Facultad de Ingeniería - UNLP

E1301 – E0301 Introducción a los Sistemas Lógicos y Digitales

Curso 2025 - Trabajo Entregable Integrador

Isabella Valenzi Octavio Tomás - 03665/8

Rodriguez Fausto - 03766/2

Cladera Blas - 03694/3

Patané Baltazar - 03773/1

Repositorio de GitHub del grupo

Introducción

En el presente informe se describe el diseño, implementación y simulación de un circuito digital desarrollado en Quartus, cuyo objetivo es implementar un contador UP/DOWN de 12 bits, con incremento variable y límites de conteo programables. El circuito posee las siguientes características:

UP/DOWN: Entrada que define el sentido del conteo.

DATA [11..0]: Bus de datos bidireccional para valor inicial y final.

LOAD_I y LOAD_F: Señales que cargan los valores inicial y final al bus de datos.

INC [2..0]: valor del incremento o decremento en 3 bits.

PAUSA: se activa si el valor de incremento es 0. **Activo en alto.**

FIN: Se activa durante un ciclo de reloj cuando se alcanza el valor final. **Activo en bajo.**

CONTINUAR: En alto reinicia la cuenta y en bajo la detiene.

RECARGAR: En alto detiene el contador y recarga los valores

START: Inicia el funcionamiento del contador.

ERROR: se activa cuando los valores inicial y final son inconsistentes con el sentido. **Activo en bajo.**

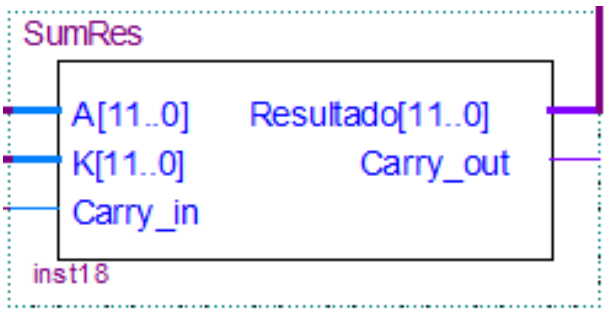
Desarrollo

Para el desarrollo del contador, dividimos inicialmente el trabajo en la creación de un sumador, un comparador, un analizador de corte y un sistema de buses y almacenamiento

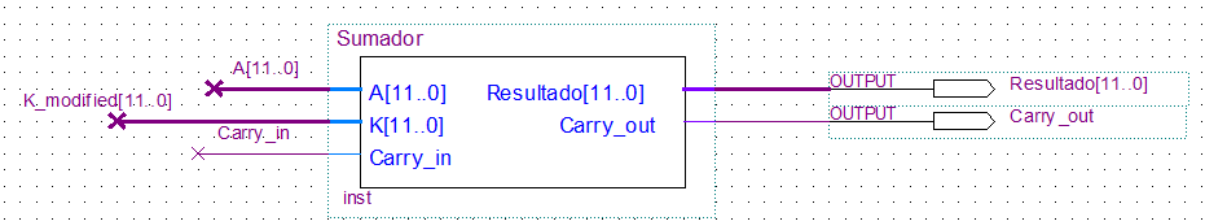
Sumador

El bloque SumRes recibe un bus de datos de 12 bits que representa el primer operando por la entrada “A[11..0]”, otro bus de datos de 12 bits que representa el segundo operando por la entrada “K[11..0]”, y un bit de carry usado para definir si será una operación de suma o resta a partir de la señal UP/DOWN por la entrada “Carry_in”. En este caso, se sumarán los operandos si la entrada es 0, y se restarán si es 1.

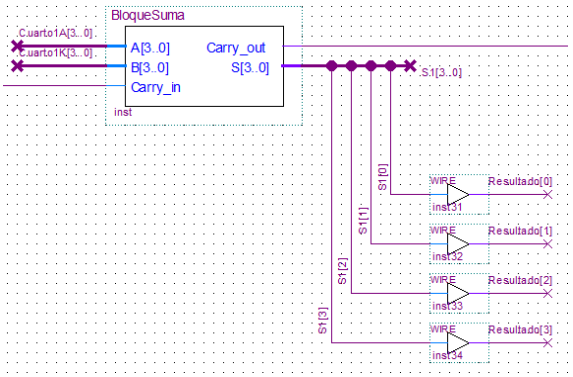
Por otro lado, sus salidas son: otro bus de datos de 12 bits por la salida “Resultado[11..0]” que posee el resultado de la operación aritmética (ya sea suma o resta) y otro bit de carry para mantener la reusabilidad del sumador por la salida “Carry_out”.



La lógica interna del bloque se basa en modificar el signo del número en Ca2 de la entrada K si “Carry_in” fuera 1. Esto significaría sumar un número negativo que sería lo mismo que restarlo. Este bus “K_modified[11..0]” se vuelve la nueva entrada del bloque Sumador.

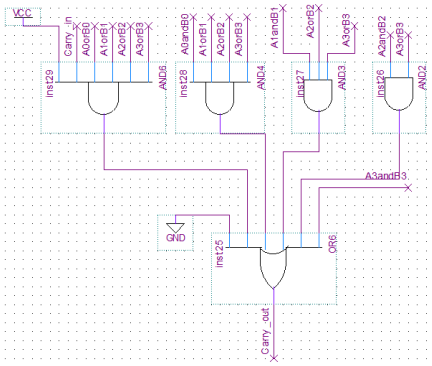


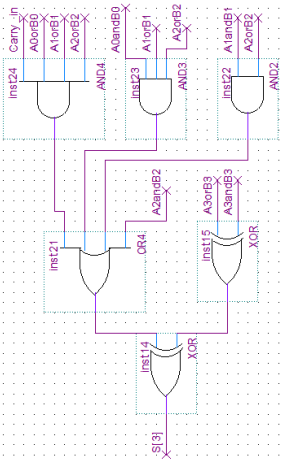
El bloque Sumador se encarga de dividir los bits de los operandos en tres partes con el objetivo de reutilizar el BloqueSuma con cada una de estas partes para que resuelva sus sumas y devuelva todos los bits del resultado al bus de acuerdo a con qué parte de los operandos se está trabajando..

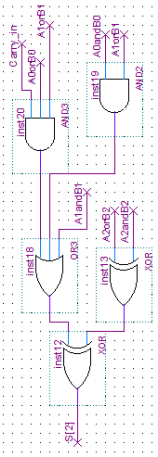


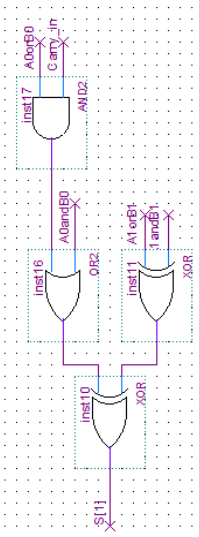
El BloqueSuma se basa en un sumador de 4 bits que suma dos entradas “A[3..0]” y “B[3..0]” junto con un bit de carry inicial “Carry_in”, produciendo una salida de 4 bits “S[3..0]” y un nuevo bit de carry de salida “Carry_out”.

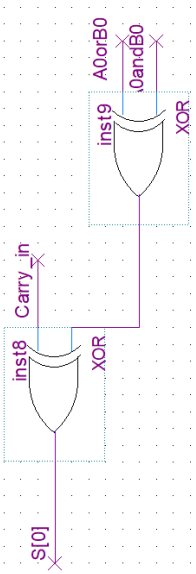
Este sumador se encarga de calcular los bits de carry de una operación antes de que dicha operación comience. Al ser un circuito paralelo se consigue evitar el efecto “cascada” del sumador en serie que trae consigo un retardo proporcional a la cantidad de sumadores. De hecho, lo que se hace es implementar el sumador utilizando tres BloqueSuma conectados en serie, teniendo en cuenta que cada BloqueSuma es un sumador en paralelo de 4 bits. Así es que se aprovecha la ventaja de los sumadores paralelos de no tener un gran retardo y se simplifica el diseño del sumador conectando 3 de estos sumadores paralelos en serie (la escalabilidad del sumador en paralelo se complejiza mucho a más bits se añaden en comparación con el sumador en serie). *Aclaración: utilizamos el sumador en paralelo explicado en las diapositivas de teoría.*







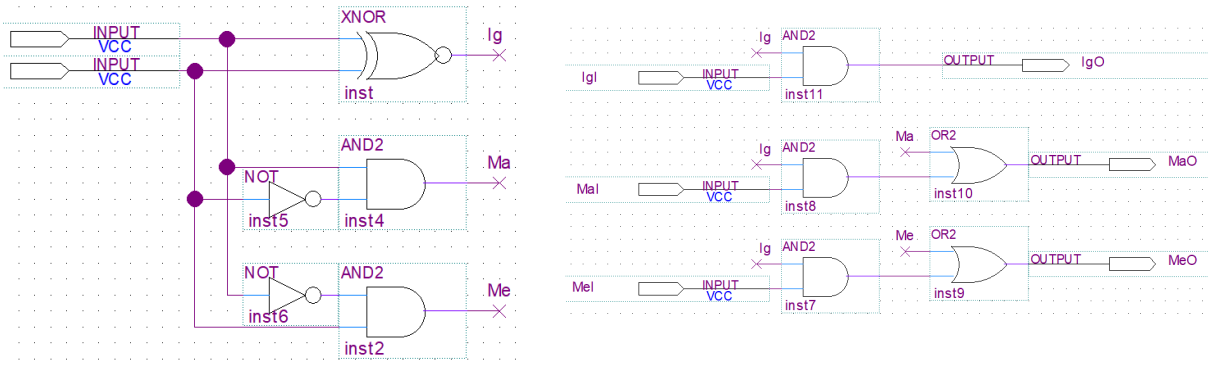




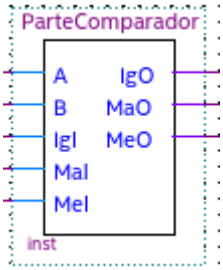
Comparador

La idea fue sencilla, para comparar dos números simplemente te fijas en su dígito más significativo, este puede indicar que uno de los dos números es mayor o si son iguales| vas al siguiente bit.

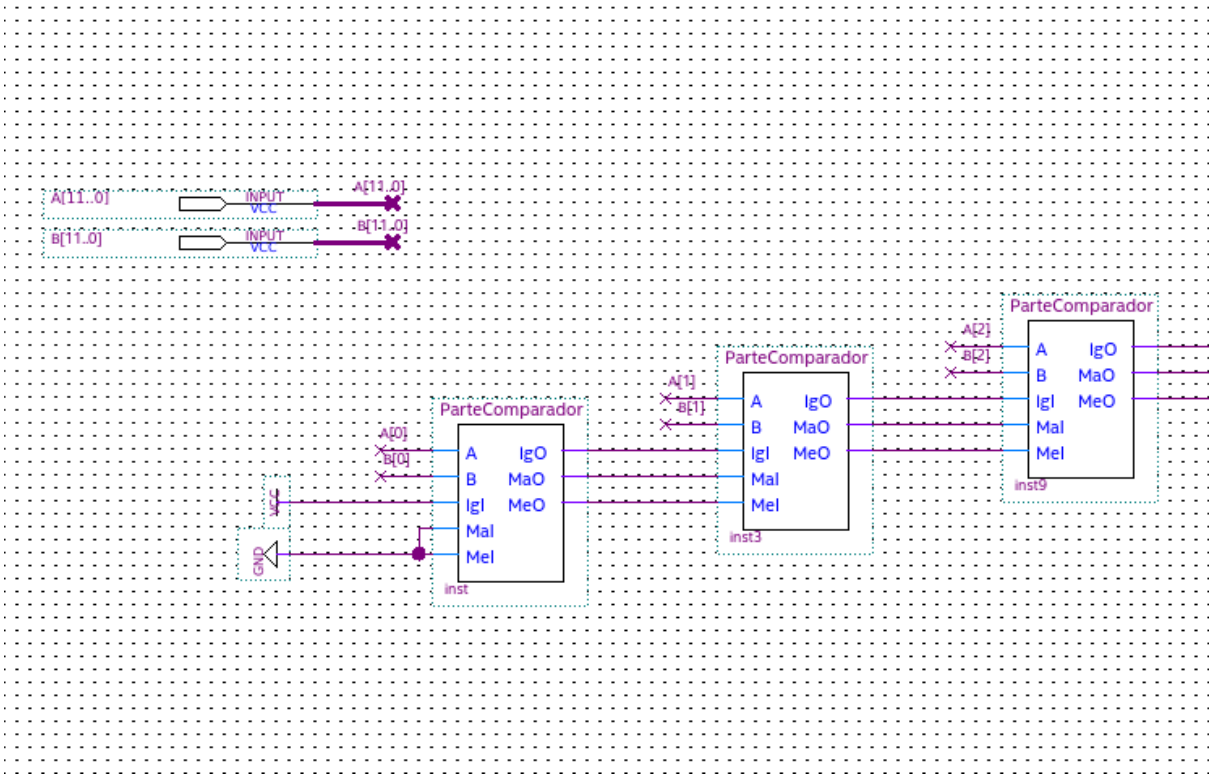
Así con esta idea sencilla hago un bloque para comparar 2 bits. Si se indica un resultado se manda para los bits más significativos, si son iguales se fija lo que reciba de los bits menos significativos.

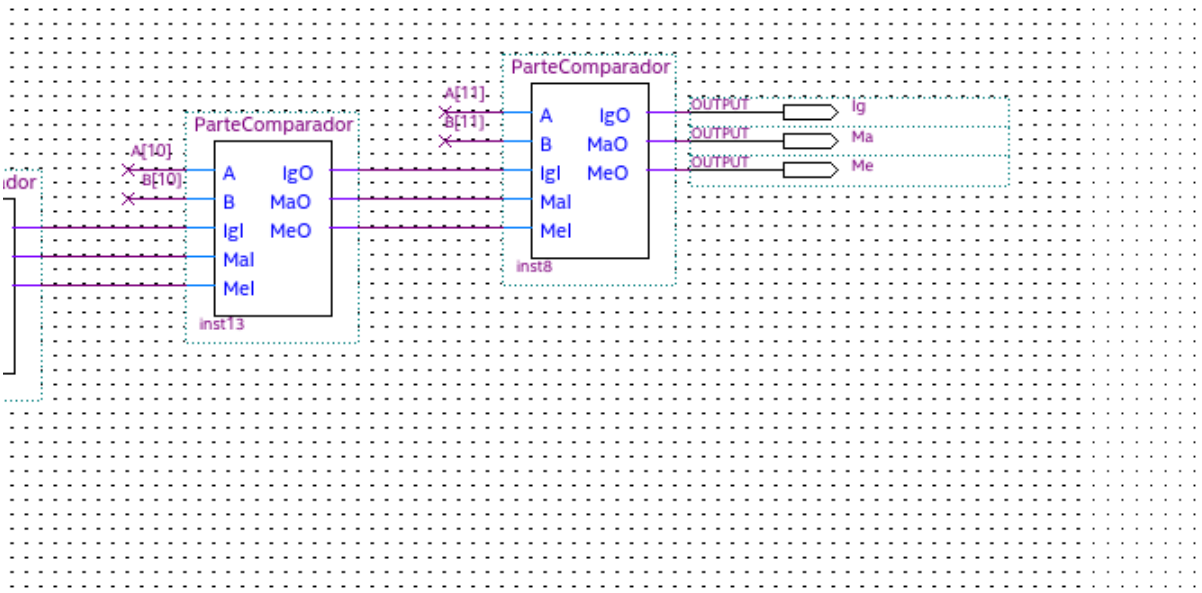


Esta parte quedaría como la siguiente imagen:



Que todo conectado en serie haría al comparador

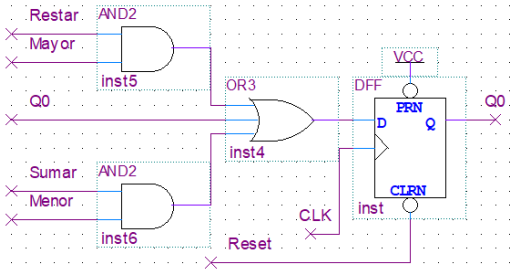


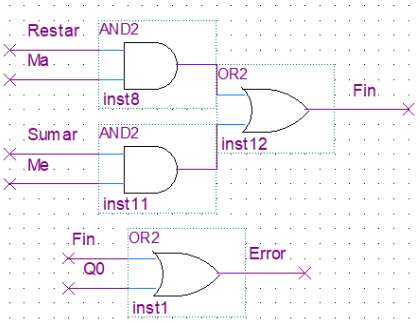


Analizador de Corte

El bloque Analizador de Corte tiene que indicar si hay que seguir contando o si es necesario lanzar un error. Este último ocurre cuando directamente no cumple para la primera instancia que sea mayor/menor dependiendo el caso, ya superando la primera instancia de comparación se debe indicar de alguna forma que la primera instancia fue superada.

A partir de esto se llega a la conclusión de que hay que usar un FF tipo D que sea usado para ver si fue superada la primera instancia.





Haciendo una tabla, viendo como son las entradas y salidas se termina haciendo el sistema para cumplir esto

	Q0	Restar	Mayor	Menor	Q0+	Error	Fin	D0
Caso Resta	0	1	1	0	1	1	1	1
	1	1	1	0	1	1	1	1
	1	1	0	X	1	1	0	0
Caso Suma	0	0	0	1	1	1	1	1
	1	0	0	1	1	1	1	1
	1	0	X	0	1	1	0	0
	0	1	0	X	0	0	0	0
	0	0	X	0	0	0	0	0

D0

Q0 | Restar	Mayor "00"	Mayor "01"	Mayor "11"	Mayor "10"
"00"	0	1	0	0
"01"	0	0	0	1
"11"	1	1	0	1
"10"	1	1	0	1

D0 = (Q0 * ~Me) + (Q0 * ~Ma) + (Re * Ma * ~Me) + (~Re * ~Ma * Me)
D0 = (Q0 * ~Me) + (Q0 * ~Ma) + (Re * Ma) + (~Re * Me)

D0 = Q0 + (Re*Ma) + (Re *Me)

~Error

Q0 | Restar	Mayor "00"	Mayor "01"	Mayor "11"	Mayor "10"
"00"	1	0	0	1
"01"	1	1	0	0
"11"	0	0	0	0
"10"	0	0	0	0

~Error = ~Q0 * ~Re * ~Me + ~Q0 * Re * ~Ma
Error = (Q0 + Re + Me) * (Q0 + ~Re + Ma)

Error = (Q0) + (Re * Q0) + (Me * Q0) + (~Re * Q0) + (Me * ~Re) + (Re * Ma)
Error = (Q0) + (Me * ~Re) + (Re * Ma)

Fin

Q0 | Restar	Mayor "00"	Mayor "01"	Mayor "11"	Mayor "10"
"00"	0	1	0	0
"01"	0	0	0	1
"11"	0	0	0	1
"10"	0	0	0	0

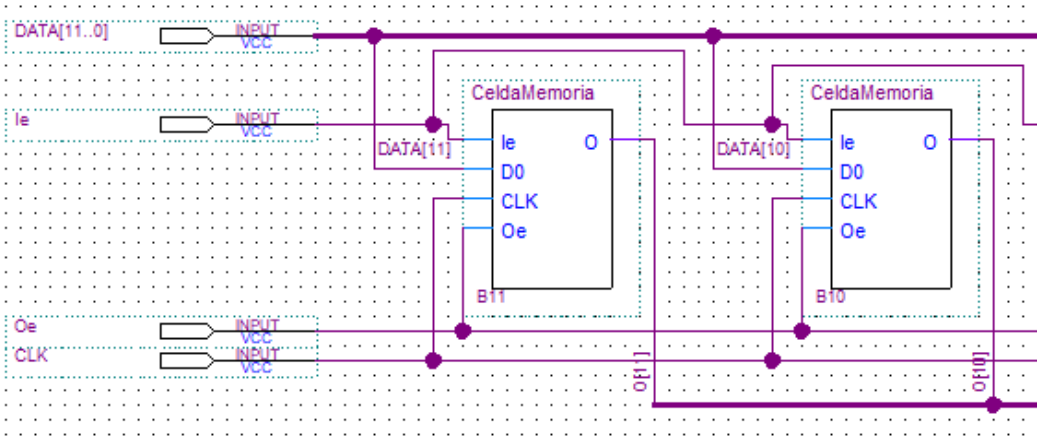
Fin = Restar * Mayor * ~Menor + ~Restar * ~Mayor * Menor
Fin = Re * Ma + ~Re * Me

Buses y Almacenamiento

El bloque Registro12 es un registro de 12 bits compuesto por 12 instancias del módulo CeldaMemoria. Este módulo actúa como un registro de almacenamiento paralelo sincrónico que permite la lectura y escritura controlada de un bus de datos de 12 bits.

El registro recibe el bus de datos “DATA[11..0]”, la señal de habilitación de entrada “Ie”, la señal de habilitación de salida ”Oe” y la señal del reloj “CLK”. En el momento que lo requiera, envía los datos a la salida a través del bus “O[11..0]”.

El funcionamiento del registro depende de Ie. Cuando este se encuentra en alto, los datos presentes en DATA se escriben en las celdas de memoria en el flanco de subida del clock. Por otro lado, cuando Oe está en alto, los datos almacenados en las celdas se envían al bus de salida O.



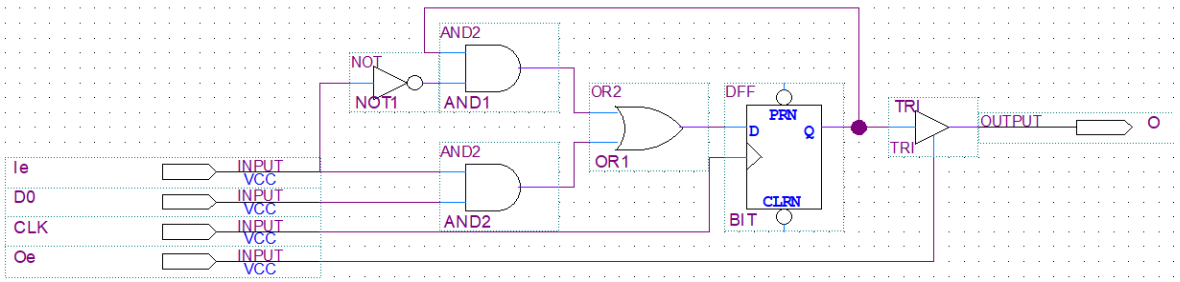
El bloque CeldaMemoria es una celda de memoria básica de 1 bit, diseñada como un registro de almacenamiento controlado. Sus entradas y salidas son las mismas que el bloque Registro12, pero en vez de poseer un bus de 12 bits para el valor de entrada DATA o para el de salida O, posee solamente 1 bit.

La celda de memoria funciona a partir de un flip-flop de tipo D y de un buffer triestado.

Dado que se trata de un flip-flop tipo D, en él se almacena 1 bit de información. Esta celda de memoria consta del valor de entrada D0, que será escrito en el buffer únicamente cuando la entrada Ie esté en 1. Esto nos permite controlar cuándo la información en la memoria es escrita y cuando es preservada.

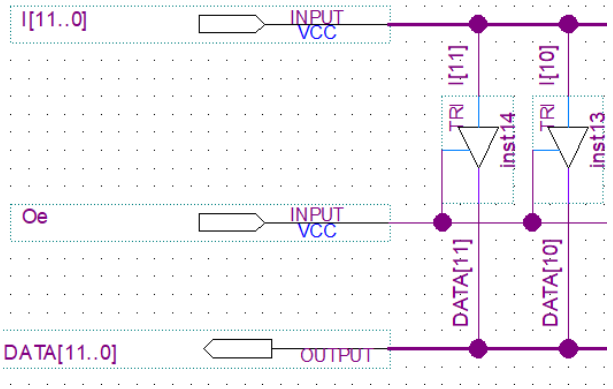
Este también consta de un bit de salida O, que tomará el valor actual del buffer únicamente cuando Oe esté en 1, de lo contrario la salida será alta impedancia (HI Z), lo que “desconectará” al buffer del bus. Esto es así porque varios elementos compartirán el mismo bus, por lo que nos interesa controlar rigurosamente qué información circula por este.

Todo esto, además es controlado por el clock, donde la escritura del dato del flip-flop se realiza en un flanco de subida del reloj

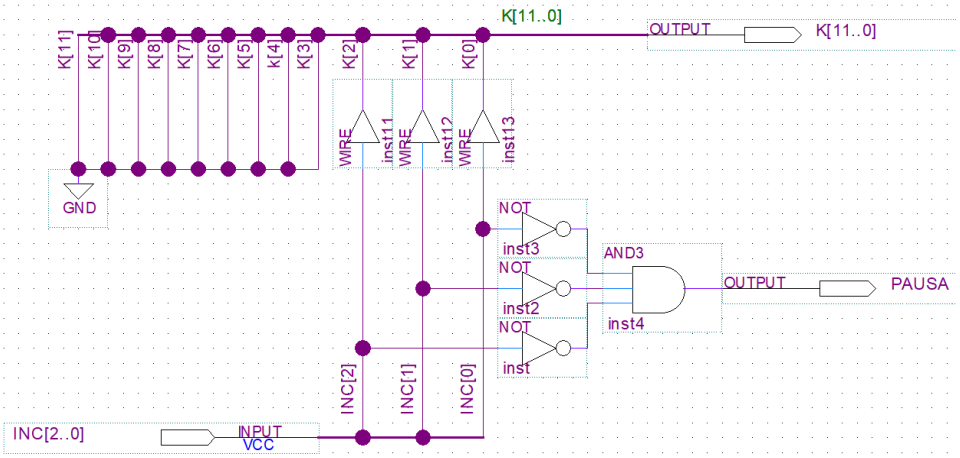


El bloque TRI12 se basa en un buffer triestado de 12 bits, utilizado para controlar el acceso de un bus de entrada de 12 bits a un bus de salida compartido. Está compuesto por 12 instancias del componente TRI, cada una manejando un bit del bus.

Fue necesario crear este componente para poder hacer un manejo eficiente del bus bidireccional DATA, que por momentos debería funcionar como input y por instantes como output, estados que deberán controlar qué elementos del circuito leen y escriben en el bus en determinados instantes.



El bloque “K_Pausa” se utiliza para identificar los momentos donde la suma o resta no debería hacerse ya que entraría en bucle. En este caso, el módulo recibe un bus de entrada de 3 bits “INC[2..0]”, donde si todos sus valores están en 0, la señal de salida “PAUSA” se pone en 1. Además, este bloque también se encarga de formar el operando K del circuito Sumador ya mencionado, que lo compone con los 9 bits más significativos en 0 y con los 3 menos significativos a partir de los valores de la entrada INC. Esto se debe a que el Sumador opera con números representados en 12 bits.



Funcionamiento del Circuito

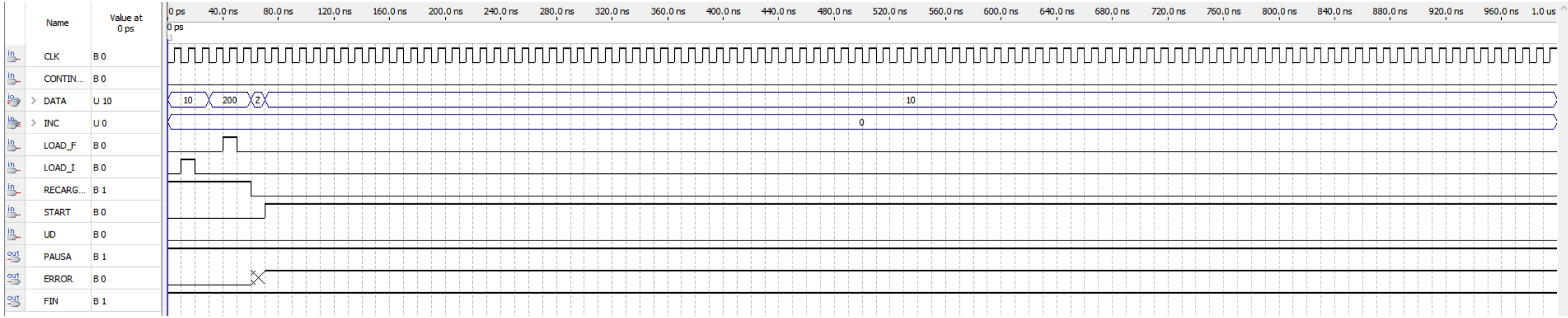
Inicialmente, se enciende la entrada **RECARGAR**, que habilitará los tristate que permiten la carga de los registros inicial y final, y deshabilitará la salida del valor actual del contador. Estando esta entrada en alto, los valores en **DATA** se cargarán en los registros inicial y final si sus entradas respectivas también están en alto. Esto se hace con un AND entre **RECARGAR** y **LOAD_X**, que activa el **le** respectivo.

Una vez completo el proceso de carga, estando la entrada **RECARGAR** en bajo el bus **DATA** será usado como un bus de salida, que reflejará el valor actual del contador, contenido en el **REG_A**. Se activa la entrada **START** e inicia el proceso de conteo, donde en cada ciclo de reloj en el bloque **SUMADOR** se efectúa la suma **REG_A + K**, cuyo resultado se almacena en **REG_A**. Este valor se compara con **REG_FINAL** en el **COMPARADOR**, y dependiendo de qué salida se active (**Ma** o **Me** o ninguna en el caso de iguales) el **ANALIZADOR DE CORTE** mediante las salidas **FIN** y **ERROR** determinará si es necesario continuar el conteo o si hay que interrumpirlo. Cabe aclarar que previamente se comparó **REG_INICIAL** con **REG_FINAL** para constatar de que los valores ingresados tienen sentido respecto del modo de conteo ingresado (**U/D**).

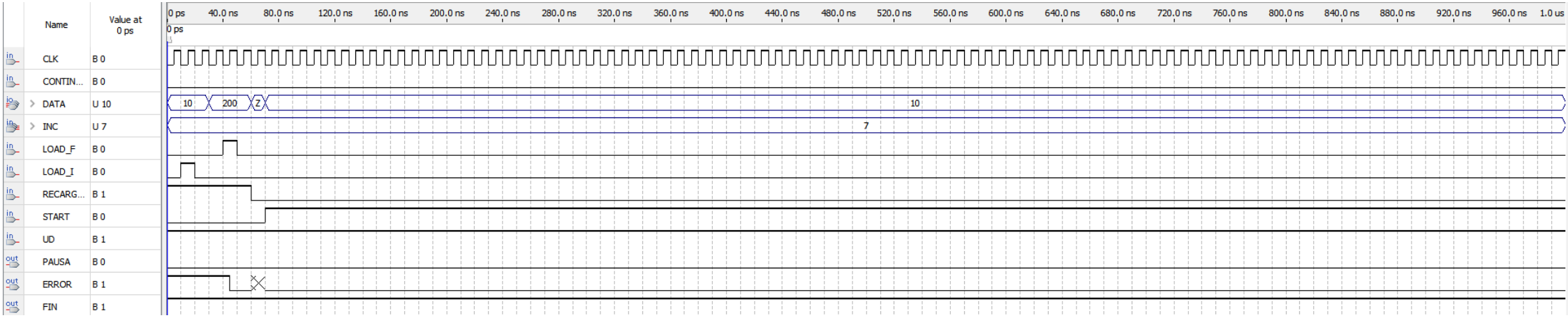
Continuar o no con el conteo consta de activar o desactivar un buffer tristate que controla si el resultado del **SUMADOR** es almacenado o no en **REG_A**, de esta manera si se desea pausar el conteo solo basta con no guardar el resultado en **REG_A**.

A continuación se muestran imágenes del funcionamiento en distintos casos (incitamos al lector el uso del zoom):

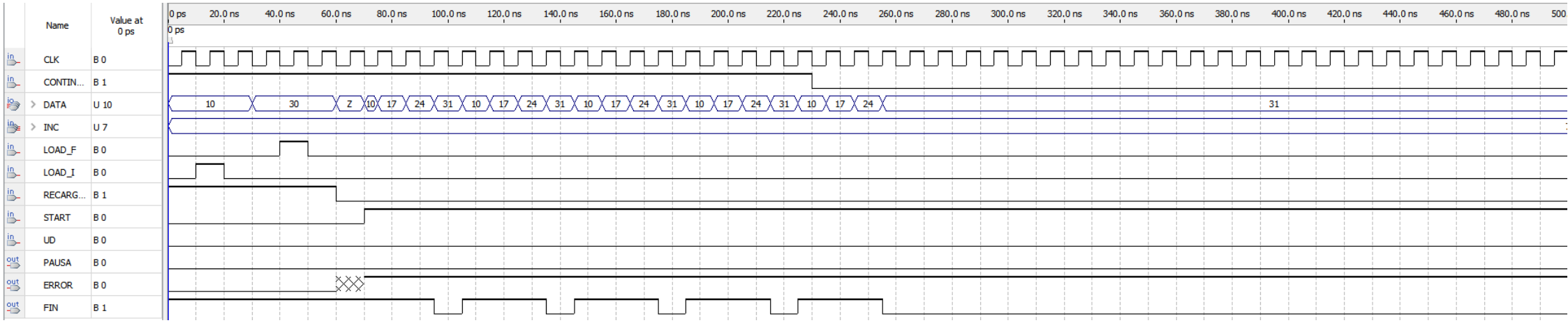
- Estado de **PAUSA**:



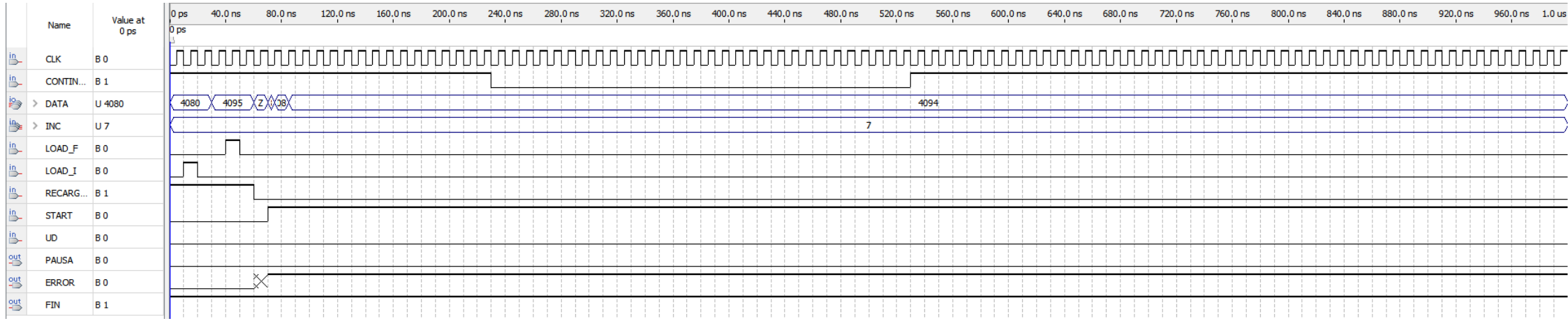
- Estado de **ERROR**:



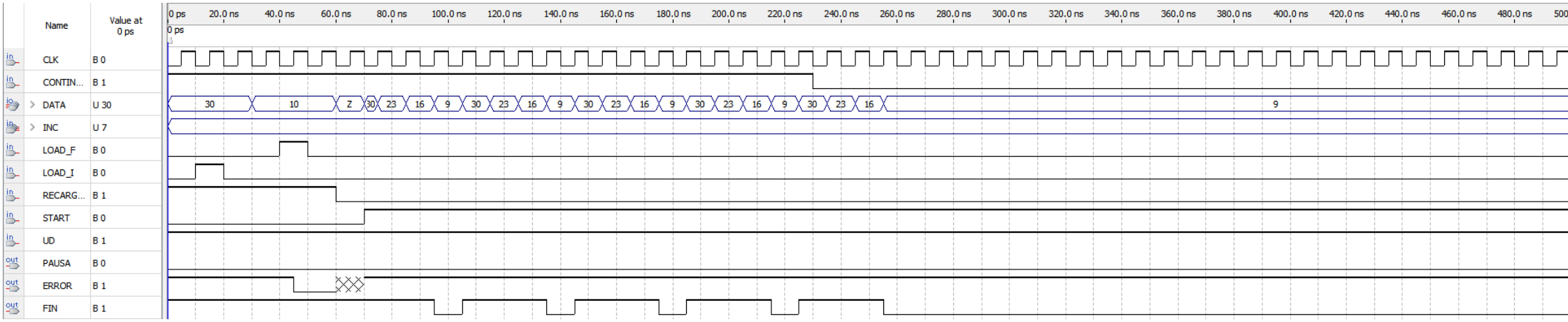
- Suma (prestar atención a la entrada **CONTINUAR**):



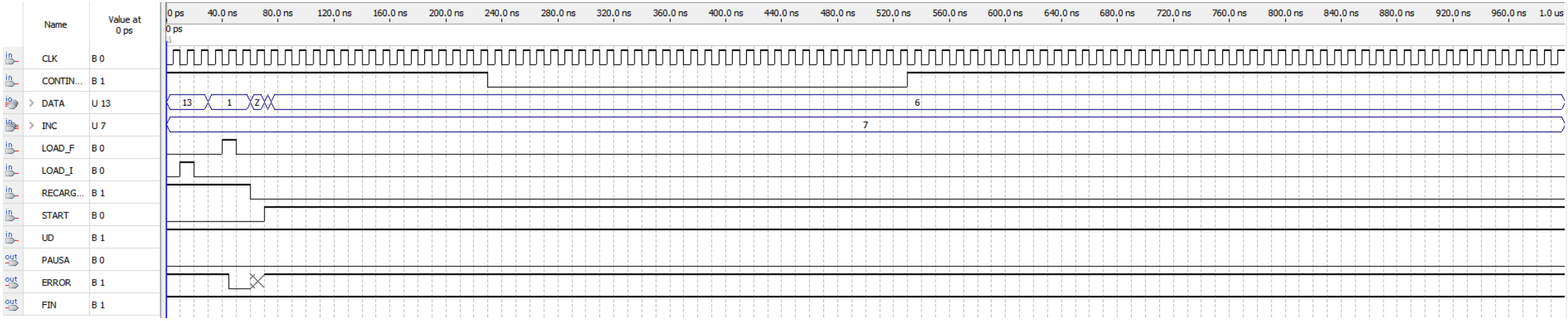
- Suma con **OVERFLOW** (el conteo se detiene antes del desbordamiento):



- Resta (prestar atención a la entrada **CONTINUAR**):



- Resta con **OVERFLOW** (el conteo se detiene antes del desbordamiento):



CIRCUITO FINAL