica si la próxima ventana de registros a utilizar sobreescribirá a una que ya está utilizada. Para ello recibe los registros *WIM* y *CWP* y devuelve el valor del CWP-ésimo bit del *WIM*. Si el bit vale 1, el CWP está apuntando a la ventana más antigua que está siendo utilizada.

Este componente consta de un registro para almacenar los valores que recibe por los ports WIM (32 bits) y otro para los CWP (5 bits).

En el siguiente cuadro pueden verse los distintos ports de entrada y salida (In y Out) existentes:

Línea	In	Out	Descripción
CWP0..4	X		Current Window Pointer
WIM0..31	X		Window Invalid Mask
RES		X	Res = WIM_{CWP}

RESULTADOS

Con el objetivo de comprobar el funcionamiento, incluimos a continuación, los resultados varias simulaciones. La programación de los modelos que poseen puertos de mas un bit, como se detalla mas arriba, se resolvió convirtiendo estos los valores a números entornos y luego se procesaron dichas entradas como enteros y solo se transformaron nuevamente al finalizar los cálculos, por lo tanto, el funcionamiento de los modelos debe ser similar para cualquier valor de entrada de estos puertos. Testeamos entonces casos con valores "comunes" y casos podrían parecer particulares por el comportamiento esperado del modelo. En el mul/div incluimos un test donde se le envía al modelo un mensaje no esperado (un valor tres en cambio de un uno o cero), el modelo se comporta como si hubiera recibido un uno, ya que la clase bit realiza un cast del valor de entrada a bool y en C++ todo valor distinto a cero se castea a uno en bool, en realidad a true [SB00]) También a modo de ejemplo incluimos un test de varios modelos acoplados que representan el bloque de registros de uso general descripto en [DDTZW98].

El primero es el componente AlignS, configurado con un tiempo de respuesta de 00:00:20:00. Cargamos en OP 0x0000FF00 en SIZE Half word y en A_1A_0 0x01, obtenemos en la salida 0x00FF0000. Luego ponemos el SIZE en Byte y en la salida obtenemos 0x0000FF00, finalmente seteamos el SIZE en Word y la salida no se modifica.

aligns.ev
00:00:00:08 OP8 1
00:00:00:09 OP9 1
00:00:00:10 OP10 1
00:00:00:11 OP11 1
00:00:00:12 OP12 1
00:00:00:13 OP13 1
00:00:00:14 OP14 1
00:00:00:15 OP15 1
00:00:01:00 SIZE0 0
00:00:01:01 SIZE1 1
00:00:02:00 A0 1
00:00:02:01 A1 0
00:01:01:00 SIZE0 1
00:01:01:01 SIZE1 0
00:03:01:00 SIZE0 0
00:03:01:01 SIZE1 0

aligns.out
00:00:22:001 res23 1
00:00:22:001 res22 1
00:00:22:001 res21 1
00:00:22:001 res20 1
00:00:22:001 res19 1
00:00:22:001 res18 1
00:00:22:001 res17 1
00:00:22:001 res16 1
00:01:21:001 res23 0
00:01:21:001 res22 0
00:01:21:001 res21 0
00:01:21:001 res20 0
00:01:21:001 res19 0
00:01:21:001 res18 0
00:01:21:001 res17 0
00:01:21:001 res16 0

```
00:01:21:001 res15 1
00:01:21:001 res14 1
00:01:21:001 res13 1
00:01:21:001 res12 1
00:01:21:001 res11 1
00:01:21:001 res10 1
00:01:21:001 res9 1
00:01:21:001 res8 1
```

Luego probaremos una entrada del tamaño indica en los bits de SIZE, por ejemplo 0x00FFFF00 con un size Half Word. Y la alineación en A_0 en uno. Obtenemos como resultado 0xFFFF0000. Probando con un byte únicamente (0x00FF0000) pero alineado con A_1A_0 igual a 2, el modelo realiza un decalaje y obtenemos como resultado 0x00.

```
Aligns1.ev
00:00:00:10 OP8  1
00:00:00:10 OP9  1
00:00:00:10 OP10 1
00:00:00:10 OP11 1
00:00:00:10 OP12 1
00:00:00:10 OP13 1
00:00:00:10 OP14 1
00:00:00:10 OP15 1
00:00:00:10 OP16 1
00:00:00:10 OP17 1
00:00:00:10 OP18 1
00:00:00:10 OP19 1
00:00:00:10 OP20 1
00:00:00:10 OP21 1
00:00:00:10 OP22 1
00:00:00:10 OP23 1
00:00:00:10 A0 1
00:00:00:10 SIZE0 0
00:00:00:10 SIZE1 1

00:01:00:00 OP8  0
00:01:00:00 OP9  0
00:01:00:00 OP10 0
00:01:00:00 OP11 0
00:01:00:00 OP12 0
00:01:00:00 OP13 0
00:01:00:00 OP14 0
00:01:00:00 OP15 0
00:01:00:00 A1 1
00:01:01:00 SIZE0 1
00:01:01:01 SIZE1 0


aligns1.out
00:00:20:010 res31 1
00:00:20:010 res30 1
00:00:20:010 res29 1
00:00:20:010 res28 1
00:00:20:010 res27 1
00:00:20:010 res26 1
00:00:20:010 res25 1
00:00:20:010 res24 1
00:00:20:010 res23 1
00:00:20:010 res22 1
00:00:20:010 res21 1
00:00:20:010 res20 1
00:00:20:010 res19 1
00:00:20:010 res18 1
00:00:20:010 res17 1
00:00:20:010 res16 1
```

```
00:01:21:001 res31 0
00:01:21:001 res30 0
00:01:21:001 res29 0
00:01:21:001 res28 0
00:01:21:001 res27 0
00:01:21:001 res26 0
00:01:21:001 res25 0
00:01:21:001 res24 0
00:01:21:001 res23 0
00:01:21:001 res22 0
00:01:21:001 res21 0
00:01:21:001 res20 0
00:01:21:001 res19 0
00:01:21:001 res18 0
00:01:21:001 res17 0
00:01:21:001 res16 0
```

El CCLogic se encuentra configurado con un tiempo de respuesta de 00:00:10:00, incluimos el resultado de todos los comandos para dos combinaciones de flags distintas. Primero prendemos todos los bits de flags en uno(hay que aclarar que esta es una combinación no posible, ya que el bit de zero no puede prenderse en simultaneo con el de negativo). Probamos en orden todos los comandos uno a uno, y como se observa los resultados son los esperados según el comportamiento descripto mas arriba. Luego seteamos los flags de carry y overflow en uno y los demás en cero. El resultado obtenido aquí, también es el correcto. En este test se ve claramente el funcionamiento de clase o_bit para manejar los puertos de salida y no enviar dos veces el mismo mensaje, ya que este modelo posee un solo bit de out, vemos en la salida que los valores reportados son siempre distintos al inmediato anterior.

```
cclogic.ev
00:00:10:00 C 1
00:00:10:00 N 1
00:00:10:00 Z 1
00:00:10:00 V 1

00:01:00:00 COND0 1
00:01:00:00 COND1 0
00:01:00:00 COND2 0
00:01:00:00 COND3 0

00:02:00:00 COND0 0
00:02:00:00 COND1 1
00:02:00:00 COND2 0
00:02:00:00 COND3 0

00:03:00:00 COND0 1
00:03:00:00 COND1 1
00:03:00:00 COND2 0
00:03:00:00 COND3 0

00:04:00:00 COND0 0
00:04:00:00 COND1 0
00:04:00:00 COND2 1
00:04:00:00 COND3 0

00:05:00:00 COND0 1
00:05:00:00 COND1 0
00:05:00:00 COND2 1
00:05:00:00 COND3 0

00:06:00:00 COND0 0
00:06:00:00 COND1 1
00:06:00:00 COND2 1
00:06:00:00 COND3 0
```

00:07:00:00 COND0 1
00:07:00:00 COND1 1
00:07:00:00 COND2 1
00:07:00:00 COND3 0

00:08:00:00 COND0 0
00:08:00:00 COND1 0
00:08:00:00 COND2 0
00:08:00:00 COND3 1

00:09:00:00 COND0 1
00:09:00:00 COND1 0
00:09:00:00 COND2 0
00:09:00:00 COND3 1

00:10:00:00 COND0 0
00:10:00:00 COND1 1
00:10:00:00 COND2 0
00:10:00:00 COND3 1

00:11:00:00 COND0 1
00:11:00:00 COND1 1
00:11:00:00 COND2 0
00:11:00:00 COND3 1

00:12:00:00 COND0 0
00:12:00:00 COND1 0
00:12:00:00 COND2 1
00:12:00:00 COND3 1

00:13:00:00 COND0 1
00:13:00:00 COND1 0
00:13:00:00 COND2 1
00:13:00:00 COND3 1

00:14:00:00 COND0 0
00:14:00:00 COND1 1
00:14:00:00 COND2 1
00:14:00:00 COND3 1

00:15:00:00 COND0 1
00:15:00:00 COND1 1
00:15:00:00 COND2 1
00:15:00:00 COND3 1

01:00:10:00 C 1
01:00:10:00 N 0
01:00:10:00 Z 0
01:00:10:00 V 1

01:01:00:00 COND0 1
01:01:00:00 COND1 0
01:01:00:00 COND2 0
01:01:00:00 COND3 0

01:02:00:00 COND0 0
01:02:00:00 COND1 1
01:02:00:00 COND2 0
01:02:00:00 COND3 0

01:03:00:00 COND0 1
01:03:00:00 COND1 1
01:03:00:00 COND2 0
01:03:00:00 COND3 0

01:04:00:00 COND0 0

01:04:00:00 COND1 0
01:04:00:00 COND2 1
01:04:00:00 COND3 0

01:05:00:00 COND0 1
01:05:00:00 COND1 0
01:05:00:00 COND2 1
01:05:00:00 COND3 0

01:06:00:00 COND0 0
01:06:00:00 COND1 1
01:06:00:00 COND2 1
01:06:00:00 COND3 0

01:07:00:00 COND0 1
01:07:00:00 COND1 1
01:07:00:00 COND2 1
01:07:00:00 COND3 0

01:08:00:00 COND0 0
01:08:00:00 COND1 0
01:08:00:00 COND2 0
01:08:00:00 COND3 1

01:09:00:00 COND0 1
01:09:00:00 COND1 0
01:09:00:00 COND2 0
01:09:00:00 COND3 1

01:10:00:00 COND0 0
01:10:00:00 COND1 1
01:10:00:00 COND2 0
01:10:00:00 COND3 1

01:11:00:00 COND0 1
01:11:00:00 COND1 1
01:11:00:00 COND2 0
01:11:00:00 COND3 1

01:12:00:00 COND0 0
01:12:00:00 COND1 0
01:12:00:00 COND2 1
01:12:00:00 COND3 1

01:13:00:00 COND0 1
01:13:00:00 COND1 0
01:13:00:00 COND2 1
01:13:00:00 COND3 1

01:14:00:00 COND0 0
01:14:00:00 COND1 1
01:14:00:00 COND2 1
01:14:00:00 COND3 1

01:15:00:00 COND0 1
01:15:00:00 COND1 1
01:15:00:00 COND2 1
01:15:00:00 COND3 1

cclogic.out
00:01:10:000 res 1
00:03:10:000 res 0
00:04:10:000 res 1
00:09:10:000 res 0
00:11:10:000 res 1
00:12:10:000 res 0

01:02:10:000 res 1
01:06:10:000 res 0
01:07:10:000 res 1
01:10:10:000 res 0
01:14:10:000 res 1
01:15:10:000 res 0

El Clock no posee entrada, lo mostramos aquí con un tiempo de respuesta (medio periodo) de 0:0:0:100. Este componente, antes de ser modificado, no funcionaba correctamente, ya que solamente enviaba el valor uno a intervalos regulares. Este comportamiento era doblemente erróneo, perímetro, por enviar por la salida la secuencia correcta (ceros y unos alternadamente) y segundo porque repetía mensajes sin que hubiera cambio en el valor del puerto.

clock.out
00:00:00:000 clck 1
00:00:00:100 clck 0
00:00:00:200 clck 1
00:00:00:300 clck 0
00:00:00:400 clck 1
00:00:00:500 clck 0
00:00:00:600 clck 1

El Cmp se encuentra configurado con un tiempo de 00:00:20:00. Primero seteamos 0xFFFFFFFF en ambos ports de entrada, y obtenemos, luego del tiempo de respuesta seteado, que eq se vuelve uno. A continuación seteamos 0xFFFFFFDF en el registro OPA y obtenemos en la salida eq igual a cero y lw_gt en uno ya que OPA es menor que OPB, finalmente invertimos el valor de ambos registros (OPA = 0xFFFFFFFF y OPB = 0xFFFFFFDF) y el bit de lw_gt se apaga pero eq conserva su valor (no hay mensaje).

cmp.ev
00:00:10:00 OPA0 1
00:00:10:00 OPA1 1
00:00:10:00 OPA2 1
00:00:10:00 OPA3 1
00:00:10:00 OPA4 1
00:00:10:00 OPA5 1
00:00:10:00 OPA6 1
00:00:10:00 OPA7 1
00:00:10:00 OPA8 1
00:00:10:00 OPA9 1
00:00:10:00 OPA10 1
00:00:10:00 OPA11 1
00:00:10:00 OPA12 1
00:00:10:00 OPA13 1
00:00:10:00 OPA14 1
00:00:10:00 OPA15 1
00:00:10:00 OPA16 1
00:00:10:00 OPA17 1
00:00:10:00 OPA18 1
00:00:10:00 OPA19 1
00:00:10:00 OPA20 1
00:00:10:00 OPA21 1
00:00:10:00 OPA22 1
00:00:10:00 OPA23 1
00:00:10:00 OPA24 1
00:00:10:00 OPA25 1
00:00:10:00 OPA26 1
00:00:10:00 OPA27 1
00:00:10:00 OPA28 1
00:00:10:00 OPA29 1
00:00:10:00 OPA30 1
00:00:10:00 OPA31 1
00:00:10:00 OPB0 1
00:00:10:00 OPB1 1

00:00:10:00 OPB2 1
00:00:10:00 OPB3 1
00:00:10:00 OPB4 1
00:00:10:00 OPB5 1
00:00:10:00 OPB6 1
00:00:10:00 OPB7 1
00:00:10:00 OPB8 1
00:00:10:00 OPB9 1
00:00:10:00 OPB10 1
00:00:10:00 OPB11 1
00:00:10:00 OPB12 1
00:00:10:00 OPB13 1
00:00:10:00 OPB14 1
00:00:10:00 OPB15 1
00:00:10:00 OPB16 1
00:00:10:00 OPB17 1
00:00:10:00 OPB18 1
00:00:10:00 OPB19 1
00:00:10:00 OPB20 1
00:00:10:00 OPB21 1
00:00:10:00 OPB22 1
00:00:10:00 OPB23 1
00:00:10:00 OPB24 1
00:00:10:00 OPB25 1
00:00:10:00 OPB26 1
00:00:10:00 OPB27 1
00:00:10:00 OPB28 1
00:00:10:00 OPB29 1
00:00:10:00 OPB30 1
00:00:10:00 OPB31 1
00:00:50:300 OPA5 0
00:01:40:300 OPA5 1
00:01:40:300 OPB5 0

cmp.out
00:00:30:000 eq 1
00:01:10:300 eq 0
00:01:10:300 lw_gt 1
00:02:00:300 lw_gt 0

El CS se encuentra configurado con un tiempo de 00:00:05:00 y un intervalo de [0x0001000, 0x0FFFFFFF]. En primer termino habilitamos la entrada (AS en uno) y seteamos la dirección 0x0FFFFFFF, ya que se encuentra dentro del rango obtenemos un uno en la salida. Luego probamos la dirección 0xFFFFFFFF y la salida se vuelve cero ya que la entrada es mayor que el rango.

Cs.ev
00:00:10:00 AS 1
00:00:10:00 A0 1
00:00:10:00 A1 1
00:00:10:00 A2 1
00:00:10:00 A3 1
00:00:10:00 A4 1
00:00:10:00 A5 1
00:00:10:00 A6 1
00:00:10:00 A7 1
00:00:10:00 A8 1
00:00:10:00 A9 1
00:00:10:00 A10 1
00:00:10:00 A11 1
00:00:10:00 A12 1
00:00:10:00 A13 1
00:00:10:00 A14 1

00:00:10:00 A15 1
00:00:10:00 A16 1
00:00:10:00 A17 1
00:00:10:00 A18 1
00:00:10:00 A19 1
00:00:10:00 A20 1
00:00:10:00 A21 1
00:00:10:00 A22 1
00:00:10:00 A23 1
00:00:10:00 A24 1
00:00:10:00 A25 1
00:00:10:00 A26 1
00:00:10:00 A27 1
00:00:00:00 AS 0
00:00:15:00 AS 1
00:00:10:00 A28 1
00:00:10:00 A29 1
00:00:10:00 A30 1
00:00:10:00 A31 1

cs.out
00:00:35:000 cs 1
00:01:05:000 cs 0

Repetimos el test con la misma entrada pero esta vez configuramos el modelo con el intervalo inverso (con bottom en 0x0FFFFFFF y top en 0x0001000) esta vez no obtenemos salida alguna ya que para ningún valor que la entrada sea mayor o igual bottom y menor o igual que top.

El test no produce resultados

El componente CWPLogic se encuentra configurado con un tiempo de 00:00:05:00 Primero seteamos 0x07 en CWP y 0x03 en Sel, como es menor que 8, obtenemos GSel = Sel y R/G en cero. A los 20 minutos seteamos Sel en 0x0B por lo tanto GSel no cambia su valor pero Rsel pasa a ser CWP * 16 – SEL – 8) % 512 ((7 * 16 + 11 – 8) % 512) igual a 0x73) y R/G uno.

cwplogic.ev
00:00:00:10 CWP0 1
00:00:00:10 CWP1 1
00:00:00:10 CWP2 1
00:00:00:10 SEL0 1
00:00:00:10 SEL1 1
00:20:00:09 SEL3 1

cwplogic.out
00:00:05:010 gsel0 1
00:00:05:010 gsel1 1
00:20:05:009 rsel0 1
00:20:05:009 rsel1 1
00:20:05:009 rsel4 1
00:20:05:009 rsel5 1
00:20:05:009 rsel6 1
00:20:05:009 r_g 1

Luego elegimos el registro 0x1F y la ventana 0x1F, el resultado de este calculo supero el valor 512, por lo tanto se produce el calculo del modulo y el resultado obtenido es el registro 0x07 y el bit R/G en uno.

cwplogic1.ev
00:00:00:01 CWP0 1
00:00:00:02 CWP1 1
00:00:00:03 CWP2 1
00:00:00:04 CWP3 1
00:00:00:05 CWP4 1
00:00:00:06 SEL0 1
00:00:00:07 SEL1 1

```
00:00:00:08 SEL2        1
00:00:00:09 SEL3        1
00:00:00:10 SEL4        1

cwplogic1.out
00:00:05:010 rsel0 1
00:00:05:010 rsel1 1
00:00:05:010 rsel2 1
00:00:05:010 r_g 1
```

El componente Inc4 se encuentra configurado con un tiempo de 00:00:20:00. Para testearlo seteamos 0x03 en su entrada y obtenemos 0x07 en su salida.

```
inc4.ev
00:00:10:00 OP0 1
00:00:15:00 OP1 1

inc4.out
00:00:35:000 res0 1
00:00:35:000 res1 1
00:00:35:000 res2 1
```

Luego testamos su comportamiento al incrementar un valor en donde el resultado sobrepase los 32 bits. Primero con el valor 0xFFFFFFFF y luego con 0xFFFFFFFE, obtenemos en la salida 32 bits menos significativos del valor real (0x03 y 0x02 respectivamente). Ya que el inc4 se utiliza para incrementar los program counters este resultado no es de mucha importancia (si se intentara incrementar el PC mas allá de los 32bits no más espacio direccionable para el y el problema sería otro).

```
Inc41.ev
00:00:10:00 OP0 1
00:00:10:00 OP1 1
00:00:10:00 OP2 1
00:00:10:00 OP3 1
00:00:10:00 OP4 1
00:00:10:00 OP5 1
00:00:10:00 OP6 1
00:00:10:00 OP7 1
00:00:10:00 OP8 1
00:00:10:00 OP9 1
00:00:10:00 OP10 1
00:00:10:00 OP11 1
00:00:10:00 OP12 1
00:00:10:00 OP13 1
00:00:10:00 OP14 1
00:00:10:00 OP15 1
00:00:10:00 OP16 1
00:00:10:00 OP17 1
00:00:10:00 OP18 1
00:00:10:00 OP19 1
00:00:10:00 OP20 1
00:00:10:00 OP21 1
00:00:10:00 OP22 1
00:00:10:00 OP23 1
00:00:10:00 OP24 1
00:00:10:00 OP25 1
00:00:10:00 OP26 1
00:00:10:00 OP27 1
00:00:10:00 OP28 1
00:00:10:00 OP29 1
00:00:10:00 OP30 1
00:00:10:00 OP31 1
00:01:00:00 OP0 0
```

inc41.ev
00:00:30:000 res0 1
00:00:30:000 res1 1
00:01:20:000 res0 0

El componente Inc/Dec se encuentra configurado con un tiempo de 00:00:05:00. Primero realizamos una resta sobre el valor 0x0, con lo que obtenemos 0x1F en la salida ya que el indec posee un comportamiento “circular”. Luego seteamos el valor 0x1F y realizamos una suma, obteniendo el valor 0x0. Finalmente seteamos el valor 0x03 y como resultado de la suma obtenemos 0x04.

incdec.ev
00:00:10:00 FCOD 0
00:01:10:00 OP0 1
00:01:10:00 OP1 1
00:01:10:00 OP2 1
00:01:10:00 OP3 1
00:01:10:00 OP4 1
00:01:10:00 FCOD 1
00:02:10:00 OP0 1
00:02:10:00 OP1 1
00:02:10:00 OP2 0
00:02:10:00 OP3 0
00:02:10:00 OP4 0

incdec.out
00:00:15:000 res0 1
00:00:15:000 res1 1
00:00:15:000 res2 1
00:00:15:000 res3 1
00:00:15:000 res4 1
00:01:15:000 res0 0
00:01:15:000 res1 0
00:01:15:000 res2 0
00:01:15:000 res3 0
00:01:15:000 res4 0
00:02:15:000 res2 1

Testeamos el modulo IrqLogic, configurado con un tiempo de respuesta de 00:00:20:00 con un interrupt request 2

irqlogic.ev
00:00:10:000 IRQ2 1

irqlogic.out
00:00:30:000 TF 1
00:00:30:000 TT1 1
00:00:30:000 TT4 1

El Latch se encuentra configurado con un tiempo de 00:00:20:00. Primero cargamos 0x09 en la entrada y prendemos pero sin prender el bit de EIN, por lo tanto, el latch no cambia su valor, luego prendemos EIN y vemos el cambio reflejado en la salida, finalmente reseteamos mediante el bit Clear y la salida se vuelve cero (aunque los valores de la entrada no sean cero). Ya que el latch se logra alamcenando un entero (la entrada en bits es convertida de un arreglo de bits a un entero). El comportamiento para cualquier valor de entrada será similar.

latch.ev
00:00:01:00 EIN 0
00:00:10:00 IN0 1
00:00:13:00 IN3 1
00:01:01:00 EIN 1
00:01:10:00 IN0 1
00:01:11:00 IN1 0
00:01:12:00 IN2 0
00:01:13:00 IN3 1

01:00:01:00 CLEAR 1

latch.out
00:01:33:000 out0 1
00:01:33:000 out3 1
01:00:21:000 out0 0
01:00:21:000 out3 0

Modulo Mem

Vamos a leer la posicion cero de la memoria con el byte select en 0.

mem.ev

00:00:10:000 AS 1

mem.out

00:00:02:000 DATA0 1
00:00:02:000 DATA8 1
00:00:02:000 DTACK 1

Los bits 0 y 8 de datos en 1 corresponden al contenido de la memoria en ese momento en la direccion requerida (la memoria se cargó previamente desde el archivo).

Ahora probamos intentar acceder a una direccion de memoria por arriba del tamano establecido (32768)

mem.ev

00:00:01:000 AS 1
00:00:01:000 A2 1
00:00:01:000 A3 1
00:00:01:000 A4 1
00:00:01:000 A5 1
00:00:01:000 A6 1
00:00:01:000 A7 1
00:00:01:000 A8 1
00:00:01:000 A9 1
00:00:01:000 A10 1
00:00:01:000 A11 1
00:00:01:000 A12 1
00:00:01:000 A13 1
00:00:01:000 A14 1
00:00:01:000 A15 1
00:00:01:000 A16 1
00:00:01:000 A17 1
00:00:01:000 A18 1
00:00:01:000 A19 1
00:00:01:000 A20 1
00:00:01:000 A21 1

mem.out

00:00:02:000 ERR 1

Se encendio el bit de error porque la direccion es erronea.

Ahora testeamos escribir algo y luego leerlo:

mem.ev

00:00:01:000 BSEL0 1

00:00:01:000 BSEL1 1
00:00:01:000 BSEL2 1
00:00:01:000 BSEL3 1
00:00:01:000 DATA_IN0 1
00:00:01:000 RD_WR 1

mem.out

00:00:02:000 DTACK 1

Aqui nos devolvio el bit de acknowledge en 1. Ahora intentamos leer el dato.

mem.ev

00:00:01:000 AS 1
00:00:01:000 RD_WR 0
00:00:01:000 BSEL0 1
00:00:01:000 BSEL1 1
00:00:01:000 BSEL2 1
00:00:01:000 BSEL3 1
00:00:01:000 RESET 1

mem.out

00:00:02:000 DTACK 1
00:00:02:000 DATA_OUT0 1

Este test corresponde al modelo MulDiv que tiene cuatro operaciones UMUL,SMUL,UDIV,SDIV primero utilizamos la entrada de FCOD en 0 para testear la función UMUL y la entrada OPA=1024 y OPB=449. El modulo se encuentra configurado con un tiempo de 00:00:01:000.

mul.ev

00:00:10:000 OPA10 1
00:00:10:000 OPB0 1
00:00:10:000 OPB6 1
00:00:10:000 OPB7 1
00:00:10:000 OPB8 1

mul.out

00:00:11:000 RES10 1
00:00:11:000 RES16 1
00:00:11:000 RES17 1
00:00:11:000 RES18 1

En este caso res=459776 = 1024*449

Ahora testeamos UMUL pero con un bit en 3, que debería ser tomado como 1 (pues todo numero mayor a 0 se toma como un 1).

mul.ev

00:00:01:00 OPA1 3
00:00:01:00 OPB1 1

mul.out

00:00:02:00 RES2 1

Testeamos multiplicar un numero por 0 para ver si el modulo setea el flag Z.

mul.ev

00:00:01:00 OPA1 1

mul.out

00:00:02:00 Z 1

El resultado fue el flag activado.

Luego usamos la función SMUL

mul.ev
00:00:10:00 OPA10 1
00:00:10:00 OPA11 1
00:00:10:00 OPA12 1
00:00:10:00 OPA13 1
00:00:10:00 OPA14 1
00:00:10:00 OPA15 1
00:00:10:00 OPA16 1
00:00:10:00 OPA17 1
00:00:10:00 OPA18 1
00:00:10:00 OPA19 1
00:00:10:00 OPA20 1
00:00:10:00 OPA21 1
00:00:10:00 OPA22 1
00:00:10:00 OPA23 1
00:00:10:00 OPA24 1
00:00:10:00 OPA25 1
00:00:10:00 OPA26 1
00:00:10:00 OPA27 1
00:00:10:000 OPA28 1
00:00:10:000 OPA29 1
00:00:10:000 OPA30 1
00:00:10:000 OPA31 1
00:00:10:000 OPB1 1
00:00:10:000 FCOD0 1
00:00:10:000 FCOD1 0

mul.out

00:00:30:000 res11 1
00:00:30:000 res12 1
00:00:30:000 res13 1
00:00:30:000 res14 1
00:00:30:000 res15 1
00:00:30:000 res16 1
00:00:30:000 res17 1
00:00:30:000 res18 1
00:00:30:000 res19 1
00:00:30:000 res20 1
00:00:30:000 res21 1
00:00:30:000 res22 1
00:00:30:000 res23 1
00:00:30:000 res24 1
00:00:30:000 res25 1
00:00:30:000 res26 1
00:00:30:000 res27 1
00:00:30:000 res28 1
00:00:30:000 res29 1
00:00:30:000 res30 1
00:00:30:000 res31 1
00:00:30:000 yout0 1
00:00:30:000 yout1 1

```
00:00:30:000 yout2 1
00:00:30:000 yout3 1
00:00:30:000 yout4 1
00:00:30:000 yout5 1
00:00:30:000 yout6 1
00:00:30:000 yout7 1
00:00:30:000 yout8 1
00:00:30:000 yout9 1
00:00:30:000 yout10 1
00:00:30:000 yout11 1
00:00:30:000 yout12 1
00:00:30:000 yout13 1
00:00:30:000 yout14 1
00:00:30:000 yout15 1
00:00:30:000 yout16 1
00:00:30:000 yout17 1
00:00:30:000 yout18 1
00:00:30:000 yout19 1
00:00:30:000 yout20 1
00:00:30:000 yout21 1
00:00:30:000 yout22 1
00:00:30:000 yout23 1
00:00:30:000 yout24 1
00:00:30:000 yout25 1
00:00:30:000 yout26 1
00:00:30:000 yout27 1
00:00:30:000 yout28 1
00:00:30:000 yout29 1
00:00:30:000 yout30 1
00:00:30:000 yout31 1
00:00:30:000 n 1
```

Aquí se ve que el resultado obtenido en YOUT y RES = -2048 = -1024*2 (OPA*OPB)

Luego probamos UDIV

```
mul.ev

00:00:10:000 OPA10 1
00:00:10:000 OPB1   1
00:00:10:000 FCOD0 0
00:00:10:000 FCOD1 1

mul.out

00:00:10:000 RES9 1
```

Luego testeamos la division por cero.

```
mul.ev

00:00:01:000 OPA1 1
00:00:01:000 FCOD0 0
00:00:01:000 FCOD1 1

mul.out

00:00:02:000 Z 1
```

El componente Mux se encuentra configurado con un tiempo de 00:00:10:00. Primero cargamos 0x01 el puerto A y con 0x0F el B, como el bit SELA/SELB se encuentra apagado, en la salida obtenemos 0x0F (el valor del puerto B), luego seteamos a uno el bit de selección, y el puerto de salida se setea en 0x01.

```
mux.ev
00:00:10:00 A1 1
00:00:10:00 B0 1
00:00:10:00 B1 1
00:00:10:00 B2 1
00:00:10:00 B3 1
00:00:35:00 SELA_SELB 1

mux.out
00:00:20:000 out0 1
00:00:20:000 out1 1
00:00:20:000 out2 1
00:00:20:000 out3 1
00:00:45:000 out0 0
00:00:45:000 out2 0
00:00:45:000 out3 0
```

El componente RegBolck se encuentra configurado con un tiempo de 00:00:40:00. Primero cargamos 0x10000015 en el puerto CIN, y habilitamos la carga mediante CEN al registro 0x101 (mediante CSEL), deshabilitamos la entrada, ponemos CIN en cero y seleccionamos el registro 0x001, habilitamos la entrada por un minuto a continuación elegimos 0x101 para la salida por AOUT y 0x001 para BOUT, vemos que en la salida obtenemos los valores previamente seteados. Finalmente reseteamos el bloque y la salida se vuelve cero.

```
regblock.ev
00:00:00:01 CIN0        1
00:00:00:03 CIN2        1
00:00:00:05 CIN4        1
00:00:00:32 CIN31       1
00:00:00:33 CEN         1
00:00:00:51 CSEL0       1
00:00:00:59 CSEL8       1
00:00:01:33 CEN         0
00:01:02:02 CIN0        0
00:01:02:04 CIN31       0
00:01:02:59 CSEL8       0
00:01:03:33 CEN         1
00:02:03:33 CEN         0
10:00:00:33 ASEL0       1
10:00:00:41 ASEL8       1
10:00:00:42 BSEL0       1
20:00:00:00 RESET       0

regblock.out
10:00:00:090 aout0      1
10:00:00:090 aout2      1
10:00:00:090 bout2      1
10:00:00:090 aout4      1
10:00:00:090 bout4      1
10:00:00:090 aout31     1
20:00:00:040 aout0      0
20:00:00:040 aout2      0
20:00:00:040 bout2      0
20:00:00:040 aout4      0
20:00:00:040 bout4      0
20:00:00:040 aout31     0
```

Probamos luego seleccionar el mayor registro posible, 0x1FF, tanto para la entrada como para ambas salidas, y setearle el mayor valor posible 0xFFFFFFFF. Obtenemos la salida correcta. Finalmente reseteamos y la salida se vuelve cero.

```
Regblock1.ev
00:00:00:01 CIN0        1
00:00:00:02 CIN1        1
00:00:00:03 CIN2        1
```

00:00:00:04 CIN3 1
00:00:00:05 CIN4 1
00:00:00:06 CIN5 1
00:00:00:07 CIN6 1
00:00:00:08 CIN7 1
00:00:00:09 CIN8 1
00:00:00:10 CIN9 1
00:00:00:11 CIN10 1
00:00:00:12 CIN11 1
00:00:00:13 CIN12 1
00:00:00:14 CIN13 1
00:00:00:15 CIN14 1
00:00:00:16 CIN15 1
00:00:00:17 CIN16 1
00:00:00:18 CIN17 1
00:00:00:19 CIN18 1
00:00:00:20 CIN19 1
00:00:00:21 CIN20 1
00:00:00:22 CIN21 1
00:00:00:23 CIN22 1
00:00:00:24 CIN23 1
00:00:00:25 CIN24 1
00:00:00:26 CIN25 1
00:00:00:27 CIN26 1
00:00:00:28 CIN27 1
00:00:00:29 CIN28 1
00:00:00:30 CIN29 1
00:00:00:31 CIN30 1
00:00:00:32 CIN31 1
00:00:00:33 CEN 1
00:00:00:51 CSEL0 1
00:00:00:52 CSEL1 1
00:00:00:53 CSEL2 1
00:00:00:54 CSEL3 1
00:00:00:55 CSEL4 1
00:00:00:56 CSEL5 1
00:00:00:57 CSEL6 1
00:00:00:58 CSEL7 1
00:00:00:59 CSEL8 1
10:00:00:33 ASEL0 1
10:00:00:34 ASEL1 1
10:00:00:35 ASEL2 1
10:00:00:36 ASEL3 1
10:00:00:37 ASEL4 1
10:00:00:38 ASEL5 1
10:00:00:39 ASEL6 1
10:00:00:40 ASEL7 1
10:00:00:41 ASEL8 1
10:00:00:42 BSEL0 1
10:00:00:43 BSEL1 1
10:00:00:44 BSEL2 1
10:00:00:45 BSEL3 1
10:00:00:46 BSEL4 1
10:00:00:47 BSEL5 1
10:00:00:48 BSEL6 1
10:00:00:49 BSEL7 1
10:00:00:50 BSEL8 1
20:00:00:00 RESET 1

regblock1.out
10:00:00:090 aout0 1
10:00:00:090 bout0 1
10:00:00:090 aout1 1
10:00:00:090 bout1 1
10:00:00:090 aout2 1
10:00:00:090 bout2 1
10:00:00:090 aout3 1

10:00:00:090 bout3 1
10:00:00:090 aout4 1
10:00:00:090 bout4 1
10:00:00:090 aout5 1
10:00:00:090 bout5 1
10:00:00:090 aout6 1
10:00:00:090 bout6 1
10:00:00:090 aout7 1
10:00:00:090 bout7 1
10:00:00:090 aout8 1
10:00:00:090 bout8 1
10:00:00:090 aout9 1
10:00:00:090 bout9 1
10:00:00:090 aout10 1
10:00:00:090 bout10 1
10:00:00:090 aout11 1
10:00:00:090 bout11 1
10:00:00:090 aout12 1
10:00:00:090 bout12 1
10:00:00:090 aout13 1
10:00:00:090 bout13 1
10:00:00:090 aout14 1
10:00:00:090 bout14 1
10:00:00:090 aout15 1
10:00:00:090 bout15 1
10:00:00:090 aout16 1
10:00:00:090 bout16 1
10:00:00:090 aout17 1
10:00:00:090 bout17 1
10:00:00:090 aout18 1
10:00:00:090 bout18 1
10:00:00:090 aout19 1
10:00:00:090 bout19 1
10:00:00:090 aout20 1
10:00:00:090 bout20 1
10:00:00:090 aout21 1
10:00:00:090 bout21 1
10:00:00:090 aout22 1
10:00:00:090 bout22 1
10:00:00:090 aout23 1
10:00:00:090 bout23 1
10:00:00:090 aout24 1
10:00:00:090 bout24 1
10:00:00:090 aout25 1
10:00:00:090 bout25 1
10:00:00:090 aout26 1
10:00:00:090 bout26 1
10:00:00:090 aout27 1
10:00:00:090 bout27 1
10:00:00:090 aout28 1
10:00:00:090 bout28 1
10:00:00:090 aout29 1
10:00:00:090 bout29 1
10:00:00:090 aout30 1
10:00:00:090 bout30 1
10:00:00:090 aout31 1
10:00:00:090 bout31 1
20:00:00:040 aout0 0
20:00:00:040 bout0 0
20:00:00:040 aout1 0
20:00:00:040 bout1 0
20:00:00:040 aout2 0
20:00:00:040 bout2 0
20:00:00:040 aout3 0
20:00:00:040 bout3 0
20:00:00:040 aout4 0
20:00:00:040 bout4 0

20:00:00:040 aout5 0
20:00:00:040 bout5 0
20:00:00:040 aout6 0
20:00:00:040 bout6 0
20:00:00:040 aout7 0
20:00:00:040 bout7 0
20:00:00:040 aout8 0
20:00:00:040 bout8 0
20:00:00:040 aout9 0
20:00:00:040 bout9 0
20:00:00:040 aout10 0
20:00:00:040 bout10 0
20:00:00:040 aout11 0
20:00:00:040 bout11 0
20:00:00:040 aout12 0
20:00:00:040 bout12 0
20:00:00:040 aout13 0
20:00:00:040 bout13 0
20:00:00:040 aout14 0
20:00:00:040 bout14 0
20:00:00:040 aout15 0
20:00:00:040 bout15 0
20:00:00:040 aout16 0
20:00:00:040 bout16 0
20:00:00:040 aout17 0
20:00:00:040 bout17 0
20:00:00:040 aout18 0
20:00:00:040 bout18 0
20:00:00:040 aout19 0
20:00:00:040 bout19 0
20:00:00:040 aout20 0
20:00:00:040 bout20 0
20:00:00:040 aout21 0
20:00:00:040 bout21 0
20:00:00:040 aout22 0
20:00:00:040 bout22 0
20:00:00:040 aout23 0
20:00:00:040 bout23 0
20:00:00:040 aout24 0
20:00:00:040 bout24 0
20:00:00:040 aout25 0
20:00:00:040 bout25 0
20:00:00:040 aout26 0
20:00:00:040 bout26 0
20:00:00:040 aout27 0
20:00:00:040 bout27 0
20:00:00:040 aout28 0
20:00:00:040 bout28 0
20:00:00:040 aout29 0
20:00:00:040 bout29 0
20:00:00:040 aout30 0
20:00:00:040 bout30 0
20:00:00:040 aout31 0
20:00:00:040 bout31 0

El componente RegGlob se encuentra configurado con un tiempo de 00:00:00:50. Primero cargamos 0x80000015 en el puerto CIN, y habilitamos la carga mediante CEN al registro 0x05 (mediante CSEL), deshabilitamos la entrada y ponemos CIN en 0x014 y seleccionamos el registro 0x001, habilitamos la entrada por un minuto, luego seleccionamos el registro cero para la entrada y le intentamos setear el valor 0x01.A continuación elegimos 0x05 para la salida por AOUT y 0x001 para BOUT, vemos que en la salida obtenemos los valores previamente seteados. Ponemos Bsel en cero, pero no obtenemos el valor que antes intentamos setear, ya que %G0 se encuentra fijo en cero. Finalmente reseteamos el bloque y la salida se vuelve cero. La arquitectura presentaba un error en la especificación de este componente, ya que indicaba la existencia de cuatro bits de selección del registro, cuando en realidad solo hacen falta tres

(para seleccionar ocho registros), y la salida del CWPLogic tampoco posee 4 bits. El modelo original, ya presentaba este arreglo.

regglob.ev

```
00:00:00:01 CIN0        1
00:00:00:03 CIN2        1
00:00:00:05 CIN4        1
00:00:00:32 CIN31       1
00:00:00:33 CEN         1
00:00:00:51 CSEL0       1
00:00:00:53 CSEL2       1
00:00:01:33 CEN         0
00:01:02:02 CIN0        0
00:01:02:04 CIN31       0
00:01:02:53 CSEL2       0
00:01:03:33 CEN         1
00:02:03:33 CEN         0
01:01:02:02 CIN0        1
01:01:02:51 CSEL0       0
01:01:03:33 CEN         1
01:02:03:33 CEN         0
10:00:00:33 ASEL0       1
10:00:00:35 ASEL2       1
10:00:00:42 BSEL0       1
15:00:00:42 BSEL0       0
20:00:00:00 RESET       0
```

regglob.out

```
10:00:00:090 aout0      1
10:00:00:090 aout2      1
10:00:00:090 bout2      1
10:00:00:090 aout4      1
10:00:00:090 bout4      1
10:00:00:090 aout31     1
20:00:00:040 aout0      0
20:00:00:040 aout2      0
20:00:00:040 bout2      0
20:00:00:040 aout4      0
20:00:00:040 bout4      0
20:00:00:040 aout31     0
```

El shifter se encuentra configurado con un tiempo de 00:00:10:00. Primero cargamos 0x80000003 en el puerto OPA , 0x01 en OPB y mediante FCOD la operación SRA, en la salida obtenemos 0xC0000001. Luego seteamos la operación SRL y obtenemos 0x40000001. Finalmente cargamos la operación SLL y obtenemos 0x06.

shifter.ev

```
00:00:06:00 OPA0 1
00:00:06:00 OPA1 1
00:00:06:00 OPA31 1
00:00:06:00 OPB0  1
00:00:06:02 FCOD1 1
00:00:06:03 FCOD0 1
00:01:06:00 FCOD1 1
00:01:06:00 FCOD0 0
00:02:06:00 FCOD1 0
00:02:06:00 FCOD0 1
```

shifter.out

```
00:00:16:003 res0 1
00:00:16:003 res30 1
00:00:16:003 res31 1
00:01:16:000 res31 0
00:02:16:000 res0 0
00:02:16:000 res1 1
00:02:16:000 res2 1
```

00:02:16:000 res30 0

El componente SignExtN, como ya se mencionó, se utiliza para modelar SignExt13 y 22, aquí lo mostraremos seteado en 13 y 22 bits. El tiempo se encuentra configurado en 00:00:10:00. Cargamos en el operador de entrada 0x1003 y en la salida obtenemos 0xFFFFF003 para **nbits** igual a trece. Luego seteamos la entrada en 0x00o y el resultado obtenido es 0x00 ya que no hay signo que extender.

signext13.ev
00:00:06:00 OP0 1
00:00:06:00 OP1 1
00:01:06:00 OP12 1
00:02:06:00 OP0 0
00:02:06:00 OP1 0
00:02:06:00 OP12 0

signext14.out
00:00:16:000 res0 1
00:00:16:000 res1 1
00:01:16:000 res12 1
00:01:16:000 res13 1
00:01:16:000 res14 1
00:01:16:000 res15 1
00:01:16:000 res16 1
00:01:16:000 res17 1
00:01:16:000 res18 1
00:01:16:000 res19 1
00:01:16:000 res20 1
00:01:16:000 res21 1
00:01:16:000 res22 1
00:01:16:000 res23 1
00:01:16:000 res24 1
00:01:16:000 res25 1
00:01:16:000 res26 1
00:01:16:000 res27 1
00:01:16:000 res28 1
00:01:16:000 res29 1
00:01:16:000 res30 1
00:01:16:000 res31 1
00:02:16:000 res0 0
00:02:16:000 res1 0
00:02:16:000 res12 0
00:02:16:000 res13 0
00:02:16:000 res14 0
00:02:16:000 res15 0
00:02:16:000 res16 0
00:02:16:000 res17 0
00:02:16:000 res18 0
00:02:16:000 res19 0
00:02:16:000 res20 0
00:02:16:000 res21 0
00:02:16:000 res22 0
00:02:16:000 res23 0
00:02:16:000 res24 0
00:02:16:000 res25 0
00:02:16:000 res26 0
00:02:16:000 res27 0
00:02:16:000 res28 0
00:02:16:000 res29 0
00:02:16:000 res30 0
00:02:16:000 res31 0

Luego configuramos el signext con 22 bits y probamos algunos valores de entrada. Este test muestra que el modelo responde correctamente a las diferentes configuraciones de **nbits**.

Signext22.ev
00:00:06:00 OP0 1
00:00:06:00 OP1 1
00:01:06:00 OP21 1

signext22.out
00:00:16:000 res0 1
00:00:16:000 res1 1
00:01:16:000 res21 1
00:01:16:000 res22 1
00:01:16:000 res23 1
00:01:16:000 res24 1
00:01:16:000 res25 1
00:01:16:000 res26 1
00:01:16:000 res27 1
00:01:16:000 res28 1
00:01:16:000 res29 1
00:01:16:000 res30 1
00:01:16:000 res31 1

Ahora testeamos el modulo TrapLogic usando el TRAPINST en 1 y TN en 128, configurado con un tiempo de 00:00:21:000

traplogic.ev

00:00:10:000 TRAP_INST 1
00:00:10:000 TN0 1
00:00:10:000 TN1 1
00:00:10:000 TN2 1
00:00:10:000 TN3 1
00:00:10:000 TN4 1
00:00:10:000 TN5 1
00:00:10:000 TN6 1

traplogic.out

00:00:31:000 TF 1
00:00:31:000 TT0 1
00:00:31:000 TT1 1
00:00:31:000 TT2 1
00:00:31:000 TT3 1
00:00:31:000 TT4 1
00:00:31:000 TT5 1
00:00:31:000 TT6 1
00:00:31:000 TT7 1

Ahora probamos con dos interrupciónes ILLEG_INST y WIN_OVER.

traplogic.ev

00:00:10:000 ILLEG_INST 1
00:00:10:000 WIN_OVER 1

traplogic.out

00:00:11:000 TF 1
00:00:11:000 TT 1

Aqui el TT es 1 porque es la interrupción de mayor prioridad entre ILLEG_INST y WIN_OVER.

El componente wimcheck se encuentra configurado con un tiempo de 00:00:00:40. Cargamos 0x8000000F en WIM y 0x1F en CWP, obtenemos en la salida el valor del bit 0x1F de WIM, que es uno.

Luego ponemos CWP en 0x1D y la salida se vuelve cero. Finalmente pedimos bit menos significativo (CWP en 0x00) y volvemos a obtener un uno.

wimchek.ev

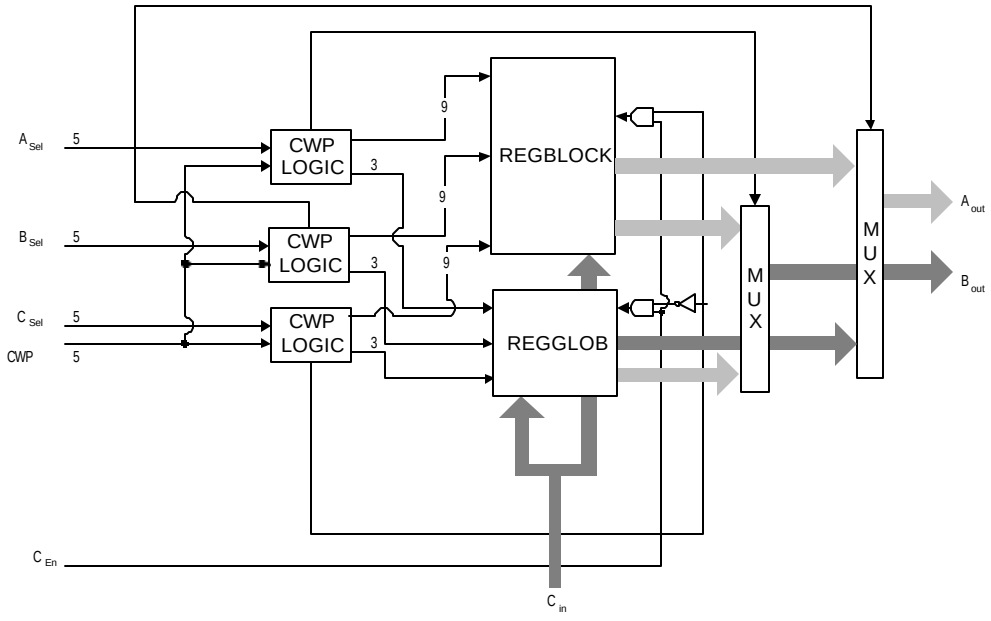
00:00:00:01 WIM0	**1**
00:00:00:02 WIM1	**1**
00:00:00:03 WIM2	**1**
00:00:00:04 WIM3	**1**
00:00:00:32 WIM31	**1**
01:00:00:00 CWP0	**1**
01:00:00:01 CWP1	**1**
01:00:00:02 CWP2	**1**
01:00:00:03 CWP3	**1**
01:00:00:04 CWP4	**1**
02:00:00:00 CWP1	**0**
01:00:00:00 CWP0	**0**
03:00:00:01 CWP1	**0**
03:00:00:02 CWP2	**0**
03:00:00:03 CWP3	**0**
03:00:00:04 CWP4	**0**

wimchek.out

00:00:00:072 res 1
02:00:00:040 res 0
03:00:00:044 res 1

Incluimos un test sobre el bloque de registros de uso general formado por tres CWPLogic, un RegBlock, un RegGlob, dos Mux y un And y un Not, que programamos particularmente para este test.

REGFILE



Primer seleccionamos el registro 0x00E para escribir (mediante CSEL), habilitamos la escritura (CEN en uno) y cargamos el valor 0x0B (en CIN), durante un minuto, y luego ponemos CEN en cero nuevamente. Después de un minuto volvemos a habilitar la escritura, pero esta vez, en el registro 0x02, lo que según vimos en el test de CWPLogic, desencadenara la escritura en RegGlob. Escribimos entonces en %G2 el valor 0x80000003 y luego el valor 0x8000000B en el registro 0, o sea en %G0, que como vimos antes, el regglob siempre devuelve cero, pero no asi el regblock (esto quiere decir que si luego pedimos el valor de ese registro y obtenemos cero, el valor viene de regglob y no de regblock). Después movemos la ventana CWP a 0x02 y seteamos en el registro 0x0E el valor 0xB000000A y luego deshabilitamos la entrada. Con el mismo CWP pedimos el registro 0x0B y obtenemos 0xB000000A. Movemos CWP a 0x0 y el valor del registro A es ahora 0x0000000B. Luego pedimos el valor del registro 0 en A y obtenemos 0x0. Por ultimo pedimos el valor del reg %G2 en B y obtenemos 0x80000003.

```
00:00:10:00 CSEL0 0
00:00:10:00 CSEL1 1
00:00:10:00 CSEL2 1
00:00:10:00 CSEL3 1
00:00:10:00 CEN 1
00:00:10:00 CIN0 1
00:00:10:00 CIN1 1
00:00:10:00 CIN2 0
00:00:10:00 CIN3 1
00:01:10:00 CEN 0

00:02:10:00 CEN 1
00:02:10:00 CSEL0 0
00:02:10:00 CSEL1 1
00:02:10:00 CSEL2 0
00:02:10:00 CSEL3 0

00:02:10:00 CIN0 1
00:02:10:00 CIN1 1
00:02:10:00 CIN2 0
00:02:10:00 CIN3 0
00:02:10:00 CIN31 1


00:03:10:00 CSEL1 0
00:03:10:00 CIN3 1
00:04:10:00 CEN 0

01:00:10:00 CSEL1 1
01:00:10:00 CSEL2 1
01:00:10:00 CSEL3 1
01:00:10:00 CWP0 0
01:00:10:00 CWP1 1
01:00:10:00 CEN 1
01:00:10:00 CIN0 0
01:00:10:00 CIN30 1
01:00:10:00 CIN31 1
01:01:10:00 CEN 0
02:00:10:00 ASEL0 0
02:00:10:00 ASEL1 1
02:00:10:00 ASEL2 1
02:00:10:00 ASEL3 1
02:10:10:00 CWP1 0
02:40:00:00 ASEL0 0
02:40:00:00 ASEL1 0
02:40:00:00 ASEL2 0
02:40:00:00 ASEL3 0
02:40:00:00 BSEL0 0
02:40:00:00 BSEL1 1
02:40:00:00 BSEL2 0
```

02:40:00:00 BSEL3 0

02:00:45:000 aout1 1
02:00:45:000 aout3 1
02:00:45:000 aout30 1
02:00:45:000 aout31 1
02:10:45:000 aout30 0
02:10:45:000 aout31 0
02:40:20:000 aout1 0
02:40:20:000 aout3 0
02:40:35:000 bout1 1
02:40:35:000 bout3 1
02:40:35:000 bout30 1
02:40:35:000 bout31 1

CONCLUSIONES

- La herramienta utilizada provee los mecanismos necesarios, facilitando la tarea y el enfoque sobre el problema a modelar, sin tener que preocuparse por el tareas generales de la simulación de eventos discretos.
- La independencia entre modelos permite que su reutilización en otro proyectos de simulación.
- Al estar la mayoria de los modelos programados de una manera similar se simplifica su mantenimiento.
- Como trabajo a futuro se prodrian reemplazar todos los componentes atomicos por su modelización con componentes de logica digital.

BIBLIOGRAFÍA Y REFERENCIAS

[**AG98**] Altman, D.; Glinsky, E. “Wimcheck, Traplogic, Regblock”. Seminario de Simulación de Eventos Discretos. Departamento de Computación. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. 1998.

[**BBW98a**] Barylko, A.; Beyoglonián, J.; Wainer, G. “GAD: a General Application DEVS environment”. Technical Report No. 98-001. Departamento de Computación. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. 1998.

[**BBW598**] Barylko, A.; Beyoglonián, J.; Wainer, G. “A General Application DEVS environment”. Technical Report No. 98-005. Departamento de Computación. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. 1998.

[**DDTZW98**] Daicz, S.; Diuk, C.; Troccoli, A.; Zlotnik, S.; Wainer, G. “Arquitectura del Modelo”. Seminario de Simulación de Eventos Discretos. Departamento de Computación. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. 1998.

[EG99] Enrique, S.; Glinsky, E.; “Implementación de componentes de un procesador SPARC utilizando simulación de eventos discretos” Curso de Organización de Computadoras I. Departamento de Computación. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. 1999.

[**ER99**] Enrique, S.; Rubinstein D. “Mem, Mux4, Inc/Dec”. Seminario de Simulación de Eventos Discretos. Departamento de Computación. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. 1999. http://members.xoom.com/senrique/files

[**FRW97**] Ferrari, A.; Romano, S.; Wainer, G. “Diseño e Implementación de una Biblioteca de Lógica Digital”. Curso de Organización de Computadoras I. Departamento de Computación. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. 1997.

[SB00] Stroustrup, B. ¨The C++ Programing Lenguaje¨, Special Edition, Addison Wesley, 2000.

[**WG96**] Wainer, G. “Introducción a la Simulación de Eventos Discretos”. Technical Report No. 96-005. Departamento de Computación. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. 1996.

[**Zei76**] Zeigler, B. “Theory of modeling and simulation”. Wiley, 1976.