# TRABAJO DE LABORATORIO I: CALCULADORA "XORONUR"

Electrónica Digital - Laboratorio III

N. A. Pizarro

## 1 DESCRIPCIÓN DEL FUNCIONAMIENTO

Se implementó una calculadora, *xoronur*, en el software `VIVADO` a partir del lenguaje `VHDL`. La misma es capaz de sumar, restar, multiplicar e indicar si dos números son iguales. De entrada se tiene los números, `A` y `B`, de 4 bits con signo cada una, expresados en la codificación "signo y módulo". `OPCION` es otra entrada de 2 bits que permite elegir entre las cuatro operaciones matemáticas a realizar entre `A` y `B`. El resultado de la operación se refleja en `SALIDA`, un bus de 5 bits con signo, también expresada en la misma codificación que la entrada. Si se produce *overflow* en alguna operación se ve reflejado en el bit `ERROR`.

Internamente, xoronur dispone de 4 bloques para realizar las operaciones: `equal_gen_1`, `sum_gen_1`, `res_gen_1` y `mul_gen_1`. Todos estos cuentan con dos entradas de n bits, una salida de n+1 bits y un bit de habilitador denominado `enable`. Aunque se hayan desarrollado de forma genérica todos estos módulos, se han implementado con $n = 4$. El bloque `mul_gen_1` cuenta además con un bit de salida `error` que indicará cuando el resultado de la operación de *overflow* y no alcancen los 5 bits de salida para representar el resultado.

Mediante un multiplexor controlado por `OPCION` controlaremos los *enables* de los 4 bloques de operaciones. Dado que tenemos 4 buses de 5 bits (5 bits por operación) y solo un solo bus de 5 bits de `SALIDA`, se multiplexa esta salida con nuevamente la ayuda de `OPCION`.

Los cuatro bloques de las operaciones trabajan con la codificación "complemento a 2" por lo que xoronur cuenta con dos bloques extra: `a_compl2_A` y `a_compl2_B` para convertir a las entradas `A` y `B` de codificación "signo y módulo" a "complemento a 2" y un bloque más: `a_compl2_salida` para convertir el resultado de la operación de "complemento a 2" a la

codificación "signo y módulo".

**¿Qué es y para qué está el `enable`?** El `enable` al tener un $'1'$ habilita el funcionamiento del bloque, mientras que con un $'0'$ lo deshabilita. Suele ser de buena práctica agregar este bit en los componentes para su reutilización. En este caso no era estrictamente necesario que esté el `enable` dado que las salidas igualmente estaban multiplexadas. Podría haber puesto todos los *enables* a '1' y la calculadora andaría bien, o directamente no haberle puesto los *enables*, (pero tengo algunos tocs).

*Disclaimer*: Esto del *enable* tendría más sentido o uso si en vez de un multiplexor para las salidas hubiera puesto todos los bloques en paralelo(i.e. cortocircuitar bit a bit las salidas de los cuatro bloques). Para hacer esto, cuando no esté habilitado el bloque debería poner las salidas del mismo a alta impedancia 'Z', con esto me ahorraría un multiplexor y le daría uso al *enable*. Pero a `VIVADO` no le gusta simular con 'Z', así que cuando el `enable` esté a '0' las salidas no estén a alta impedancia sino estén a '0'. No me siento moralmente bien poniendo en paralelo los bloques si no están en alta impedancia, terminé usando un multiplexor y bueno, los *enables* los dejé porque ya los había hecho. :-)

**¿Porqué sólo la multiplicación presentará *overflow*?** La suma y la resta de dos números de n bits nos dará un número binario representable en n+1 bits, recordando que $n = 4$, no se tendrá *overflow* en `sum_gen_1` ni `res_gen_1`. Pero la multiplicación de dos números de 4 bits nos dará un número binario representable en $2 * 4 + 1$ bits, por lo que para ciertas entradas se producirá *overflow* dado que solo disponemos de 5 bits de salida, reflejándose con un $'1'$ en la salida `error`.

**¿`a_compl2_A` es igual que `a_compl2_salida`?** Si. Si uno se pone a hacer las cuentas resulta que el algoritmo para pasar de la codificación "signo y módulo" a "complemento a 2" es la misma que de "complemento a 2" a "signo y módulo". Había que tener un poco de cuidado porque en una de las conversiones usamos números de 4 bits mientras que en otro usamos 5 bits. Esto se solucionaba haciéndolo genérico.

**¿Podría haber usado el bloque `sum_gen_1` para hacer la operación de la resta?** Si. Podría haber hecho la suma de `A` con el complemento a 2 de `B`, pero me requería más recurso mental que copiar y pegar y cambiar un signo. Aplica análogamente con la multiplicación.

**¿Porqué la calculadora da de resultado '-1' si son iguales?** Bueno, no había mencionado antes a este detalle de la operación igualdad así que lo hago ahora. Si `A` y `B` son iguales tengo de `SALIDA` un '-1' y si son diferentes tengo '0'. No hay motivo alguno de porqué elegí el '-1'.

**¿Porqué el bloque de multiplicación tiene una salida de $n + 1$ bits (5 bits con $n = 5$)?** ¡Buena pregunta!. El multiplicador no es tan genérico como los demás bloques, podría haber hecho este bloque con una salida de $2 * n + 1$ bits y fuera del bloque igualar una señal auxiliar a los $n + 1$ bits menos significativos de la salida del bloque de multiplicación(porque serían los únicos bits que se podrán ver en la salida). Tal vez en la próxima versión venga corregido.

**¿Porqué xoronur?** Es un juego de palabras entre el nombre de mi gata y la compuerta xor.

## 2 Diagrama en bloque

El diagrama en bloque de la calculadora se puede observar en la figura 2.1.

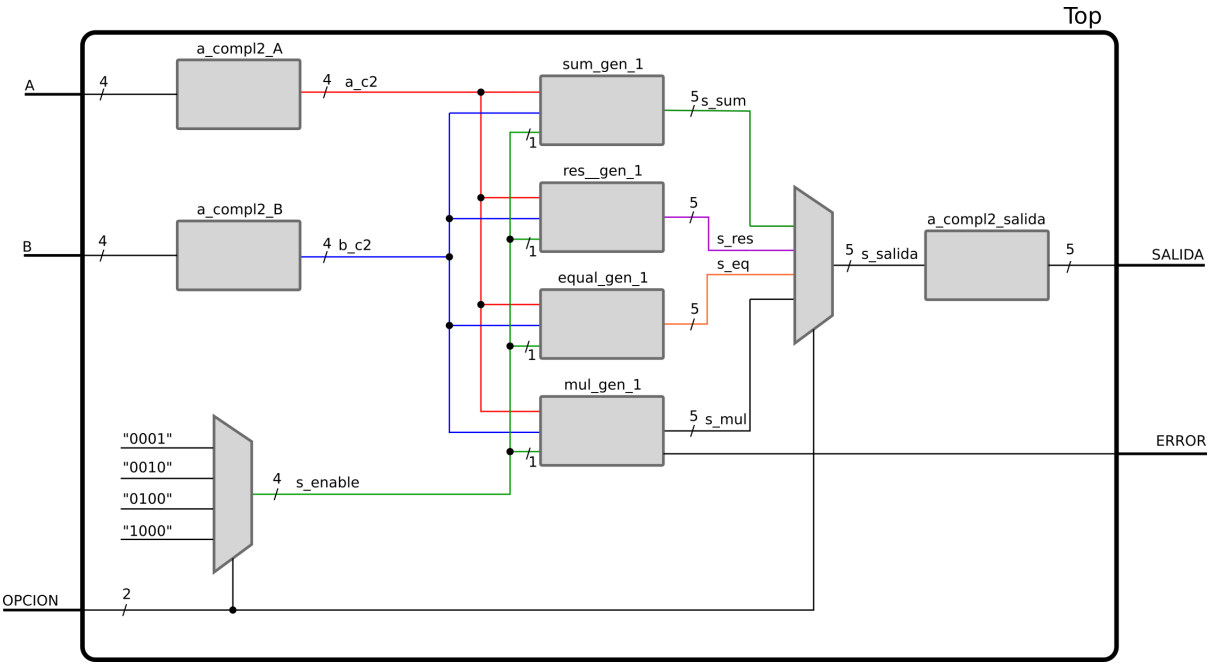


Figure 2.1: Esquema en bloques de la calculadora implementada.

## 3 Código

Todos los códigos se encuentran disponibles y descargables en el repositorio: https://github.com/AgustinaP/xoronur[1].

## 4 Verificación comportamental

Para la corroboración del correcto funcionamiento de la calculadora se realizaron los siguientes test:

[1] Por cualquier inconveniente mandar un mail a nadia.pizarro@ib.edu.ar

1. $5+2=7$
2. $4-5=-1$
3. $(-2)-3=-5$
4. $32=6$
5. $(-4)*(-3)=12$
6. $7*4 \rightarrow Error$
7. $1=7 \rightarrow 0$
8. $6=6 \rightarrow -1$

Dichos tests se simularon en el software VIVADO y en la figura 4.1 se puede observar los resultados. Para los bits de OPCION se usó el '00' para la suma, '01' para la resta, '10' para la multiplicación y el bit '11' para la igualdad.

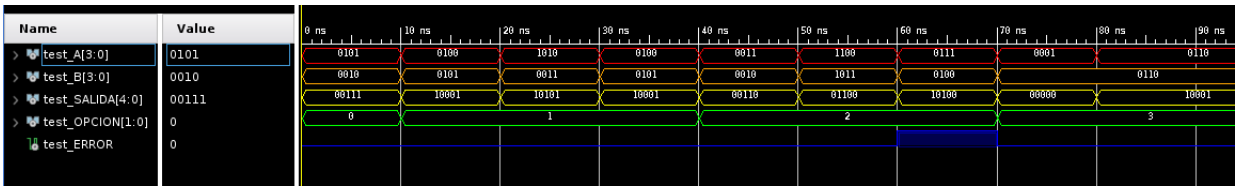


Figure 4.1: Salida del simulador VIVADO para los test propuestos.