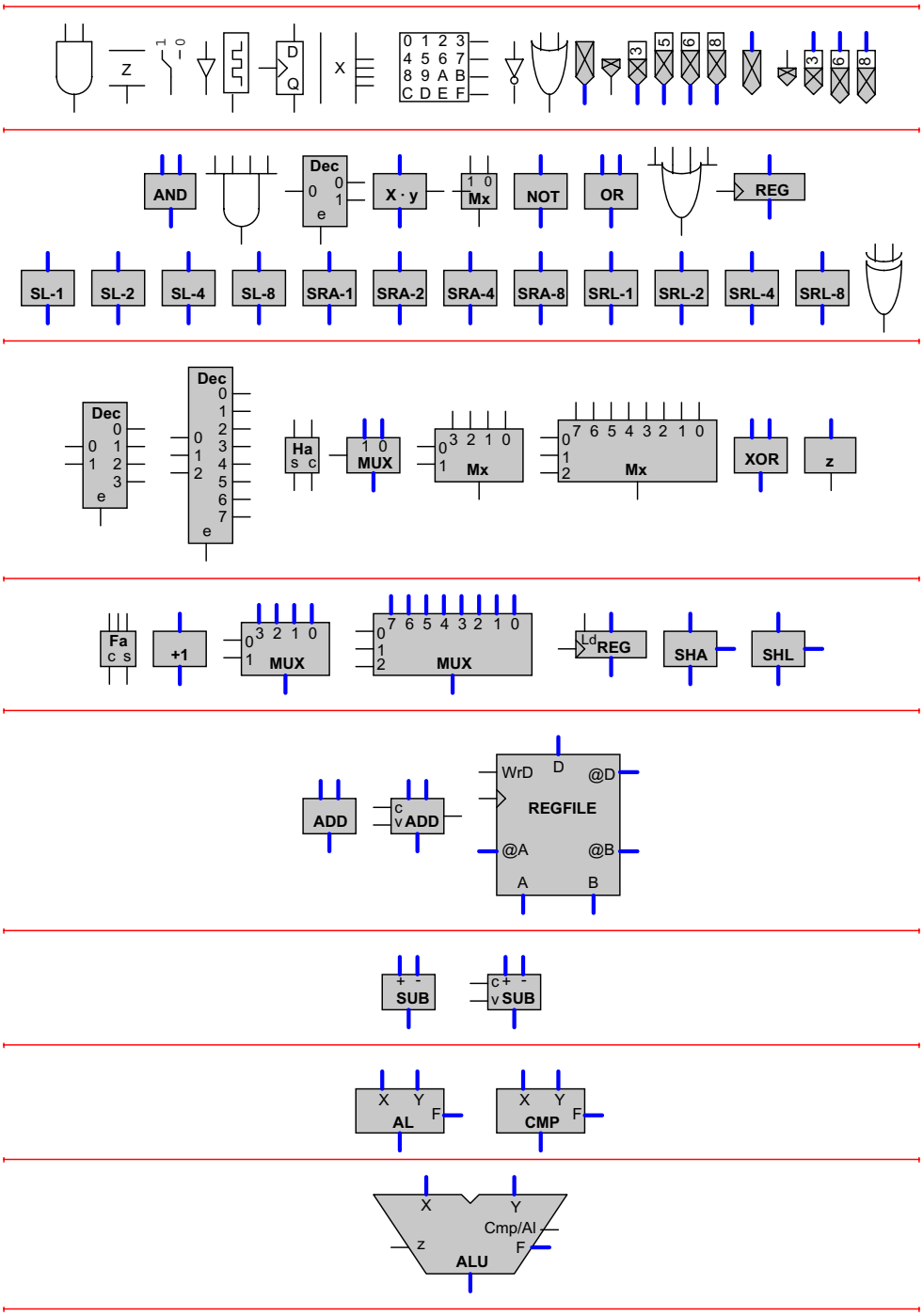
Digit@Lib

Juan J. Navarro
Toni Juan

Primera versión: 13-04-2004
Versión actual: 1-04-2005

Departament d´Arquitectura de Computadors
Facultat d´Informàtica de Barcelona
Universitat Politècnica de Catalunya



1 Introducción

En este documento se presenta la librería Digit@Lib, una librería básica de dispositivos lógicos combinacionales y secuenciales creados para ser usados por el programa de diseño y simulación de circuitos digitales LogicWorks versión 4.1. Más información sobre LogicWorks puede encontrarse en **http://www.logicworks4.com/index.htm**.

Digit@Lib ha sido creada con el objetivo de ser útil en la enseñanza y el aprendizaje del diseño digital y de la arquitectura de computadores. Dispone tanto de dispositivos que operan con unos pocos bits como de dispositivos que operan con palabras de 16-bits. Los componentes de la librería han sido diseñados usando bloques multinivel. Los dispositivos de nivel 0 son los más básicos y no tienen adjunto ningún circuito interno para especificar su funcionamiento. Son dispositivos primitivos, cuyo comportamiento está predefinido en LogicWorks. Ejemplos de dispositivos de nivel cero son, And-2 (puerta And de dos entradas) y Flip-flop (biestable D activado por flanco ascendente). Cada dispositivo de nivel superior al 0 está compuesto por dispositivos de niveles inferiores interconectados entre sí. Por ejemplo, el dispositivo ALU es una unidad aritmético-lógica que se encuentra en el nivel 7 de la librería y está formada por un dispositivo CMP y otro AL, de nivel 6, un dispositivo MUX-2-1, de nivel 3 y un dispositivo Zero, de nivel 2. A su vez cada uno de estos dispositivos están formados por otros de niveles inferiores al suyo.

El circuito interno de un dispositivo que se encuentra en un circuito abierto por LogicWorks puede verse haciendo doble clic sobre el símbolo del dispositivo. Por motivos pedagógicos cada circuito interno de cada dispositivo de Digit@Lib está dibujado sobre una hoja de tamaño A4, lo que facilita su visualización por pantalla y su impresión. En la hoja hay dos partes, en la superior se muestra una ficha con información descriptiva del dispositivo y en la parte inferior se encuentra el circuito asociado al dispositivo. En la ficha descriptiva del dispositivo se encuentra su nombre en la librería, que suele tener pocos caracteres, su nombre completo, el símbolo del dispositivo con el nombre de todas sus patillas y una descripción detallada que incluye un texto explicativo de la funcionalidad y/o una tabla de verdad y/o una expresión lógica para cada salida y/o las operaciones aritméticas que realiza.

El objetivo al crear Digit@Lib es que se usen sus dispositivos para crear nuevos dispositivos (que formarán librerías de dispositivos de nivel superior) y circuitos que las usen. Ejemplo de este uso es la carpeta SISP-I-1-Circuits&Lib que contiene circuitos que implementan un computador basado en la arquitectura pedagógica SISA-I y las librerías, extensiones de la Digit@Lib, necesarias para construir los circuitos.

2 Sobre el símbolo, mnemotécnico y patillas de un dispositivo

El símbolo de cada dispositivo se ha diseñado lo suficientemente pequeño como para que cualquiera de los circuitos internos de los dispositivos de la librería quepa en una hoja A4. Por razones de espacio dentro de los márgenes del símbolo y por razones estéticas, se han etiquetado con su nombre o número solamente las patillas cuya diferenciación es imprescindible para el correcto uso del dispositivo. Hemos definido una serie de convenciones para diferenciar unas patillas de otras sin tener que etiquetarlas con su nombre o número. A continuación describimos estas convenciones junto con otra información de interés.

La forma del símbolo usado para cada dispositivo del nivel 0 y puerta de niveles superiores sirve para diferenciarlo del resto sin tener que etiquetarlo con su nombre. El resto de dispositivos, que tienen forma rectangular --excepto el dispositivo ALU--, van etiquetados con un mnemotécnico. El mnemotécnico consiste en unos cuantos caracteres que nos recuerdan el nombre del dispositivo en la librería y su funcionalidad. Por ejemplo, el Dec-2-4 (nombre del dispositivo en la librería) es un 2-to-4 Binary Decoder (nombre completo) cuyo mnemotécnico es Dec. Este mnemotécnico es el mismo para el Dec-1-2, Dec-2-4 y Dec-3-8, ya que el número de patillas del símbolo es suficiente para diferenciarlos entre sí. Otro ejemplo es el mnemotécnico +1 que se refiere al dispositivo INC que es un 16-bit Incrementer.

El mnemotécnico del dispositivo se encuentra siempre en el interior del símbolo del dispositivo. En general, el mnemotécnico se sitúa cerca de la patilla de salida, para diferenciar esta de las patillas de entrada. Excepciones a esta regla son los decodificadores (Dec), el Half-adder (Ha), el Full-adder (Fa) y el banco de registros (REGFILE). En los decodificadores las entradas están en un lado de la caja y las salidas en otro y se diferencian por el número de patillas: k entradas y 2^k salidas. Tanto el Ha como el Fa tienen dos salidas etiquetadas con una c (carry) y una s (sum), ya que es necesario diferenciarlas. En el dispositivo REGFILE todas las entradas y salidas tienen etiqueta con su nombre.

Se han numerado las patillas, tanto de entrada como de salida, en los dispositivos en que estas no son intercambiables, como los decodificadores y multiplexores. En el caso del restador SUB se han etiquetado con el símbolo + y - las entradas para diferenciar cuál es el minuendo y cuál es el sustraendo.

Aunque no aparezca el nombre de una patilla en el símbolo de un dispositivo por razones de espacio y estéticas, todas las patillas debentener un nombre, para poder expresar con claridad la funcionalidad del dispositivo. En la parte superior derecha de la ficha de descripción de cada dispositivo se incluye su símbolo y en el exterior de cada patilla se indica su nombre. El nombre de cada patilla coincide con el del PortConnector correspondiente a esa patilla en el circuito interno del dispositivo.

Las patillas de un dispositivo pueden ser de un bit (binary input, binary output o 1-bit input, 1-bit output) o buses de n bits -- vectores de n bits-- (n-bit input, n-bit output). Este aspecto se ve reflejado en el nombre de la patilla/PortConnector.

- 1-bit: se nombran con una letra minúscula (v.g.: x) o con una letra minúscula seguida de un número (por ejemplo, x1) o con una secuencia de letras en las que la primera letra tiene que ser mayúscula, pero no pueden ser todas mayúsculas (v.g: Ld, WrRd). En un circuito, la señal de selección de un multiplexor-2-1 (tanto de buses como de bits) se suele nombrar con una secuencia de caracteres que hace referencia a la entrada o a la acción seleccionada cuando la señal de selección vale 1 (v.g.: TknBr). También se puede nombrar con dos nombres separados por el símbolo /. El primer nombre hace referencia a la entrada seleccionada por el valor 1 y el segundo a la entrada seleccionada por un 0 (v.g.: In/Ld)
- n-bits: En el nombre no se ve el valor de n (que siempre es mayor que 1 y en la mayoría de los casos en esta librería vale 16, aunque hay casos en que vale 6 y 3). Se nombran con una o varias letras mayúsculas que pueden ir seguidas de un número (v.g.: A, D0, DATAIN).

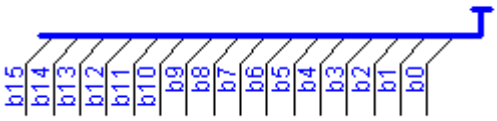
3 Sobre las señales/buses y los mnemotécnicos

Las mismas normas que se aplican a las patillas de un dispositivos (y PortConnectors de su circuito interno) se aplican a las **señales binarias** (1-bit) o a los **buses** (n-bits) que conectan patillas de dispositivos en un circuito. Además, estas mismas reglas se siguen para los mnemotécnicos y nombres de los dispositivos, de acuerdo con el número de bits de sus salidas. Por ejemplo: AND es el mnemotécnico y el nombre del 16 Bit-wise And Operator, MUX-2-1 es el nombre del 16 Bit-wide 2-to-1 Multiplexer mientras que Mx-2-1 es el nombre del 2-to-1 Multiplexer.

En LogicWorks, para obtener las señales binarias contenidas en un bus de n bits se usa un Breakout, como se muestra en la siguiente figura. Sea cual sea el nombre de un bus, en todos los circuitos internos de esta librería, las señales binarias que lo forman se nombran **siempre** como (ver figura):

bn-1, ...b2, b1, b0

La letra b hace referencia a bit. Necesitamos poner una letra ya que si una señal de un bus se llama 0 o 1, el valor de esa señal se fija siempre a 0 o 1, y eso no es lo que queremos. Esto es muy importante ya que si las señales que forman un bus no tienen estos nombres, los PortConectors de la librería no pueden funcionar correctamente. Por supuesto que buses distintos tienen que tener nombres distintos, pero sus señales internas siempre se deben llamar bn-1, ...b2, b1, b0.



4 Interpretación de la información representada en un bus o patilla de n-bits

Por último definimos un convenio que usamos a menudo en las fichas descriptivas de los dispositivos para indicar el valor que representa un vector de n bits.

Para hacer referencia a un bit concreto de un bus, o de un PortConnector de n-bits, usaremos el nombre del bus seguido del nombre del bit entre paréntesis. Por ejemplo, si el bus se denomina ALUOUT, denotamos el bit 0 como ALUOUT(b0). Para

referirnos a un subvector de bits denotamos entre paréntesis el rango de bits indicando primero el nombre del bit de mayor peso seguido del símbolo ':' y a continuación el nombre del bit de menor peso. Por ejemplo, ALUOUT(b5:b0).

Un bus, una patilla de n-bits, o un PortConector de n-bits, es un vector de n bits que puede codificar cualquier tipo de información. El significado de cada una de las posibles codificaciones del vector de bits se especificará mediante una tabla para describir la funcionalidad del dispositivo. En caso de que la interpretación sea de un número natural codificado en binario (unsigned integer) o de un número entero codificado en complemento a 2 (signed integer) se usa la siguiente convención. Para referirnos al valor numérico que representa el vector de bits se usa el mismo nombre del bus/patilla seguido de la letra minúscula u, para el valor interpretado como natural (unsigned), o bien seguido de la letra minúscula s para la interpretación como entero (signed). Así pues, dado el bus/patilla X,

$$Xu = \sum_{i=0}^{n-1} X(bi) \times 2^i$$

$$Xs = -X(bn-1) \times 2^{n-1} + \sum_{i=0}^{n-2} X(bi) \times 2^i$$

Fe de erratas

- En el dispositivo Flip-Flop, la Entrada D y la salida Q son de 1 bit y por ello, siguiendo con nuestra propia norma deberían llamarde d y q (con minúsculas) respectivamente.

Agradecimientos

Deseamos agradecer a Ramón Canal, Llorenç Cruz, Anna del Corral, Eduard Lara,Víctor Muntés y Oscar Palomar, por la ayuda recibida en la revisión del documento y por la propuesta de sugerencias para mejorarlo.

Contents

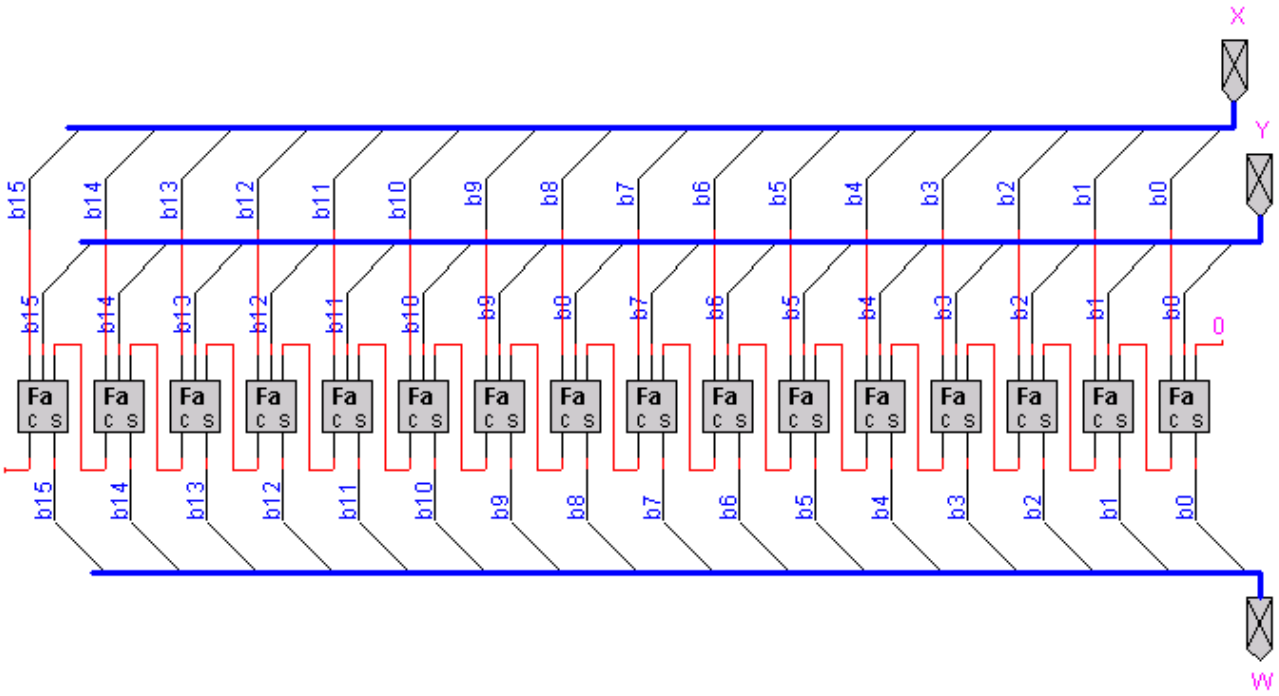
16 Bit-wide ADDER

ADD

Description:
Interpreting the 16-bit inputs X and Y and the 16-bit output W as unsigned integers or as signed integers, the output is equal to the addition of the two inputs except when overflow occurs. No overflow detection is done.

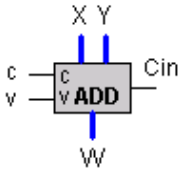
Arithmetic Operation:
Interpreting X, Y and W as unsigned integers,
If ((Xu + Yu) < 2**16) then
Wu = Xu + Yu
else
Wu = Xu + Yu - 2**16

Interpreting X, Y and W as signed integers,
if (-2**15 <= (Xs + Ys) <= (2**15)-1) then Ws = Xs + Ys
if ((Xs + Ys) > (2**15)-1) then Ws = Xs + Ys - 2**16
if (-2**15 > (Xs + Ys)) then Ws = Xs + Ys + 2**16



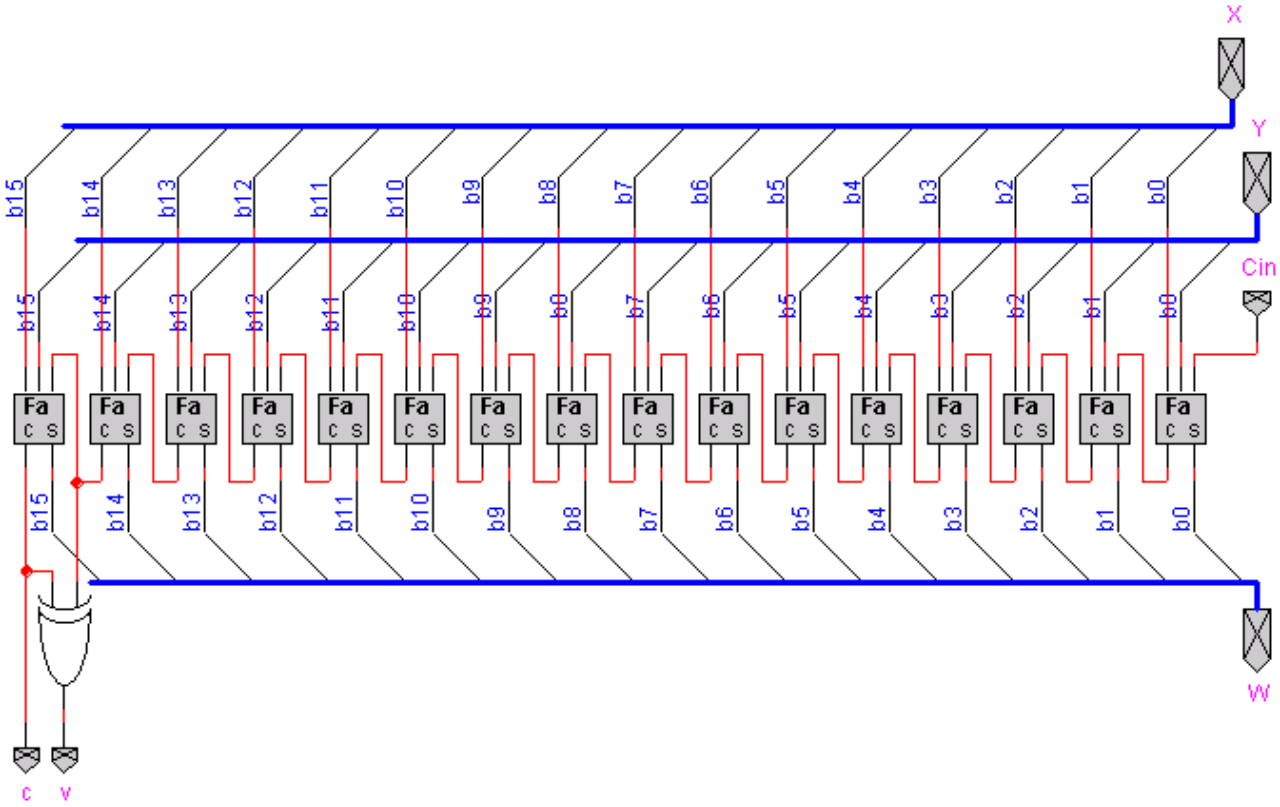
16 Bit-wide Adder with Carry In and Carry and Overflow Outputs ADDwCinCV

Description:
The 16-bit inputs X and Y and the 16-bit output W can be interpreted as signed or as unsigned integers. For both cases the output is equal to the addition of the two inputs plus the binary input Cin, except when overflow occurs. Overflow detection is indicated with the binary outputs c and v for unsigned and signed addition respectively.



Arithmetic Operation:
Interpreting X, Y and W as unsigned integers,
If ((Xu + Yu + Cin) <= (2**16)-1) then
Wu = Xu + Yu + Cin, c = 0
else
Wu = Xu + Yu + Cin - 2**16, c = 1

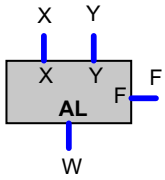
Interpreting X, Y and W as signed integers,
if (-2**15 <= (Xs + Ys + Cin) <= (2**15)-1) then Ws = Xs + Ys + Cin, v = 0
if ((Xs + Ys + Cin) > (2**15)-1) then Ws = Xs + Ys + Cin - 2**16, v = 1
if (-2**15 > (Xs + Ys + Cin)) then Ws = Xs + Ys + Cin + 2**16, v = 1

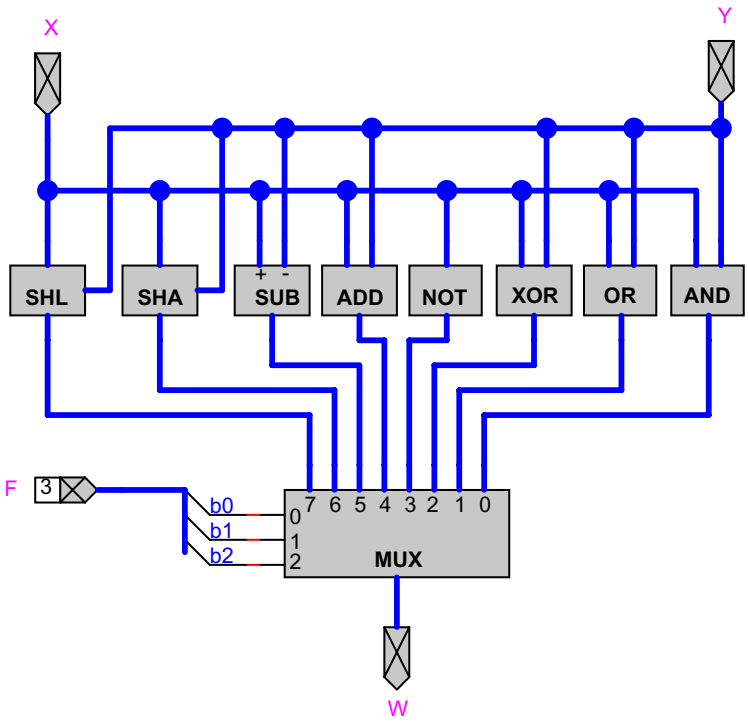


16 Bit-wide Arithmetic and Logic Operators AL

Description:
The 16-bit output W is the arithmetic or logic function chosen by the 3-bit selection input F, applied to the 16-bit inputs X and Y.
The functions are:

F	W
000	AND (X, Y)
001	OR (X, Y)
010	XOR(X, Y)
011	NOT (X)
100	ADD (X, Y)
101	SUB (X, Y)
110	SHA (X, Y)
111	SHL (X, Y)



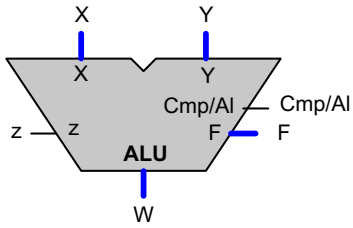


16 Bit-wide ALU

ALUupg

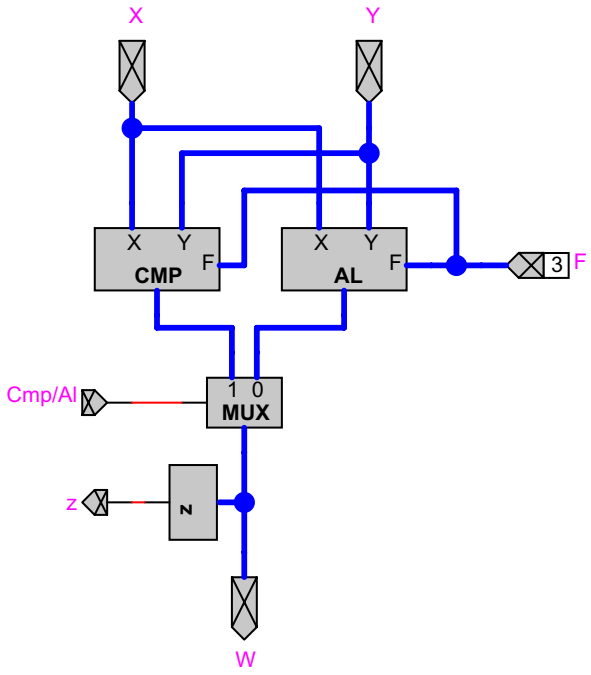
Description:

The 16-bit output W is the result of an arithmetic/logic operation or a comparison applied to the 16-bit inputs X and Y. The type of operation is selected by the 3-bit input F and by the binary input Cmp/Al according to the table:



OP	Cmp/Al = 1	Cmp/Al = 0
000	CMPLT(X, Y)	AND(X, Y)
001	CMPLE(X, Y)	OR(X, Y)
010	---	XOR(X,Y)
011	CMPEQ(X, Y)	NOT(X)
100	CMPLTU(X, Y)	ADD(X, Y)
101	CMPLEU(X, Y)	SUB(X, Y)
110	---	SHA(X ,Y)
111	---	SHL(X, Y)

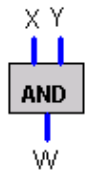
If the ALU performs a comparison the result can be true or false. If the result is true then W(b0) = 1 else W(b0) = 0. Moreover, W(bi) = 0 for i = 1 to 15.
The 1-bit output z indicates when the output W is a vector of 16 zeroes.

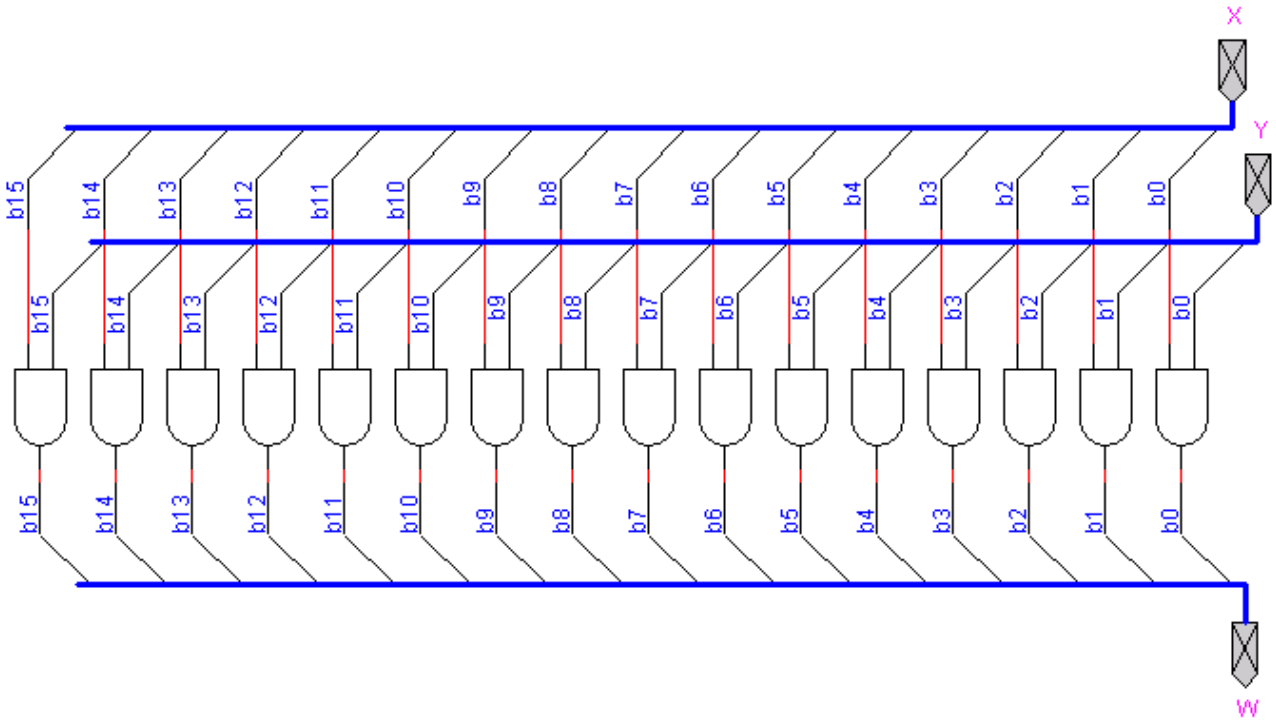


16 Bit-wise And Operator — AND

Description:
The 16-bit output W is the result of a bit-wise logical And operation of the 16-bit inputs X and Y.

Bit-level Logical Operation:
W(bi) = X(bi) * Y(bi) for i = 0 to 15

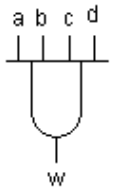


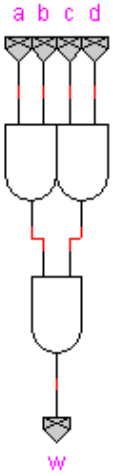


4-Input And Gate — And-4

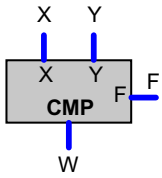
Description:
The binary output w takes the value 1 if all of the binary inputs a, b, c and d take the value 1, otherwise (one or more of the inputs are 0), w takes the value 0.

Bit-level Logical Operation:
w = a + b + c + d





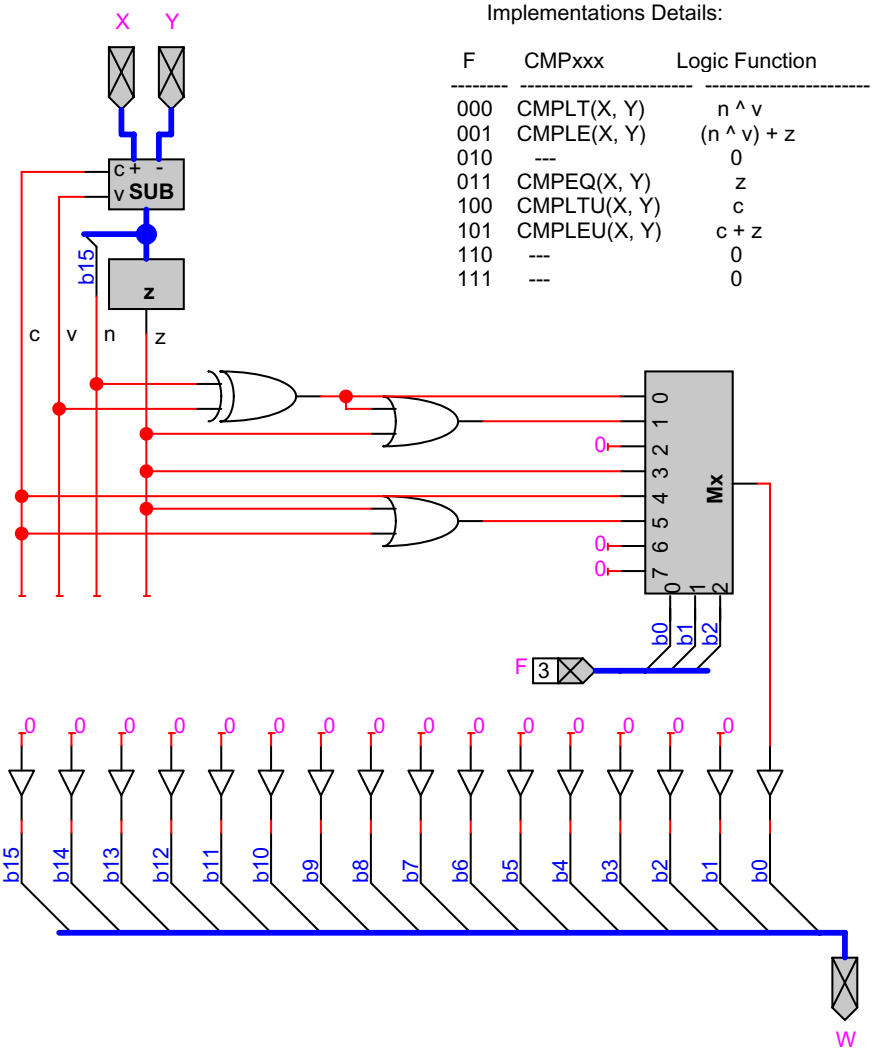
16 Bit-wide Signed and Unsigned Comparator — CMP



Description:

The 16-bit output W has only two posible values as the result of a comparison: true or false. True is coded as W(b0) = 1 and false as W(b0) = 0.
For all the cases W(bi) = 0 for i = 1 to 15. The type of comparison and the consideration of the 16-bit inputs X and Y as signed or as unsigned integers is chosen by the 3-bit selection input F according to the next table:

F	W	CMPxx(X, Y)	Name
000	Xs < Ys	CMPLT(X, Y)	Less Than (Signed)
001	Xs <= Ys	CMPLE(X, Y)	Less than or Equal (Signed)
010	---	---	
011	X ==Y	CMPEQ(X, Y)	Equal
100	Xu < Yu	CMPLTU(X, Y)	Less Than Unsigned
101	Xu <= Yu	CMPLEU(X, Y)	Less than or Equal Unsigned
110	---	---	
111	---	---	

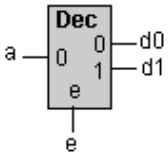


Implementations Details:

F	CMPxxx	Logic Function
000	CMPLT(X, Y)	n ^ v
001	CMPLE(X, Y)	(n ^ v) + z
010	---	0
011	CMPEQ(X, Y)	z
100	CMPLTU(X, Y)	c
101	CMPLEU(X, Y)	c + z
110	---	0
111	---	0

1-to-2 Binary Decoder — Dec-1-2

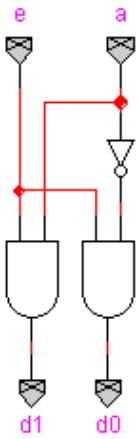
Description:
When the enable binary input e is equal to 1, if the adress binary input a is 0 then the data binary outputs d0 and d1 take the values of 1 and 0 respectively, otherwise d0 takes the value 0 and d1 takes the value 1. When the enable input e is equal to 0 all the outputs d1 and d0 take the value of 0.



Bit-level Logical Operation:
d0 = e * !a
d1 = e * a

Truth Table:

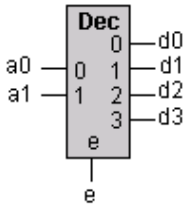
e	a	d1	d0
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	0



2-to-4 Binary Decoder — Dec-2-4

Description:

When the enable binary input e is equal to 1, only one of the four data binary ouputs d3,..., d0 is equal to 1, depending on the value of the 2-bit address input a1, a0.
The output dk takes the value 1 where k = a1 * 2 + a0 and all other outputs take the value 0.
When the enable binary input e is equal to 0 all data outputs d3 to d0 are 0.

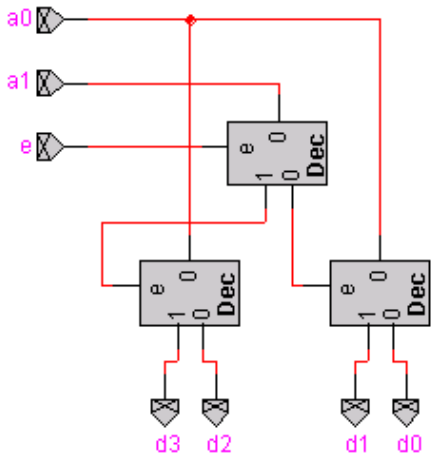


Bit-level Logical Operation:

dk = e * mk for k = 0 to 3
Where mk is the minterm k of the logical inputs (a1, a0)

Truth Table (compressed):

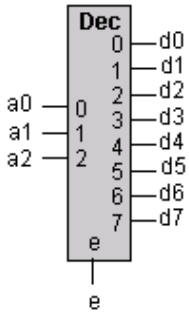
e	a1	a0	d3	d2	d1	d0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0



3-to-8 Binary Decoder — Dec-3-8

Description:

When the enable binary input e is equal to 1, only one of the eight data binary ouputs d7,..., d0 is equal to 1, depending on the value of the 3-bit address input a2, a1, a0.
The output dk takes the value 1 where k = a2 * 4 + a1 * 2 + a0 and all other outputs take the value 0.
When the enable binary input e is equal to 0 all data outputs d7 to d0 are 0.

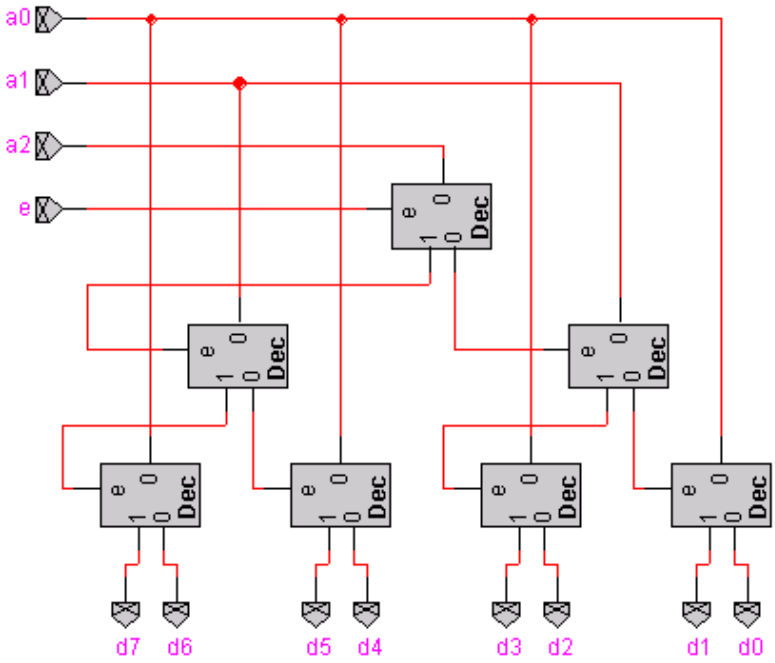


Bit-level Logical Operation:

dk = e * mk for k = 0 to 7
Where mk is the minterm k of the logical inputs (a2, a1, a0)

Truth Table (compressed):

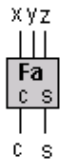
e	a2	a1	a0	d7	d6	d5	d4	d3	d2	d1	d0
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	1	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0



Full-adder

Fa

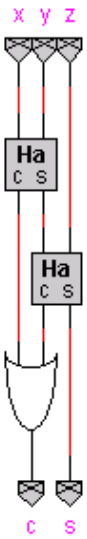
Description:
The 2-bit unsigned integer represented by the carry c and the sum s binary outputs is the arithmetic addition of the binary inputs x, y and z.



Arithmetic Operation:
wu = x + y + z
Where wu is the integer number represented in the binary number system by the 2-bit vector W = (c, s) so that wu = c * 2 + s.

Truth Table:

x	y	z	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Half-adder

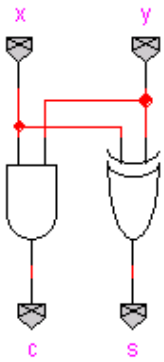
Ha

Description:
The 2-bit unsigned integer represented by the carry c and the sum s binary outputs is the arithmetic addition of the binary inputs x and y.

Arithmetic Operation:
wu = x + y
Where wu is the unsigned integer number represented in the binary number system by the 2-bit vector W = (c, s) so that wu = c * 2 + s.

Truth Table:

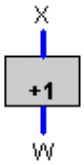
x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



16-Bit Incrementer

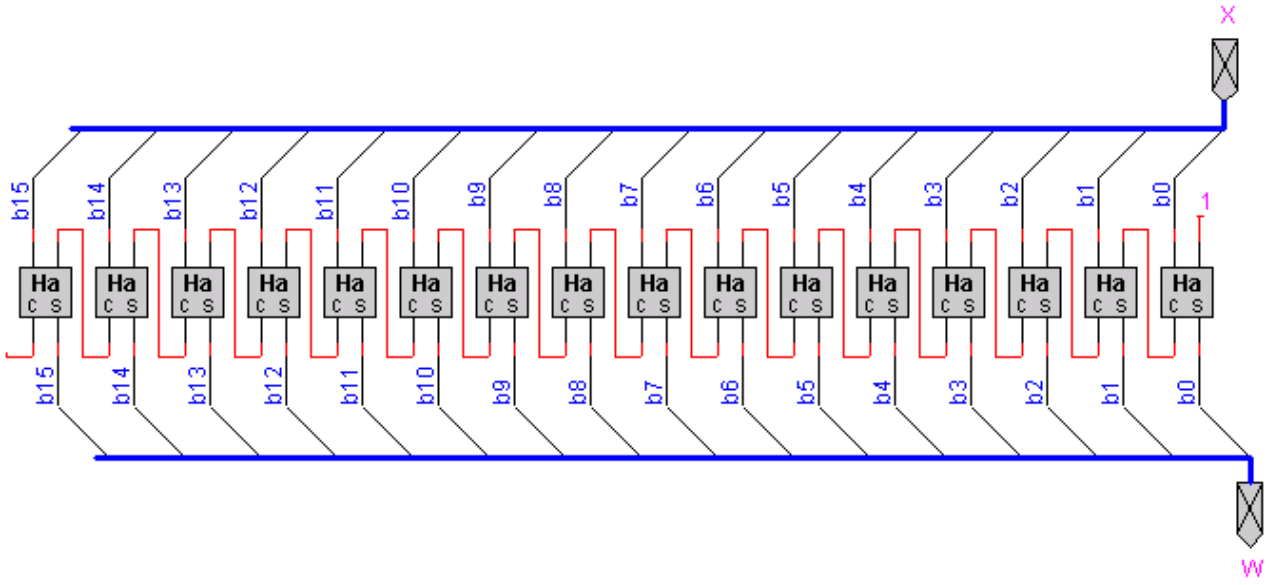
INC

Description:
Interpreting the 16-bit input X and the 16-bit output W as unsigned integers or as signed integers, the output is equal to the input incremented by 1 except when overflow occurs. No overflow detection is done.



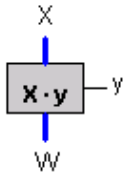
Arithmetic Operation:
Interpreting X and W as unsigned integers,
If (Xu < (2**16) -1) then
Wu = Xu +1
else
Wu = 0

Interpreting X and W as signed integers,
If (Xs < (2**15) - 1) then
Ws = Xs +1
else
Ws = - 2**15



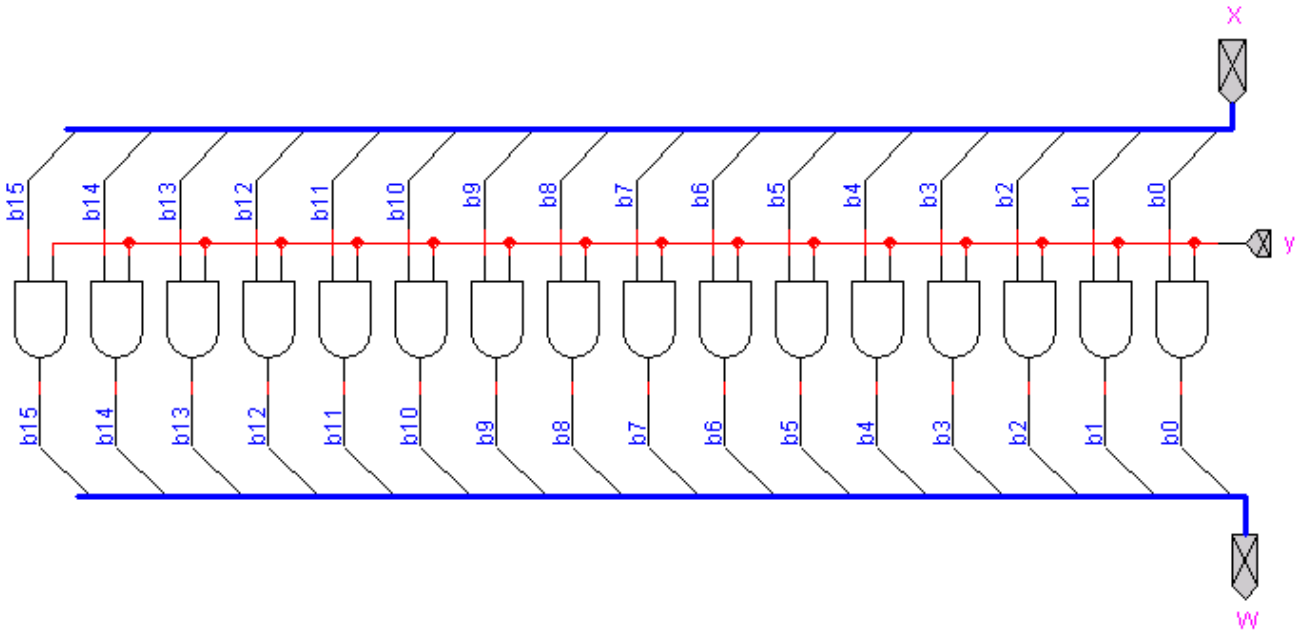
16 Bit-wide by 1 Bit Multiplier — MULBIT

Description:
Interpreting the 16-bit input X and the 16-bit output W as unsigned integers or as signed integers, the output is equal to the input multiplied by the binary input y.



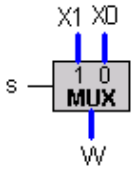
Arithmetic Operation:
Interpreting X and W as unsigned integers,
Wu = Xu * y
Interpreting X and W as signed integers,
Ws = Xs * y

Bit-level Logical Operation:
W(bi) = X(bi) * y for i = 0 to 15



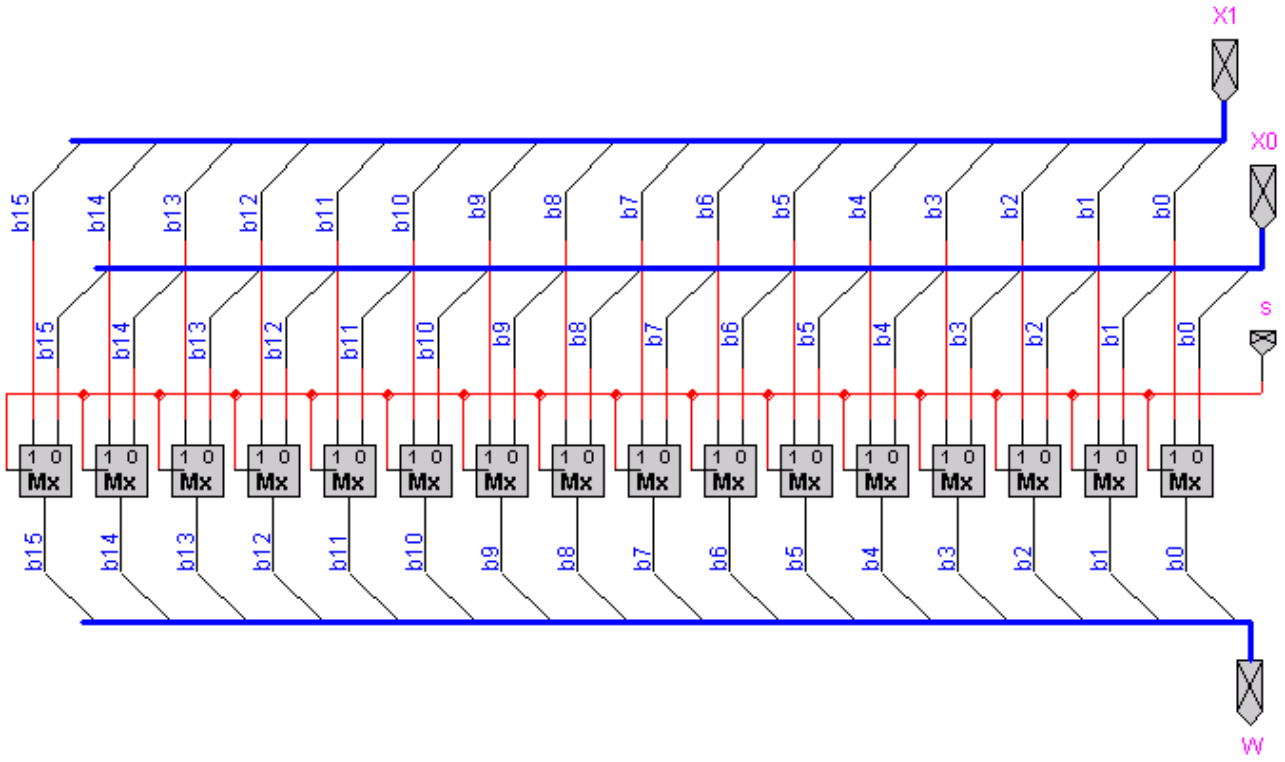
16 Bit-wide 2-to-1 MULTIPLEXER — MUX-2-1

Description:
The 16-bit output W is either the 16-bit input X1 or X0 depending on the value of the binary selection input s: If (s = 1) then W = X1 else W = X0



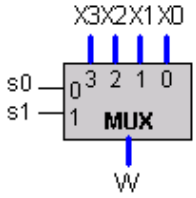
Truth Table (compressed):

s	W
0	X0
1	X1



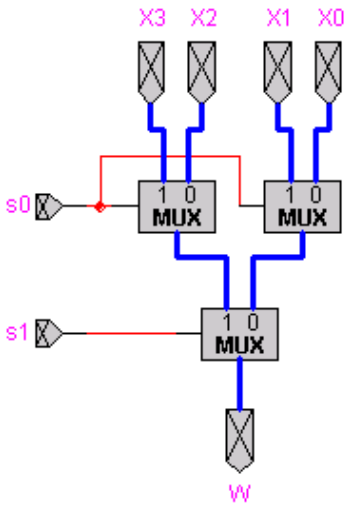
16 Bit-wide 4-to-1 Multiplexer — MUX-4-1

Description:
The 16-bit output W is one of the four 16-bit inputs X0 to X3 depending on the value of the 2-bit selection input s1, s0:
W = Xk, where k = s1 * 2 + s0



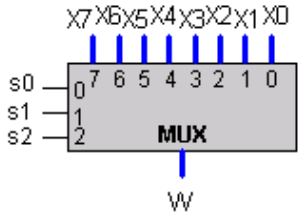
Truth Table (compressed):

s1	s0	W
0	0	X0
0	1	X1
1	0	X2
1	1	X3



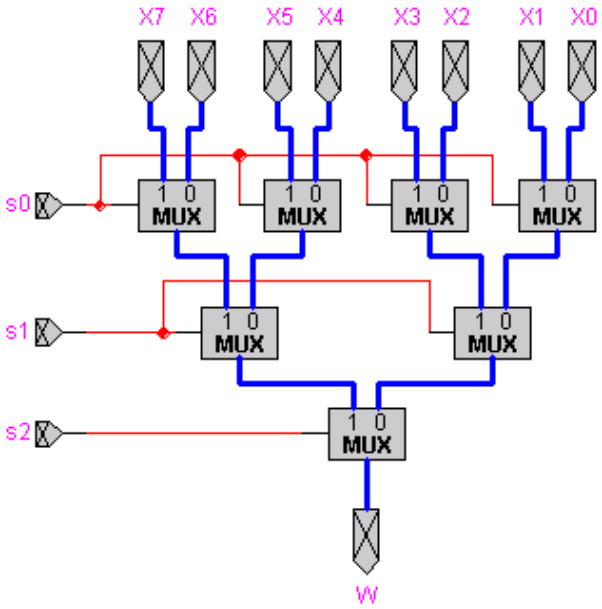
16 Bit-wide 8-to-1 Multiplexer — MUX-8-1

Description:
The 16-bit output W is one of the eight 16-bit inputs X0 to X7 depending on the value of the 3-bit selection input s2, s1, s0:
W = Xk, where k = s2 * 4 + s1 * 2 + s0



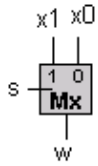
Truth Table (compressed):

s2	s1	s0	W
0	0	0	X0
0	0	1	X1
0	1	0	X2
0	1	1	X3
1	0	0	X4
1	0	1	X5
1	1	0	X6
1	1	1	X7



2-to-1 Multiplexer — Mx-2-1

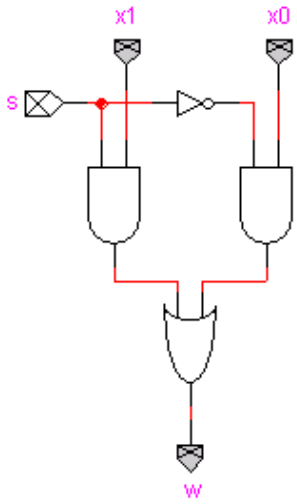
Description:
The binary output w is either x1 or x0, depending on the value of the binary input selection s: If (s = 1) then w = x1 else w = x0.



Bit-level Logical Operation:
w = s * x1 + !s * x0

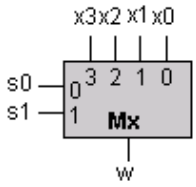
Truth Table (compressed):

s	w
0	x0
1	x1



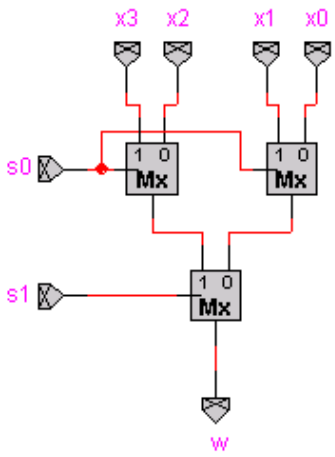
4-to-1 MULTIPLEXER Mx-4-1

Description:
The binary output w is one of the four binary inputs x0 to x3 depending on the value of the 2-bit selection input s1, s0:
w = xk, where k = s1 * 2 + s0



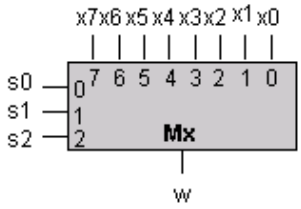
Truth Table (compressed):

s1	s0	w
0	0	x0
0	1	x1
1	0	x2
1	1	x3



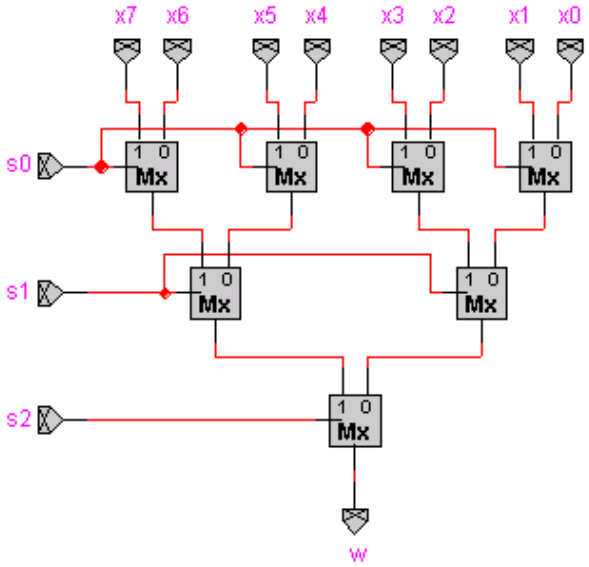
8-to-1 MULTIPLEXER — Mx-8-1

Description:
The binary output w is one of the eight binary inputs x0 to x7 depending on the value of the 3-bit selection input s2, s1, s0:
w = xk, where k = s2 * 4 + s1 * 2 + s0



Truth Table (compressed):

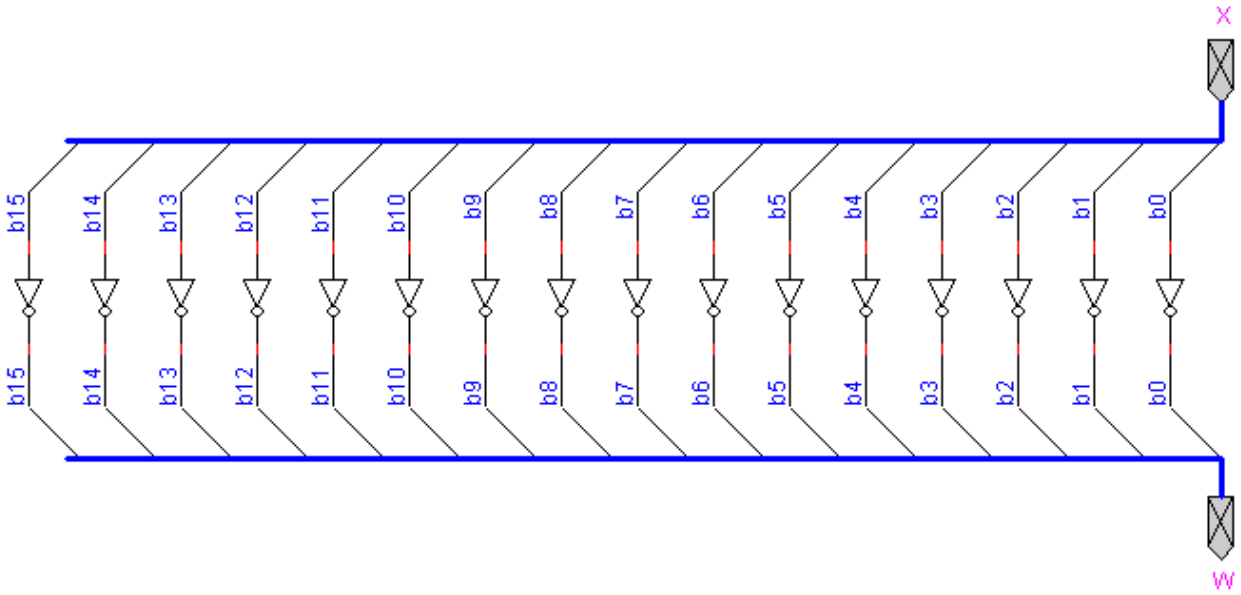
s2	s1	s0	w
0	0	0	x0
0	0	1	x1
0	1	0	x2
0	1	1	x3
1	0	0	x4
1	0	1	x5
1	1	0	x6
1	1	1	x7



16 Bit-wise Not Operator NOT

Description:
The 16-bit output W is the result of a bit-wise logical Not operation of the 16-bit input X.

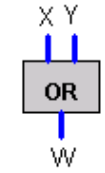
Bit Level Logical Operation:
W(bi) = !X(bi) for i = 0 to 15

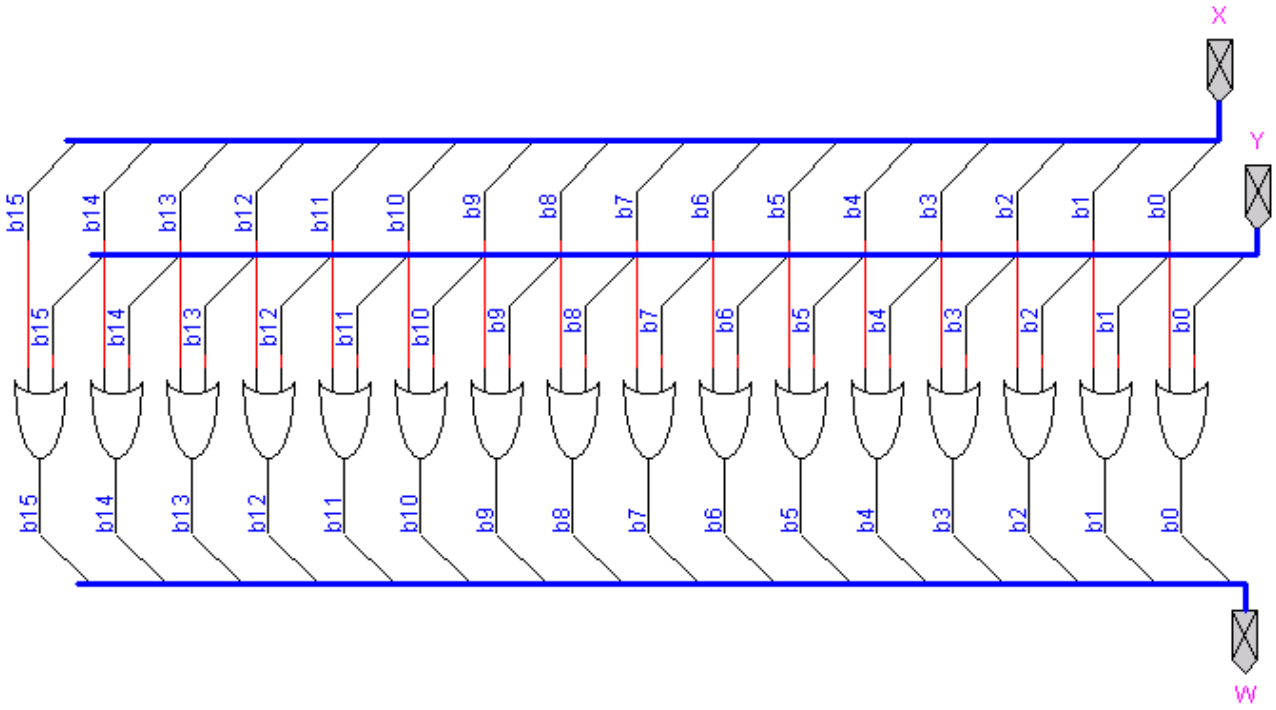


16 Bit-wise Or Operator — OR

Description:
The 16-bit output W is the result of a bit-wise logical Or operation of the 16-bit inputs X and Y.

Bit-level Logical Operation:
W(bi) = X(bi) + Y(bi) for i = 0 to 15

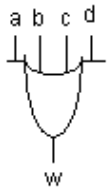


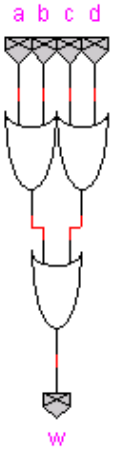


4-Input Or Gate — Or-4

Description:
The binary output w takes the value 1 if one or more of the binary inputs a, b, c or d take the value 1, otherwise (all inputs are 0), w takes the value 0.

Bit-level Logical Operation:
w = a + b + c + d

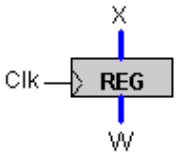




16 Bit-wide Edge-triggered Register — REG

Description:

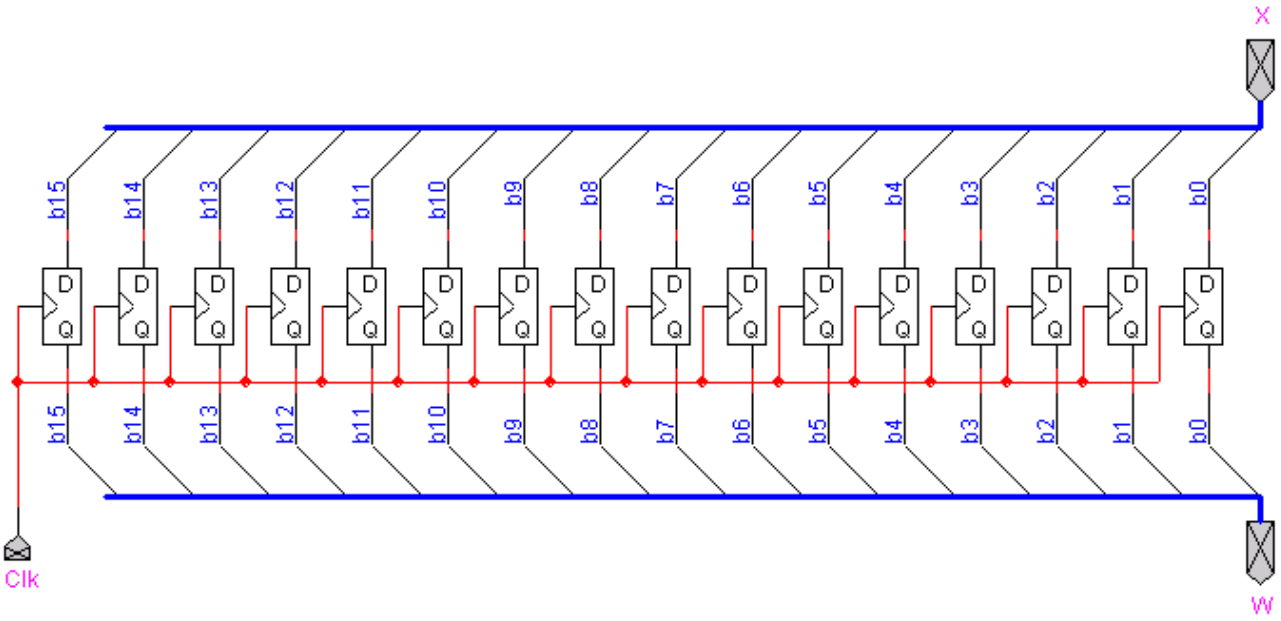
Moore sequential circuit with a 16-bit state. The binary input Clk is the positive edge triggered synchronizing clock. The 16-bit output W is equal to the state Q of the circuit. The next state Q+ is equal to the 16-bit input X. There are no asynchronous inputs.



Next state and Output Logical Expresions:

Q+ = X

W = Q

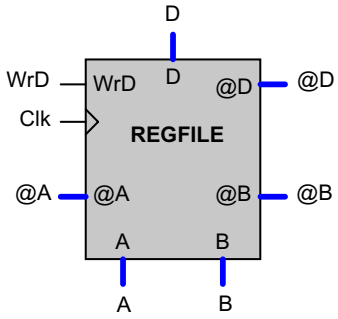


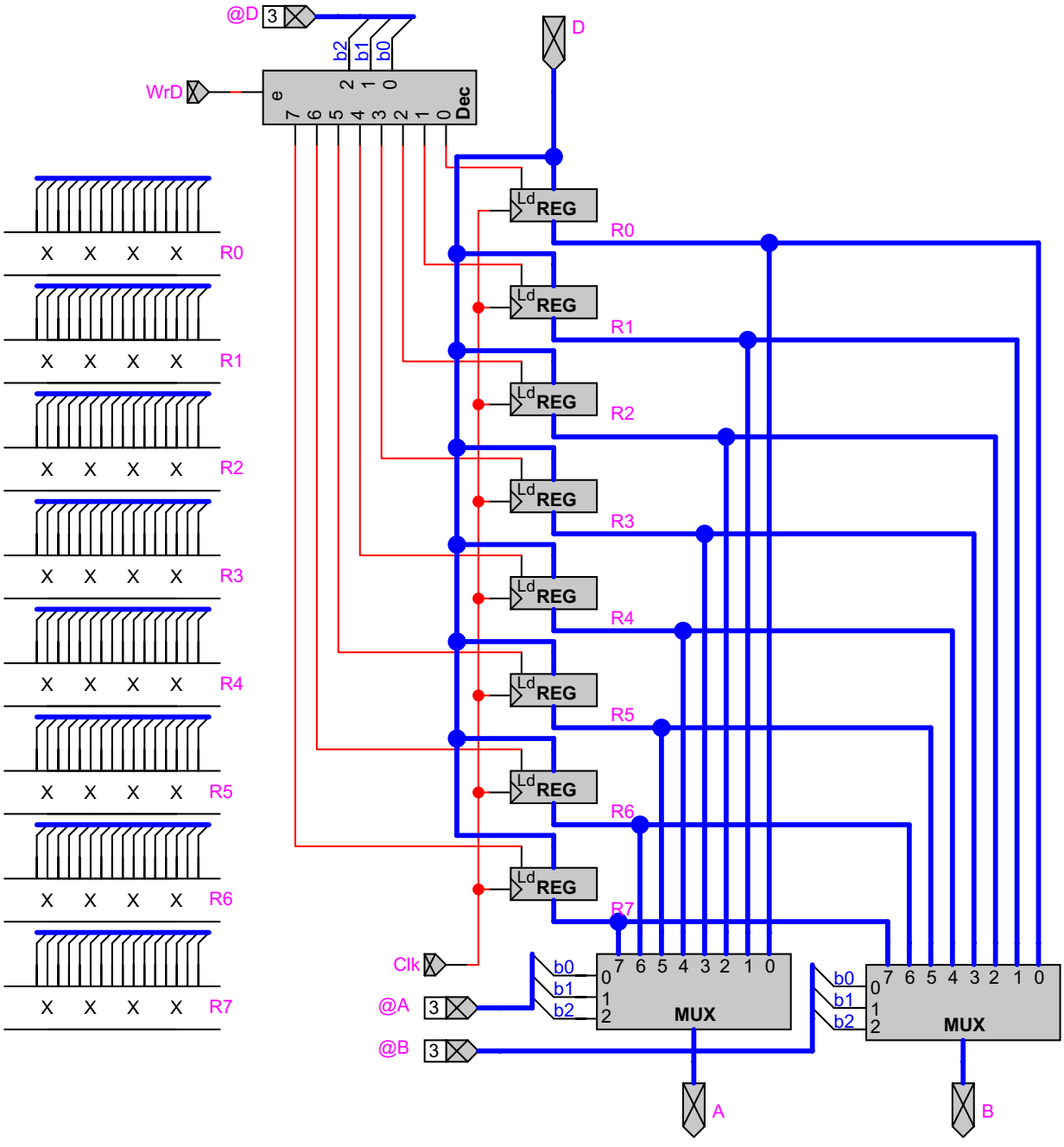
8-by-16 Bit-wide Register File with 1 Write and 2 Read Ports — REGFILE

Description:

Register file with 8 16 Bit-wide registers with 2 read ports, A and B, and 1 write port D. @A, @B and @D are the 3-bit addresses of ports A, B and D respectively. If WrD = 1 the port D is written into Register @D when the Clk binary input changes from 0 to 1.
A and B are the contents of Registers @A and @B.

For simulation purposes, a display of the contents of each register is available.

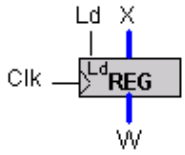




16 Bit-wide Edge-triggered Register with Load Control Signal — REGwLd

Description:

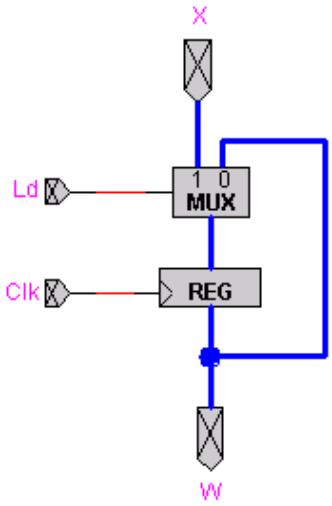
Moore sequential circuit with a 16-bit state. The binary input Clk is the positive edge triggered synchronizing clock. The 16-bit output W is equal to the state Q of the circuit. The next state Q+ is equal to the actual state Q if the binary input Ld is equal to 0 and it is equal to the 16-bit input vector X if Ld is equal to 1. There are no asynchronous inputs.



Next state and Output Logical Expresions:

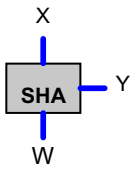
Q+ = Q * !Ld + X * Ld

W = Q



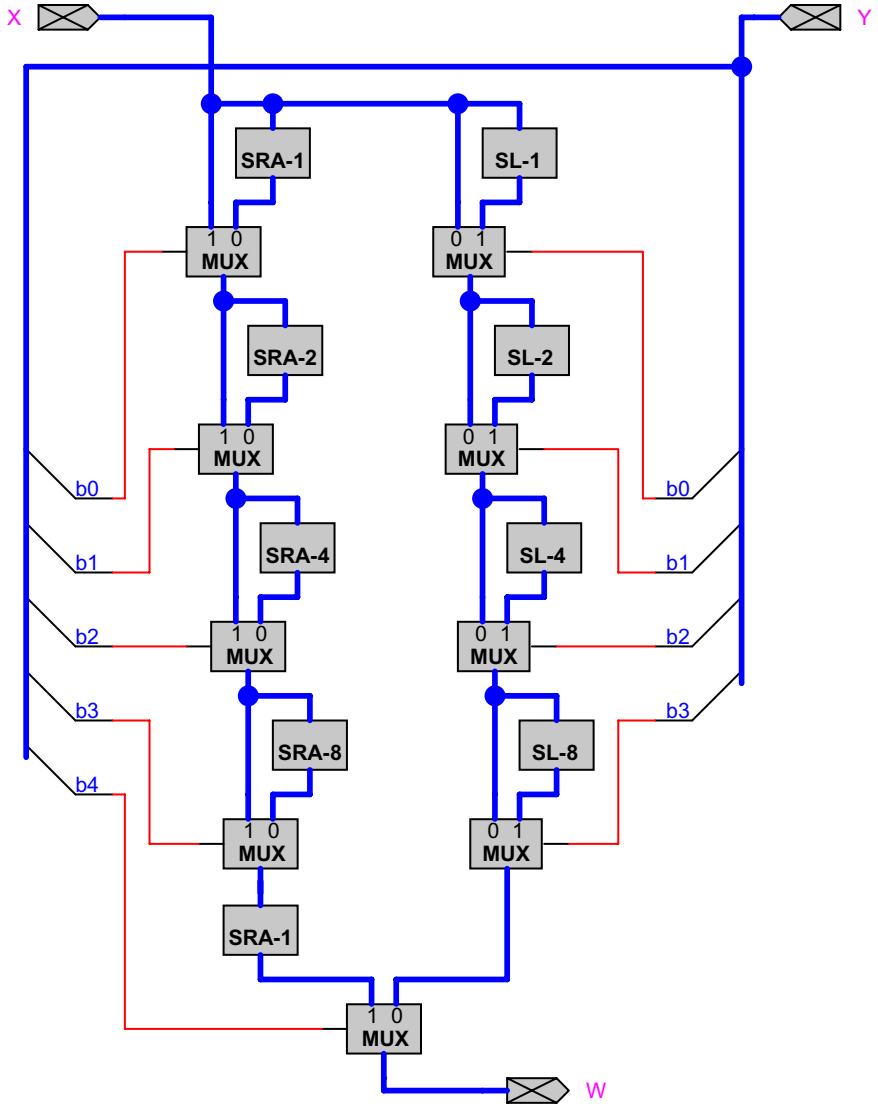
16 Bit-wide Arithmetic Shifter SHA

Description:
The 16-bit output W is the result of shifting the 16-bit input X the number of bits coded by bits 0 to 4 of the 16-bit input Y. Y(b4:b0) represents a signed integer in two´s-complement.The number of shifting bits is the absolute value of the number represented in Y. If this value is positive, the shifting is to the left and if it is negative the shifting is to the right. When SHA shifts to the left the less significant bits of W are set to 0 and the most significant bits of X are lost. When SHA shifts to the right bit X(15) is sign-extended into the most significant bits of W and the less significant bits of X are lost.



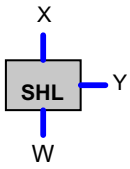
Arithmetic Operation:
Interpreting X, Y(b4:b0) and W as signed integers,
if ((-2**15) <= (Xs * (2**Y(b4:b0)s)) <= (2**15)-1) then Ws = Xs * (2**Y(b4:b0)s)
(The integer división of powers of 2 (when Y(b4:b0)s is negative) is done whith positive remainder)

Bit-level Logical Operation:
if (Y(b4:b0)s >= 0) then
W(bi) = 0 for i = 0 to Y(b4:b0)s -1
W(bi) = X(bi-Y(b4:b0)s) for i = Y(b4:b0)s to 15
if (Y(b4:b0)s < 0) then
W(bi) = X(bi-Y(b4:b0)s) for i = 0 to -Y(b4:b0)s - 1
W(bi) = X(b15) for i = -Y(b4:b0)s to 15



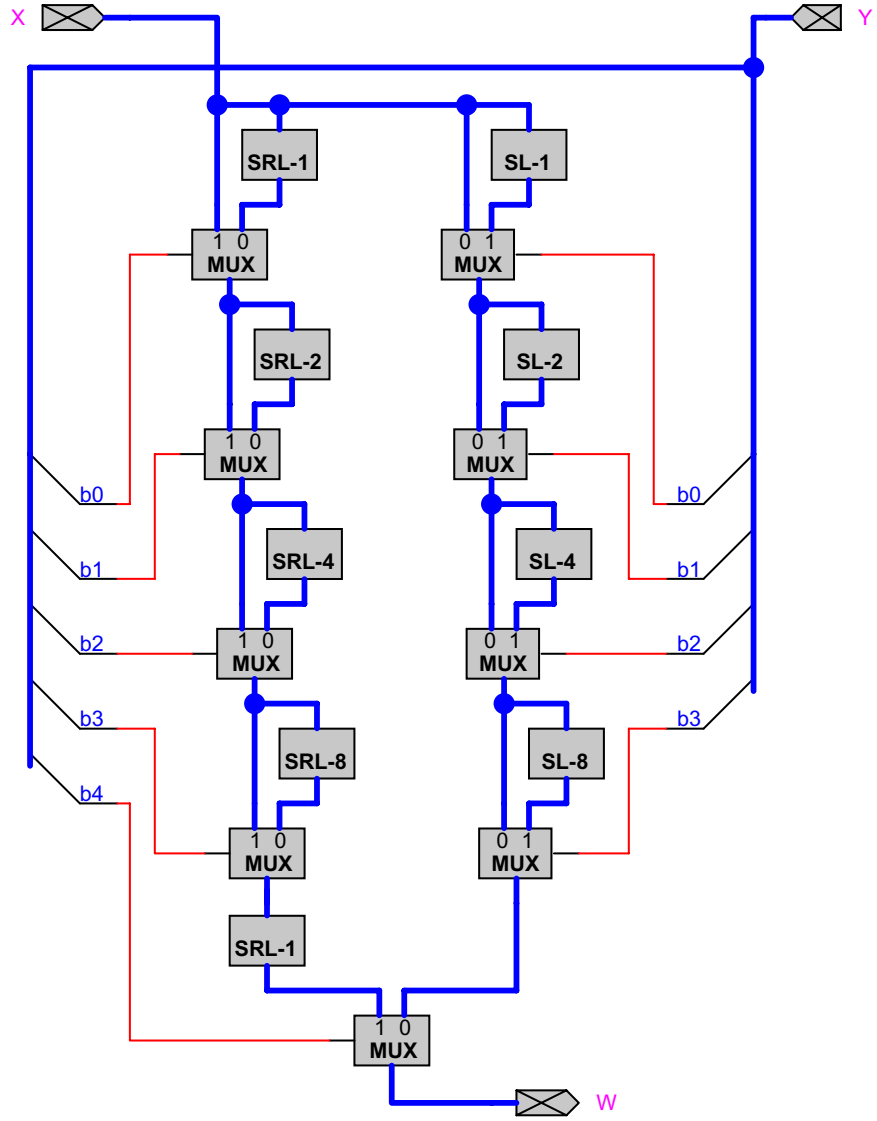
16 Bit-wide Logic Shifter — SHL

Description:
The 16-bit output W is the result of shifting the 16-bit input X the number of bits coded by bits 0 to 4 of the 16-bit input Y. Y(b0:b4) represents a signed integer in two´s-complement.The number of shifting bits is the absolute value of the number represented in Y. If this value is positive, the shifting is to the left and if it is negative the shifting is to the right. When SHL shifts to the left the less significant bits of W are set to 0 and the most significant bits of X are lost. When SHL shifts to the right the most significant bits of W are set to 0 and the less significant bits of X are lost.



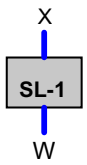
Arithmetic Operation:
Interpreting X and W as unsigned integers and Y as signed integer
if ((Xu * (2**Y(b0:b4)s)) <= (2**16)-1)) then Wu = Xu * (2**Y(b0:b4)s)

Bit-level Logical Operation:
if (Y(b0:b4)s >= 0) then
W(bi) = 0 for i = 0 to Y(b0:b4)s -1
W(bi) = X(bi-Y(b0:b4)s) for i = Y(b0:b4)s to 15
if (Y(b0:b4)s < 0) then
W(bi) = X(bi-Y(b0:b4)s) for i = 0 to -Y(b0:b4)s - 1
W(bi) = 0 for i = -Y(b0:b4)s to 15



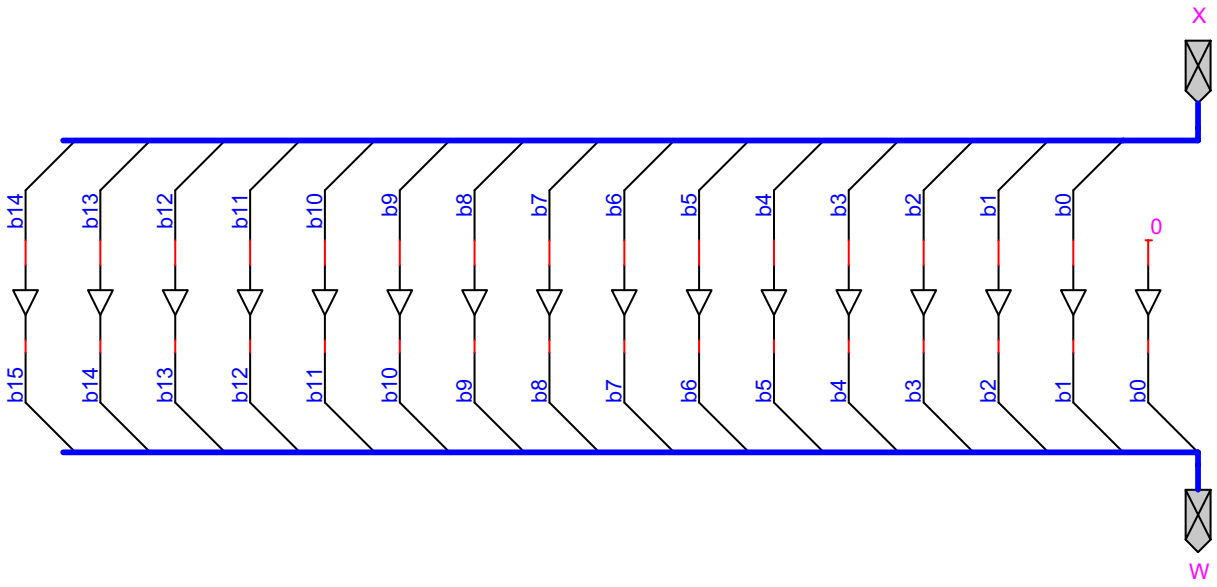
16 Bit-wide 1-Bit Left Shifter — SL-1

Description:
The 16-bit output W is the result of shifting the 16-bit input X one bit to the left. Bit W(b0) is set to 0. Bit X(b15) is lost.



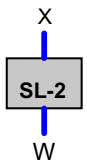
Arithmetic Operation:
Interpreting X and W as unsigned integers,
if ((Xu * 2) <= ((2**16)-1)) then Wu = Xu * 2
Interpreting X and W as signed integers,
if ((-2**15) <= (Xs * 2) <= ((2**15)-1)) then Ws = Xs * 2

Bit-level Logical Operation:
W(b15) = 0
W(bi) = X(bi-1) for i = 1 to 15



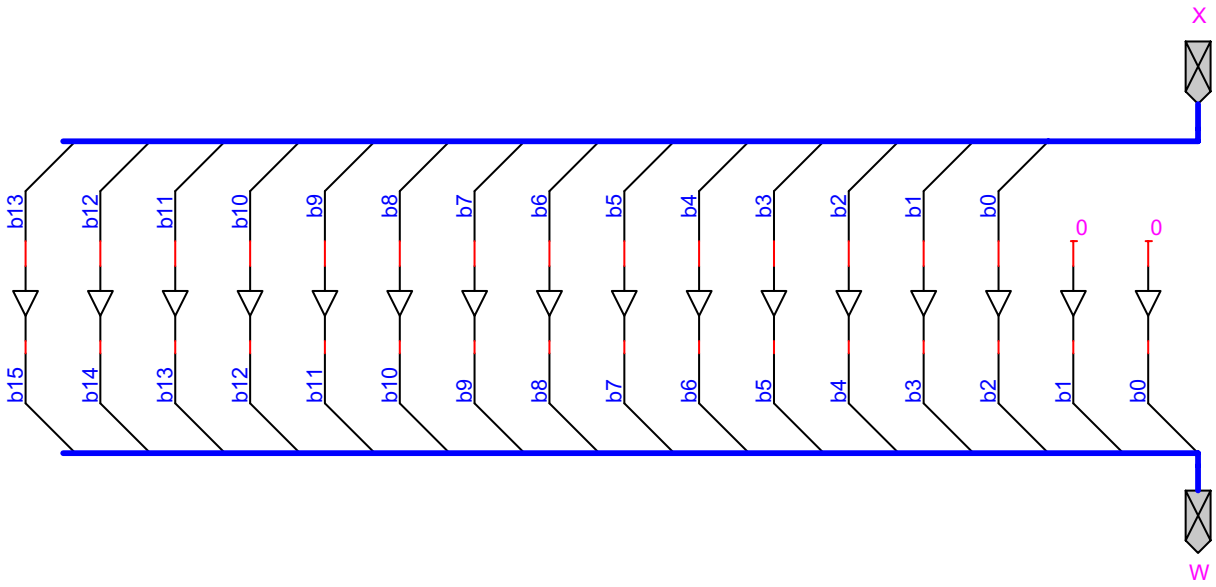
16 Bit-wide 2-Bit Left Shifter — SL-2

Description:
The 16-bit output W is the result of shifting the 16-bit input X two bits to the left. Bits W(b0) and W(b1) are set to 0. Bits X(b14) and X(b15) are lost.



Arithmetic Operation:
Interpreting X and W as unsigned integers,
if ((Xu * 4) <= ((2**16)-1)) then Wu = Xu * 4
Interpreting X and W as signed integers,
if ((-2**15) <= (Xs * 4) <= ((2**15)-1)) then Ws = Xs * 4

Bit-level Logical Operation:
W(bi) = 0 for i = 0 to 1
W(bi) = X(bi-2) for i = 2 to 15

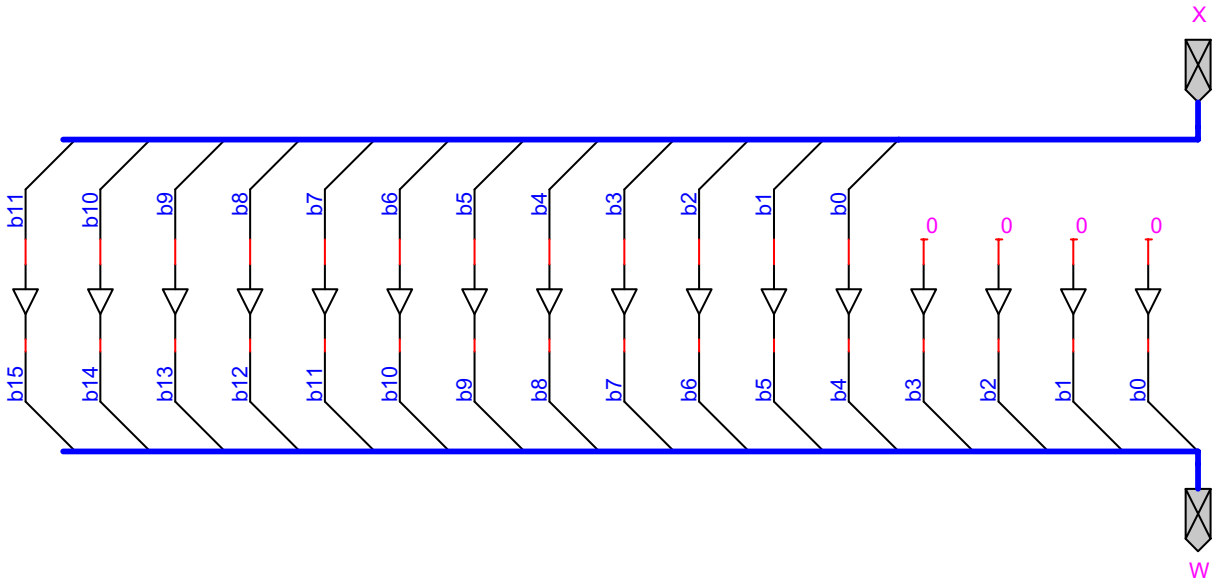


16 Bit-wide 4-Bit Left Shifter SL-4

Description:
The 16-bit output W is the result of shifting the 16-bit input X four bits to the left. Bits W(b0) to W(b3) are set to 0. Bits X(b12) to X(b15) are lost.

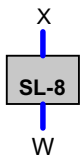
Arithmetic Operation:
Interpreting X and W as unsigned integers,
if ((Xu * 16) <= ((2**16)-1)) then Wu = Xu * 16
Interpreting X and W as signed integers,
if ((-2**15) <= (Xs * 16) <= ((2**15)-1)) then Ws = Xs * 16

Bit-level Logical Operation:
W(bi) = 0 for i = 0 to 3
W(bi) = X(bi-4) for i = 4 to 15



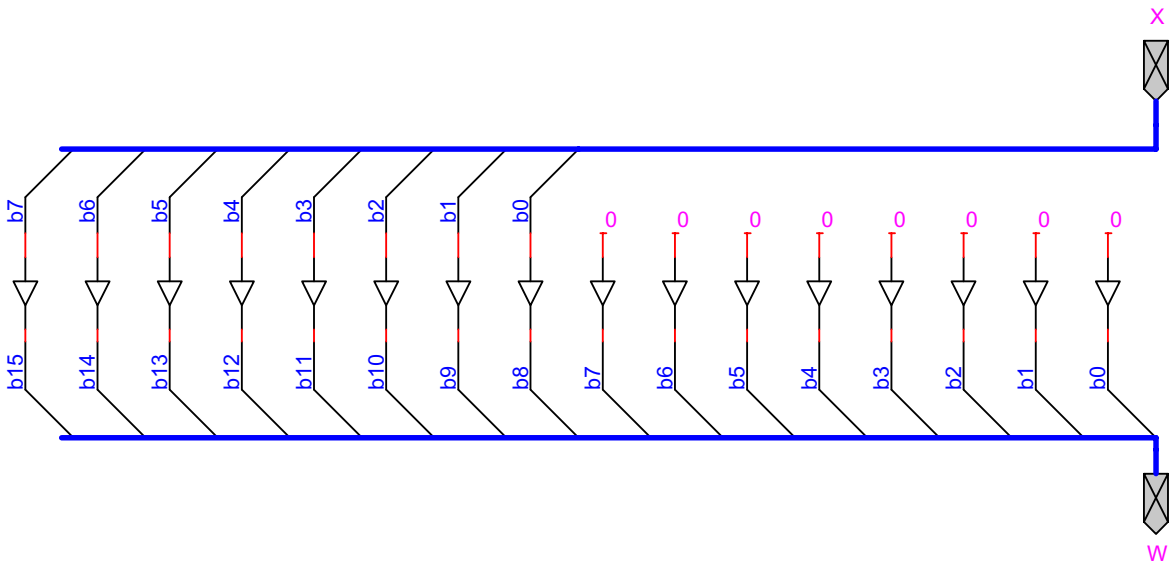
16 Bit-wide 8-Bit Left Shifter — SL-8

Description:
The 16-bit output W is the result of shifting the 16-bit input X eight bits to the left. Bits W(b0) to W(b7) are set to 0. Bits X(b8) to X(b15) are lost.



Arithmetic Operation:
Interpreting X and W as unsigned integers,
if ((Xu * 256) <= ((2**16)-1)) then Wu = Xu * 256
Interpreting X and W as signed integers,
if ((-2**15) <= (Xs * 256) <= ((2**15)-1)) then Ws = Xs * 256

Bit-level Logical Operation:
W(bi) = 0 for i = 0 to 7
W(bi) = X(bi-8) for i = 8 to 15

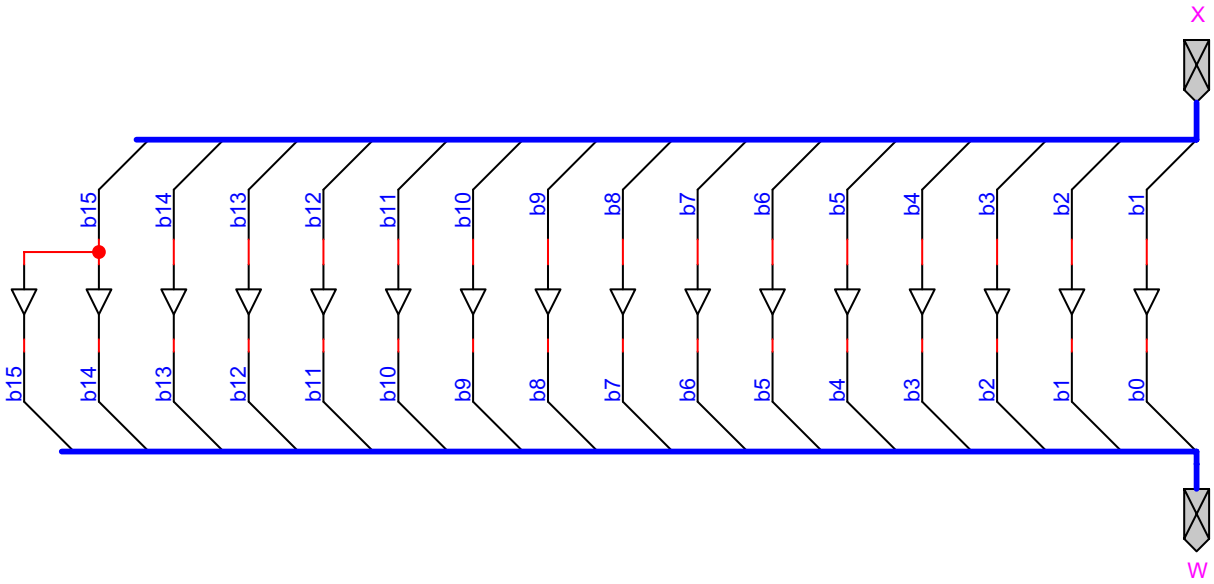


16 Bit-wide 1-Bit Right Arithmetic Shifter SRA-1

Description:
The 16-bit output W is the result of shifting the 16-bit input X one bit to the right extending the sign bit X(b15) into W(b15). Bit X(b0) is lost.

Arithmetic Operation:
If ((-2**15) <= (Xs/2) <= (2**15)-1) then Ws = Xs / 2
(The quotient of the integer division by 2 with positive reminder)

Bit-level Logical Operation:
W(bi) = X(bi+1) for i = 0 to 14
W(b15) = X(b15)

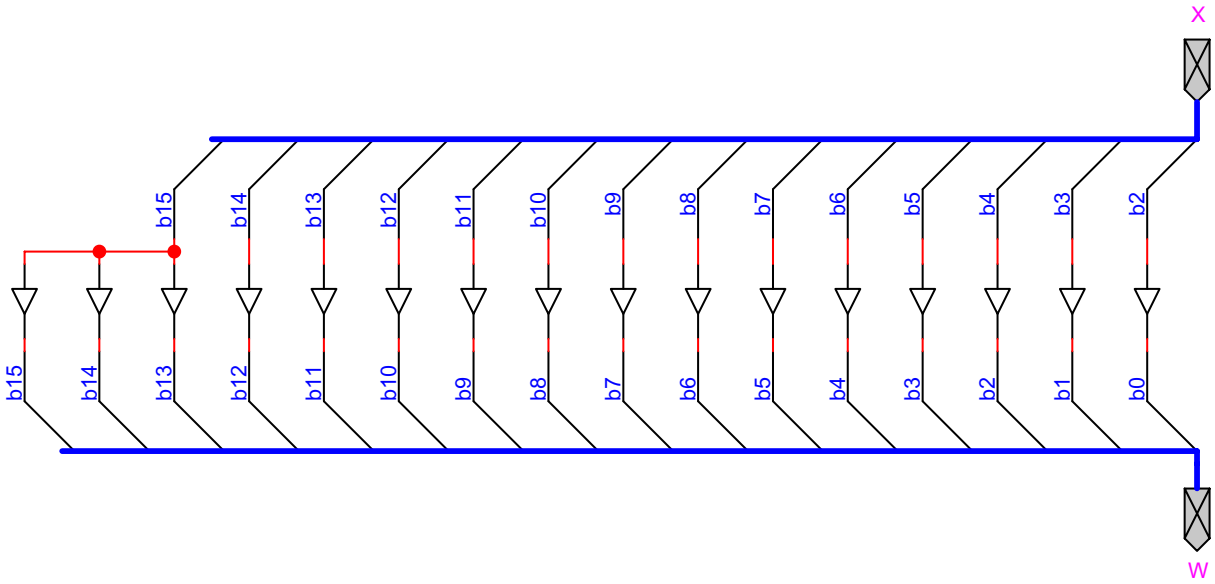


16 Bit-wide 2-Bit Right Arithmetic Shifter — SRA-2

Description:
The 16-bit output W is the result of shifting the 16-bit input X two bits to the right extending the sign bit X(b15) into bits W(b14) and W(b15). Bits X(b0) and X(b1) are lost.

Arithmetic Operation:
If ((-2**15) <= (Xs/4) <= (2**15)-1) then Ws = Xs / 4
(The quotient of the integer division by 4 with positive reminder)

Bit-level Logical Operation:
W(bi) = X(bi+2) for i = 0 to 13
W(bi) = X(b15) for i = 14 to 15

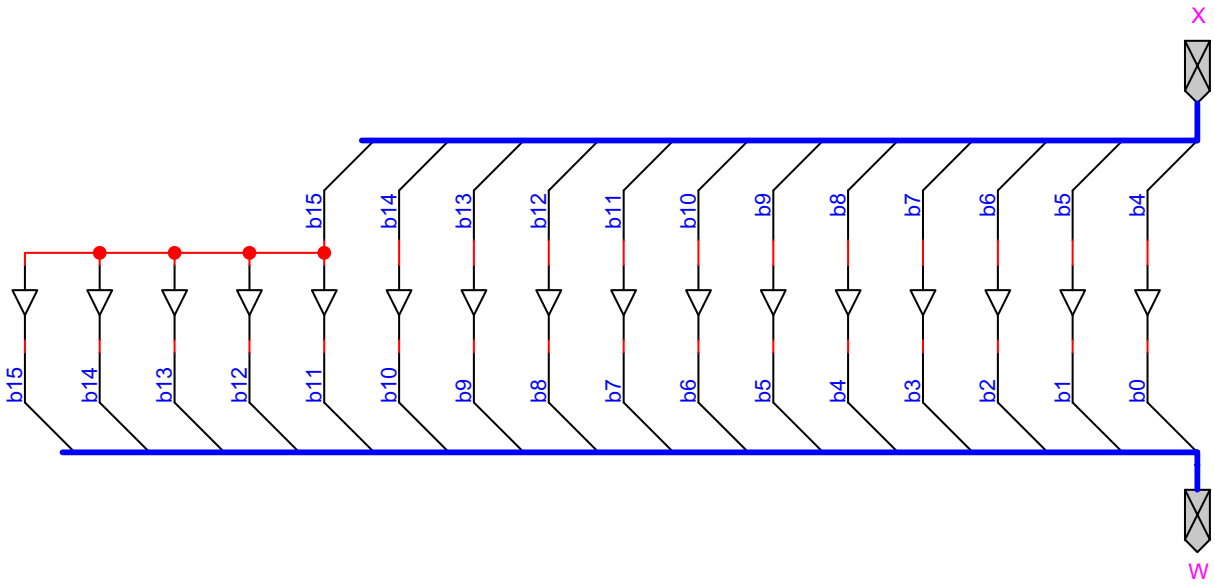


16 Bit-wide 4-Bit Right Arithmetic Shifter — SRA-4

Description:
The 16-bit output W is the result of shifting the 16-bit input X four bits to the right extending the sign bit X(b15) into bits W(b12) to W(b15). Bits X(b0) and X(b3) are lost.

Arithmetic Operation:
If ((-2**15) <= (Xs/16) <= (2**15)-1) then Ws = Xs / 16
(The quotient of the integer division by 16 with positive reminder)

Bit-level Logical Operation:
W(bi) = X(bi+4) for i = 0 to 11
W(bi) = X(b15) for i = 12 to 15

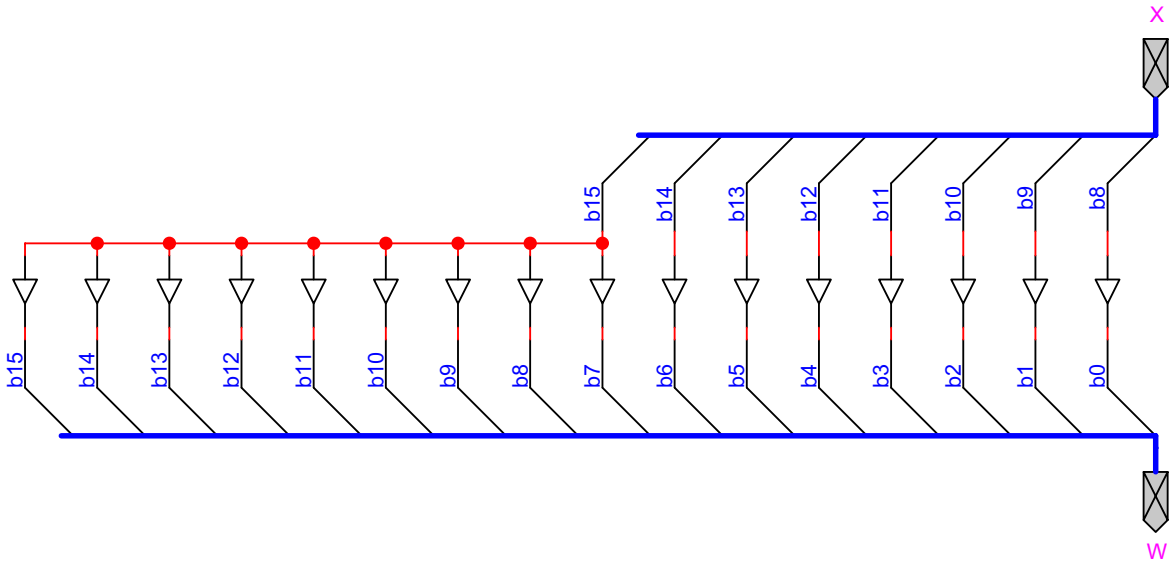


16 Bit-wide 8-Bit Right Arithmetic Shifter — SRA-8

Description:
The 16-bit output W is the result of shifting the 16-bit input X eight bits to the right extending the sign bit X(b15) into bits W(b8) to W(b15). Bits X(b0) and X(b7) are lost.

Arithmetic Operation:
If ((-2**15) <= (Xs/256) <= (2**15)-1) then Ws = Xs / 256
(The quotient of the integer division by 256 with positive reminder)

Bit-level Logical Operation:
W(bi) = X(bi+8) for i = 0 to 7
W(bi) = X(b15) for i = 8 to 15

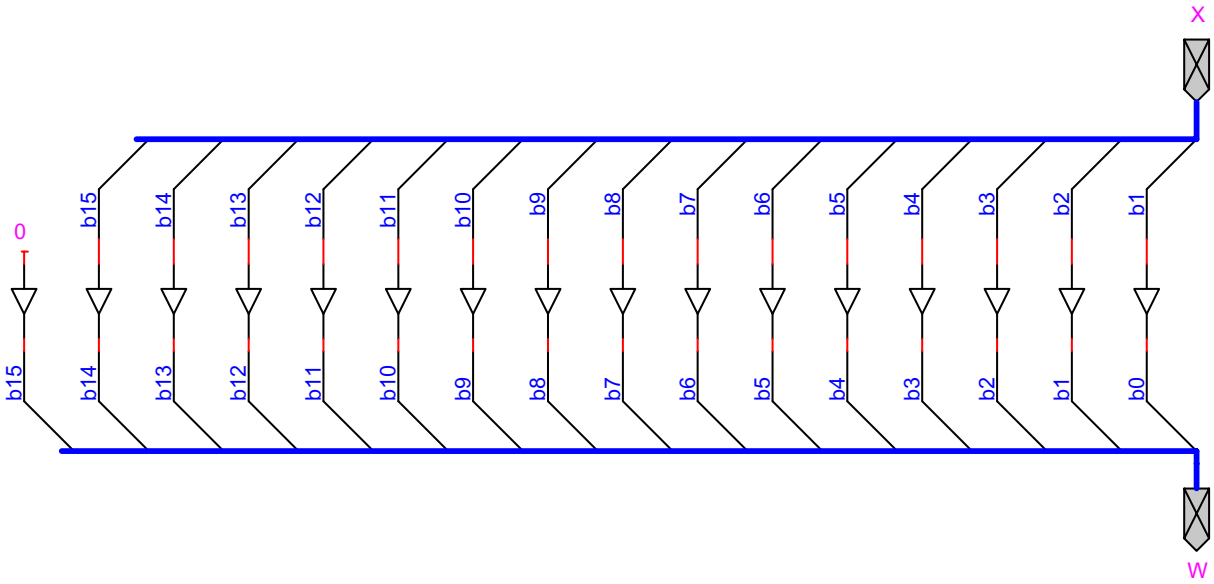


16 Bit-wide 1-Bit Right Logic Shifter SRL-1

Description:
The 16-bit output W is the result of shifting the 16-bit input X one bit to the right.
Bit W(b15) is set to 0. Bit X(b0) is lost.

Arithmetic Operation:
If ((Xu/2) <= ((2**16)-1)) then Wu = Xu / 2 (The quotient of the integer division by 2)

Bit-level Logical Operation:
W(bi) = X(bi+1) for i = 0 to 14
W(b15) = 0

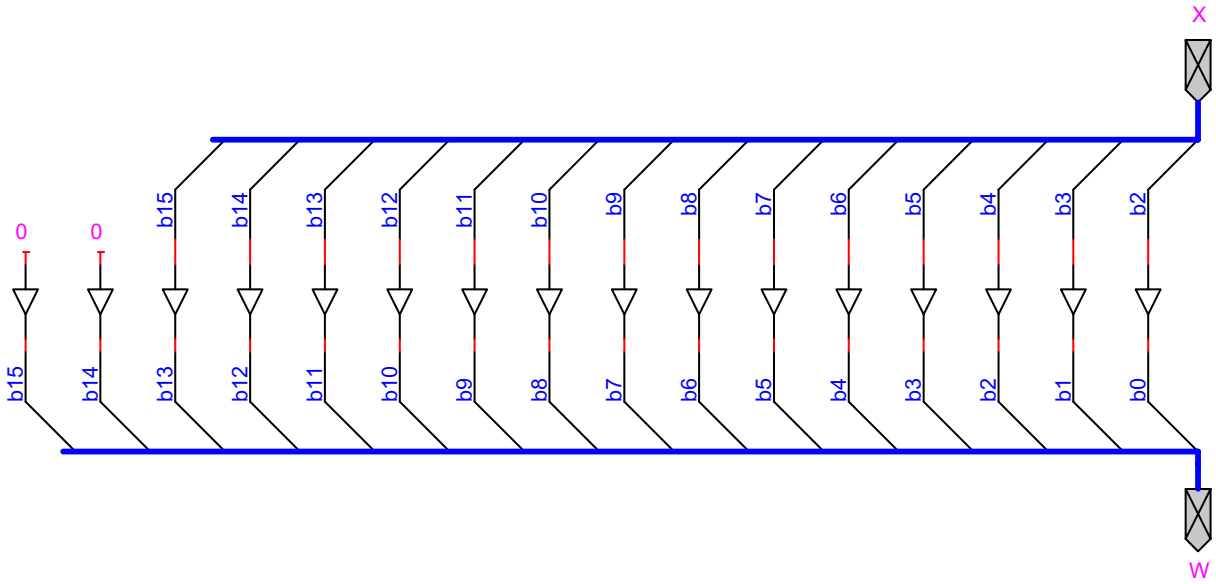


16 Bit-wide 2-Bit Right Logic Shifter — SRL-2

Description:
The 16-bit output W is the result of shifting the 16-bit input X two bits to the right. Bits W(b14) and W(b15) are set to 0. Bits X(b0) and X(b1) are lost.

Arithmetic Operation:
If ((Xu/4) <= ((2**16)-1)) then Wu = Xu / 4 (The quotient of the integer division by 4)

Bit-level Logical Operation:
W(bi) = X(bi+2) for i = 0 to 13
W(bi) = 0 for i = 14 to 15

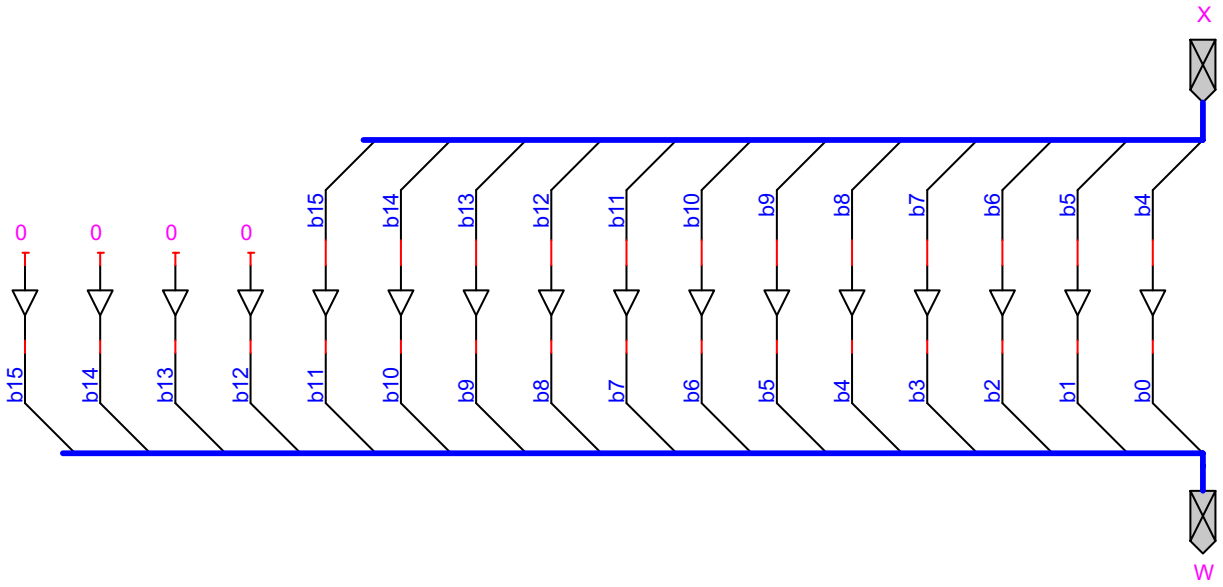


16 Bit-wide 4-Bit Right Logic Shifter — SRL-4

Description:
The 16-bit output W is the result of shifting the 16-bit input X four bits to the right.
Bits W(b12) to W(b15) are set to 0. Bits X(b0) to X(b3) are lost.

Arithmetic Operation:
If ((Xu/16) <= ((2**16)-1)) then Wu = Xu / 16 (The quotient of the integer division by 16)

Bit-level Logical Operation:
W(bi) = X(bi+4) for i = 0 to 11
W(bi) = 0 for i = 12 to 15

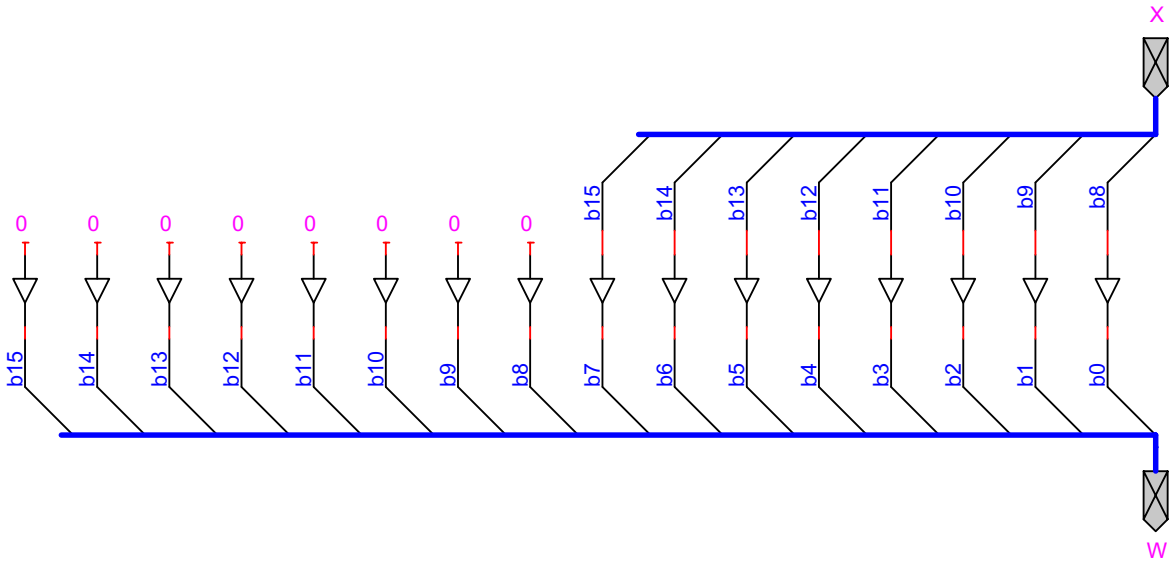


16 Bit-wide 8-Bit Right Logic Shifter — SRL-8

Description:
The 16-bit output W is the result of shifting the 16-bit input X eight bits to the right. Bits W(b8) to W(b15) are set to 0. Bits X(b0) to X(b7) are lost.

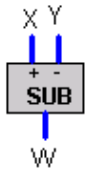
Arithmetic Operation:
If ((Xu/256) <= ((2**16)-1)) then Wu = Xu / 256 (The quotient of the integer division by 256)

Bit-level Logical Operation:
W(bi) = X(bi+8) for i = 0 to 7
W(bi) = 0 for i = 8 to 15



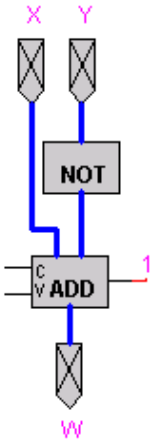
16 Bit-wide Subtractor SUB

Description:
The 16-bit inputs X and Y and the 16-bit output W can be interpreted as signed or as unsigned integers. For both cases the output is equal to the subtraction of the two inputs, except when overflow occurs. No overflow detection is done.



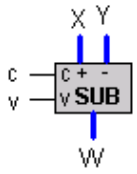
Arithmetic Operation:
Interpreting X, Y and W as unsigned integers,
If ((Xu - Yu) >= 0) then
Wu = Xu - Yu
else
Wu = Xu - Yu + 2**16

Interpreting X, Y and W as signed integers,
if (-2**15 <= (Xs - Ys) <= (2**15)-1) then Ws = Xs - Ys
if ((Xs - Ys) > (2**15)-1) then Ws = Xs - Ys - 2**16
if (-2**15 > (Xs - Ys)) then Ws = Xs - Ys + 2**16



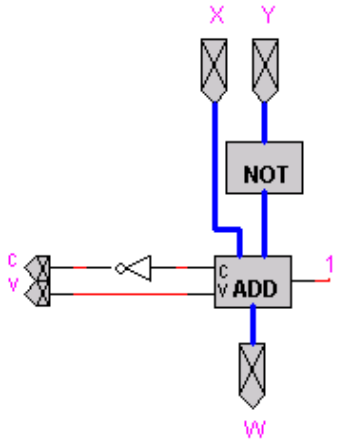
16 Bit-wide Subtractor with Carry and Overflow Outputs — SUBwCV

Description:
The 16-bit inputs X and Y and the 16-bit output W can be interpreted as signed or as unsigned integers. For both cases the output is equal to the subtraction of the two inputs, except when overflow occurs. Overflow is indicated by the binary outputs c and v for unsigned and signed subtraction respectively.



Arithmetic Operation:
Interpreting X, Y and W as unsigned integers,
If ((Xu - Yu) >= 0) then
Wu = Xu - Yu, c = 0
else
Wu = Xu - Yu + 2**16, c = 1

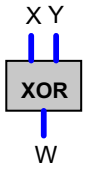
Interpreting X, Y and W as signed integers,
if (-2**15 <= (Xs - Ys) <= (2**15)-1) then Ws = Xs - Ys, v = 0
if ((Xs - Ys) > (2**15)-1) then Ws = Xs - Ys - 2**16, v = 1
if (-2**15 > (Xs - Ys)) then Ws = Xs - Ys + 2**16, v = 1

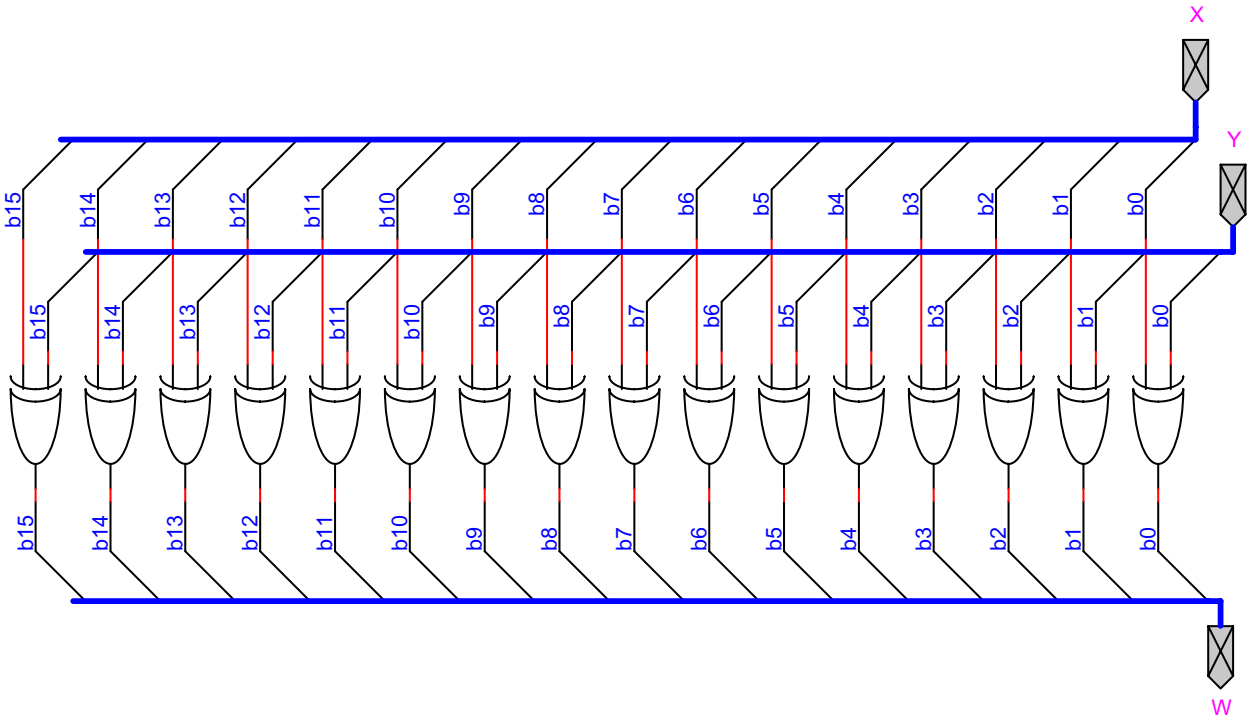


16 Bit-wise Xor Operator — XOR

Description:
The 16-bit output W is the result of a bit-wise logical Xor operation of the 16-bit inputs X and Y.

Bit-level Logical Operation:
W(bi) = X(bi) ^ Y(bi) for i = 0 to 15





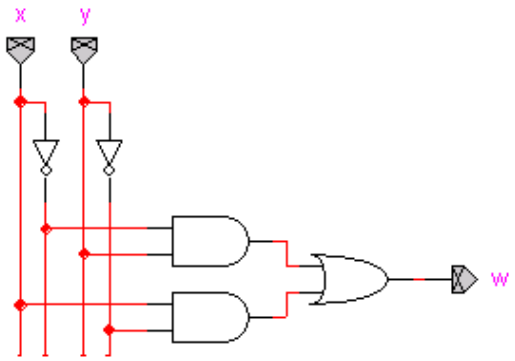
2-Input Exclusive Or Gate — Xor-2

Description:
The binary output w takes the value 1 if only one of the bits x and y takes the value 1, otherwise w takes the value 0.
(Alternative description: if (x=1) then w = !y else w = y)

Bit-level Logical Operation:
w = x ^ y

Truth Table:

x	y	w
0	0	0
0	1	1
1	0	1
1	1	0



Zero Detector — Zero

Description:
The binary output z takes the value 1 if all the bits of the 16-bit input X are 0, otherwise z takes the value 0.

Bit-level Logical Operation:
if (X(bi) = 0 for i = 0 to 15) then z = 1 else z = 0