Cómo implementar memorias en Vivado. Parte 2: memoria ROM

Creador: David Rubio G.

Entrada: https://soceame.wordpress.com/2024/11/05/como-implementar-memorias-en-vivado-parte-2-memoria-rom/

Blog: https://soceame.wordpress.com/

GitHub: https://github.com/DRubioG

Fecha última modificación: 23/02/2025

Esta es la segunda parte de cómo implementar memorias en Vivado. Primera parte:

https://soceame.wordpress.com/2024/11/03/como-implementar-memorias-en-vivado-parte-1-memoria-ram/

En esta segunda parte hablaremos de memorias de tipo ROM.

Una memoria de tipo ROM es una memoria de solo lectura, por lo que la información tiene que estar precargada dentro de la memoria antes de ser usada. Para ello se tiene que importar dentro de los bloques un fichero con los datos, también llamados coeficientes (COE).

Para ello Vivado da dos opciones para crear memorias, que son las mismas que para la ROM, pero con menos opciones de configuración, el *Block Memory Generator* y el *Distributed Memory Generator.*

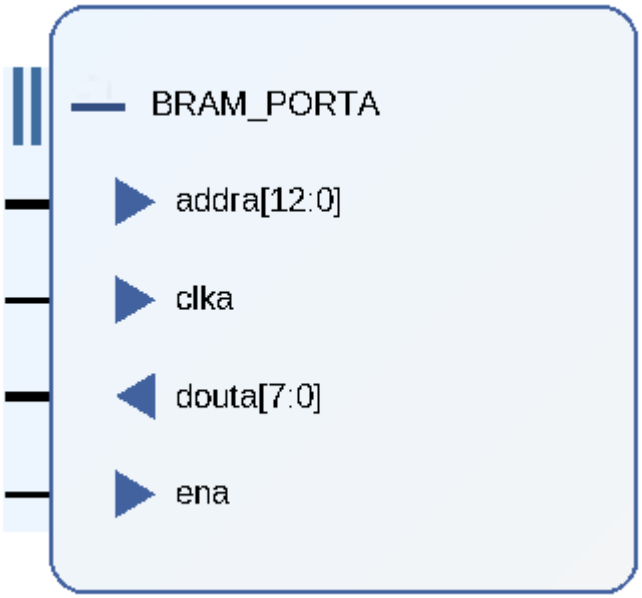
Block Memory Generator

El *Block Memory Generator* es un bloque IP de Xilinx que se vale de los recursos internos de BRAM para crear la memoria.

Para crear memorias tiene dos opciones, la *Single Port ROM* y la *Dual Port ROM*.

Single Port ROM

La *Single Port ROM* es un memoria de un solo puerto, que solo necesita de un reloj y una dirección de memoria para funcionar.

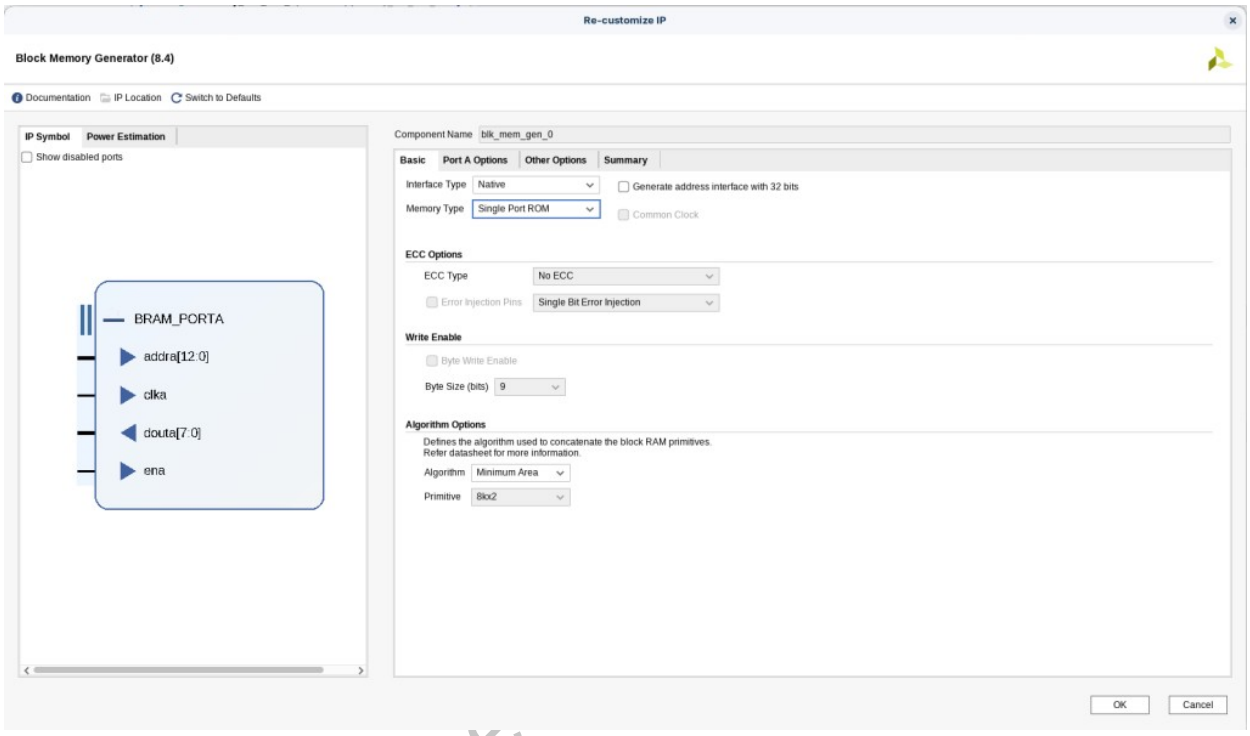


https://soceame.wordpress.com/2024/11/05/como-implementar-memorias-en-vivado-parte-2-memoria-rom/

Configuración de Single Port ROM

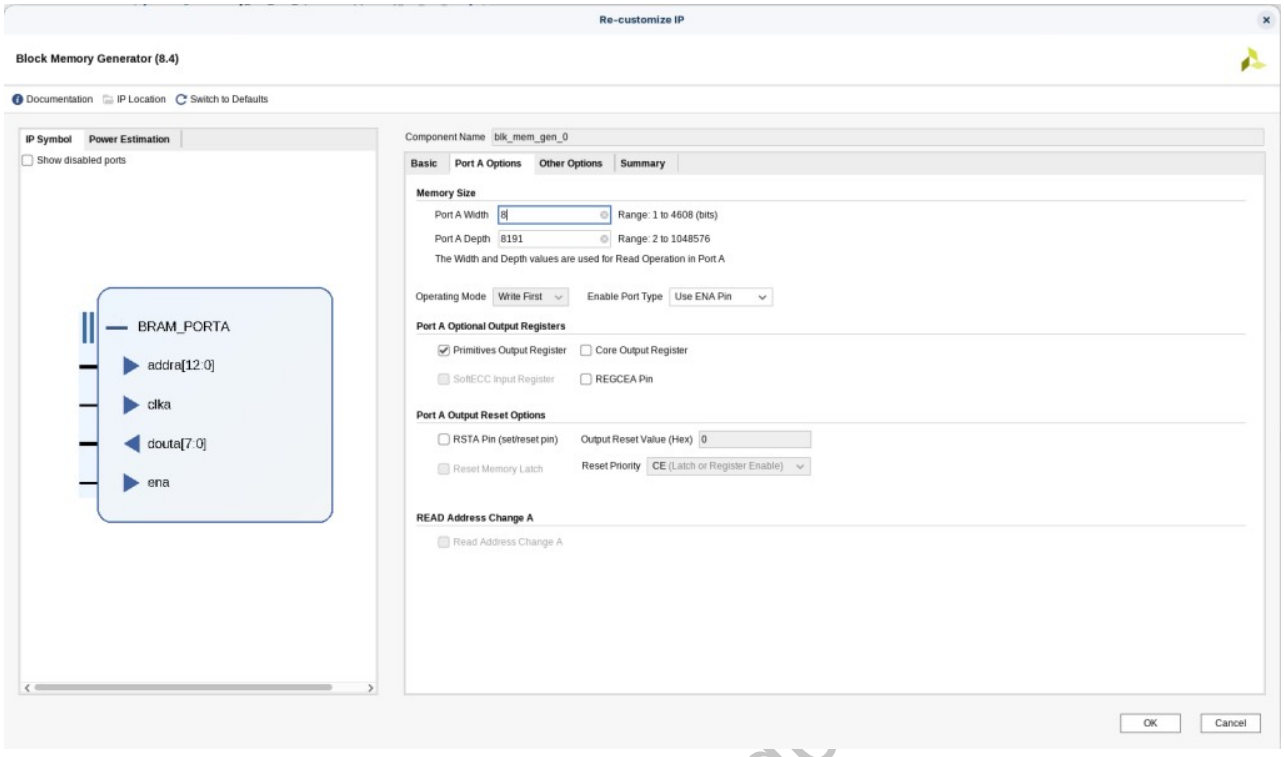
Para la configuración de una *Single Port ROM* se tiene una primera pantalla en el *Block Memory Generator.*

En esta pantalla se pueden configurar si quiere utilizar una interfaz AXI, que las direcciones sean fijas a 32 bits, dividiendo la memoria en bloques de 4 bytes, o el sistema de creación de la memoria, si se quiere que utilice un área mínima o rebaje el consumo.

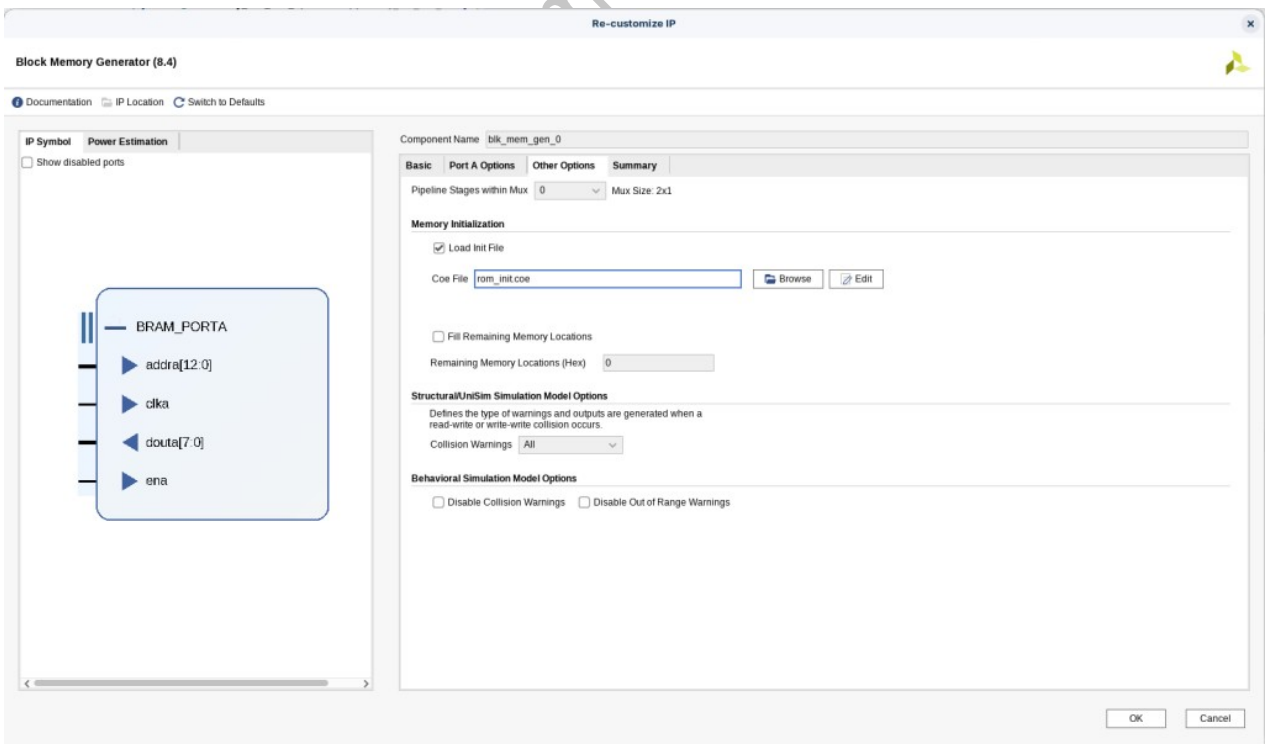


En la siguiente pestaña se puede elegir el tamaño de los datos en la memoria o la profundidad de la memoria. También, se puede configurar para que esté siempre habilitada.

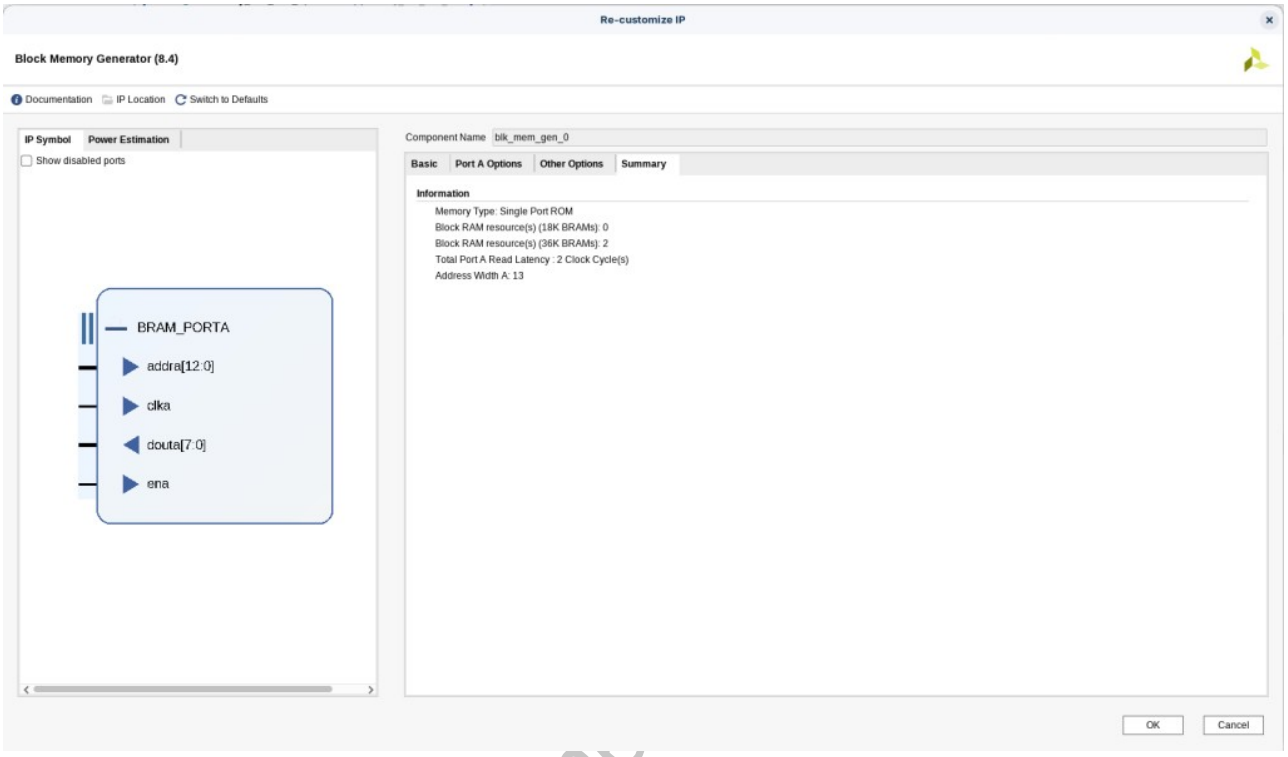
https://soceame.wordpress.com/2024/11/05/como-implementar-memorias-en-vivado-parte-2-memoria-rom/



Luego en la pestaña *Other Options*, se tiene que habilitar la opción *Load Init File*, que es la opción que permite cargar un fichero de tipo COE *(más adelante se explica como se genera un fichero de este tipo)*, que es el fichero con los datos que va a tener la memoria ROM. Tmbién se puede configurar para que aquellos datos que no se van a utilizar se rellenen con datos.

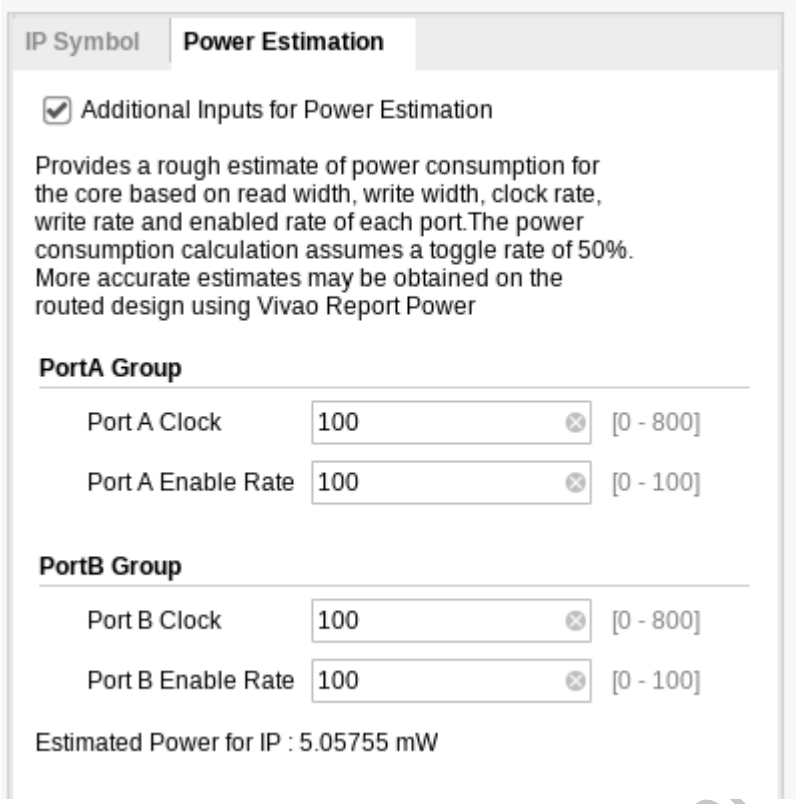


La última pestaña ofrece un resumen de la información para implementar esta memoria. Aquí aparece un dato importante que es el número de ciclos de reloj que se necesita para completar una lectura, que en este caso es de 2 debido a que el *Block Memory Generator* tarda 2 ciclos en completar cualquier lectura.



También, hay una opción para estimar el consumo de potencia de la memoria.

https://soceame.wordpress.com/2024/11/05/como-implementar-memorias-en-vivado-parte-2-memoria-rom/



Cómo crear un fichero COE

Un fichero COE es un tipo de archivo que contiene información para grabarla en una ROM de Xilinx.

Este tipo de ficheros se basa en dos filas, una que indica en que formato están guardados los datos, **MEMORY_INITIALIZATION_RADIX**, y otra fila que contiene exactamente estos datos en formato matriz, **MEMORY_INITIALIZATION_VECTOR**.

NOTA: es importante aclarar cosas aquí.

El **MEMORY_INITIALIZATION_RADIX** nos dice en que formato están representados los datos (16: hexadecimal, 8: octal, 2: binario, etc), lo que **NO nos dice es el tamaño de los datos que se van a utilizar** (si son de 16 bits, 13 bits, 10 bits, etc), esto último **NO** se define en ningún sitio. Entonces, los coeficientes se ajustan al tamaño de la salida.

Para que lo entendamos mejor. Yo puedo tener un **MEMORY_INITIALIZATION_RADIX** de 16 (formato hexadecimal) y tener un tamaño de datos de 10bits. Entonces, lo que tiene que ocurrir es que ningún coeficiente en formato hexadecimal puede exceder los 10 bits (entonces, los datos solo pueden ir del 00 al 3FF). Si no se produce un error al cargar los datos.

https://soceame.wordpress.com/2024/11/05/como-implementar-memorias-en-vivado-parte-2-memoria-rom/

Esto es un pequeño ejemplo de fichero:

```
MEMORY_INITIALIZATION_RADIX=16;
MEMORY_INITIALIZATION_VECTOR=
11,
22,
33,
44,
55,
66,
77,
88,
99,
aa,
bb,
cc,
dd,
ee,
ff,
00,
a1,
a2,
a3,
a4,
a5,
a6,
a7,
a8,
b1,
b2,
b3,
b4,
b5,
b6,
b7,
b8;
```


En la primera parte se puede ver que se le dice que los datos están en representación hexadecimal, y en la segunda parte se guardan los datos.

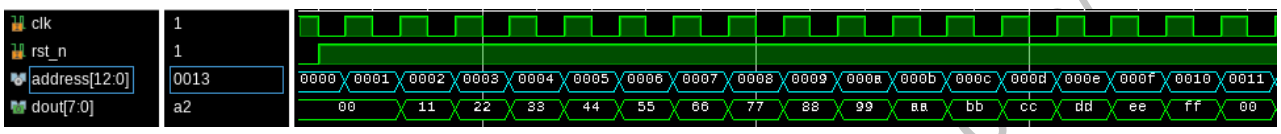
Entonces, para que este ejemplo pueda ser utilizado como coeficientes en una ROM, **la salida tiene que ser como mínimo de 8 bits** *(porque uno de los coeficientes es ‘ff’)*

NOTA: es recomendable que la profundidad de la memoria se igual que la de los coeficientes para evitar que sobre memoria (que estará llena de ceros)

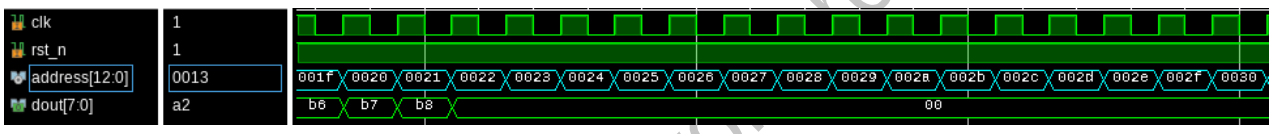
Simulación de Single Port ROM

Ahora lo que se va a hacer es hacer una simulación del funcionamiento de la ROM. Para ello lo único que se va a hacer es leer los datos grabados en la ROM. Para la ROM se va a utilizar el fichero de coeficientes anterior.

Entonces, si se le pasa un reloj con una dirección a la memoria, la ROM tarda 1 ciclo de reloj en devolver el dato.



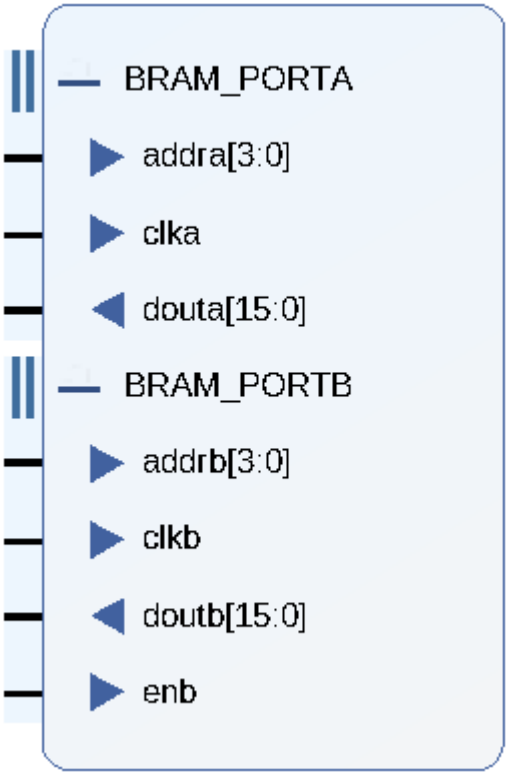
Y cuando las direcciones sobrepasan el valor de los coeficientes guardados, el valor devuelto es cero.



Dual Port ROM

La otra opción de configuración del *Block Memory Generator* es la de crear una memoria de lectura de doble puerto.

Esta opción permite crear una memoria con dos dirección de memoria distintas y dos salidas también distintas.

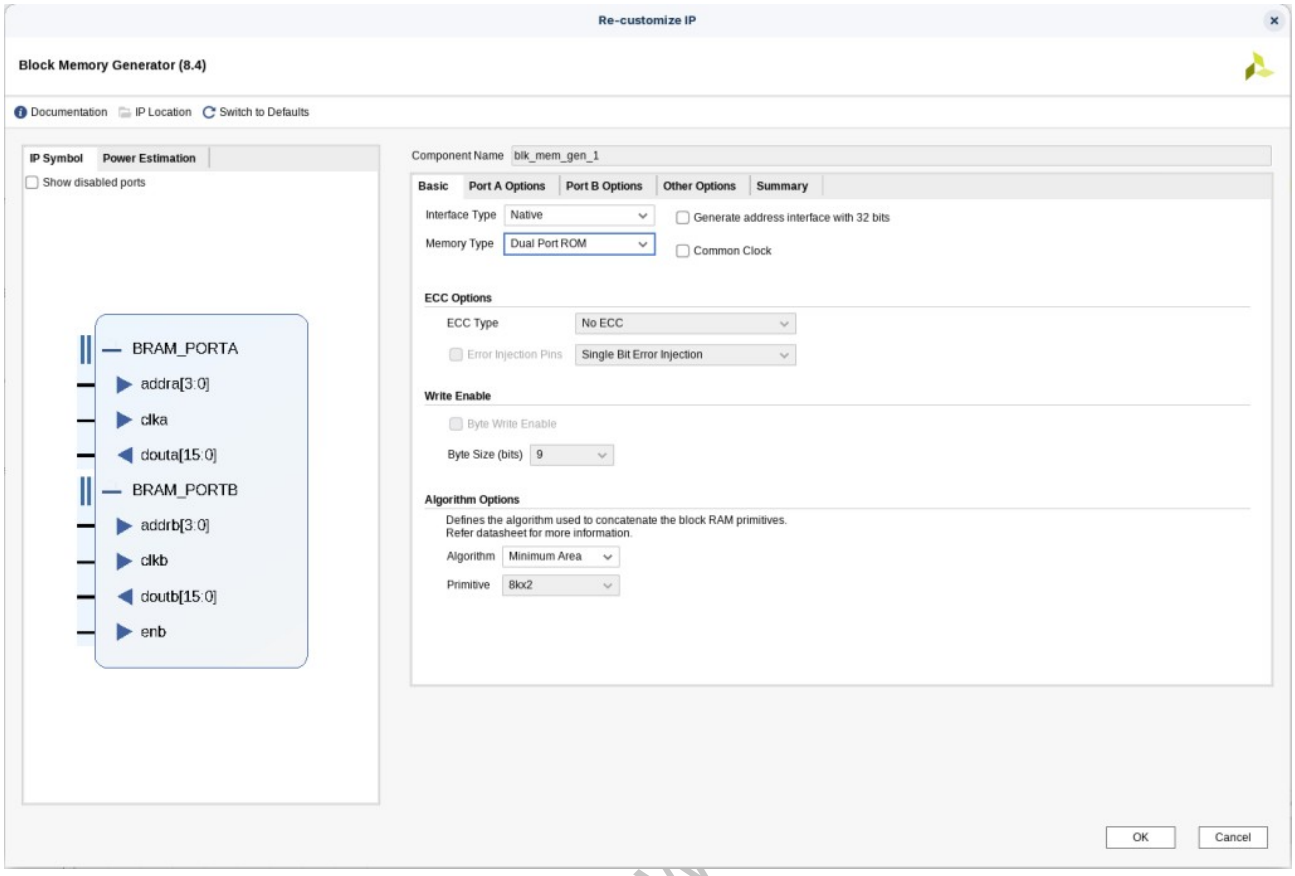


Configuración de Dual Port ROM

Para la configuración del *Block Memory Generator* como *Dual Port ROM*, se hace igual que para configurar una *Single Port ROM*.

Para ello en la primera pestaña se marca la opción de Dual Port ROM. Si se marca la casilla *Common Clock*, todas las operaciones se realizan con el reloj del puerto A.

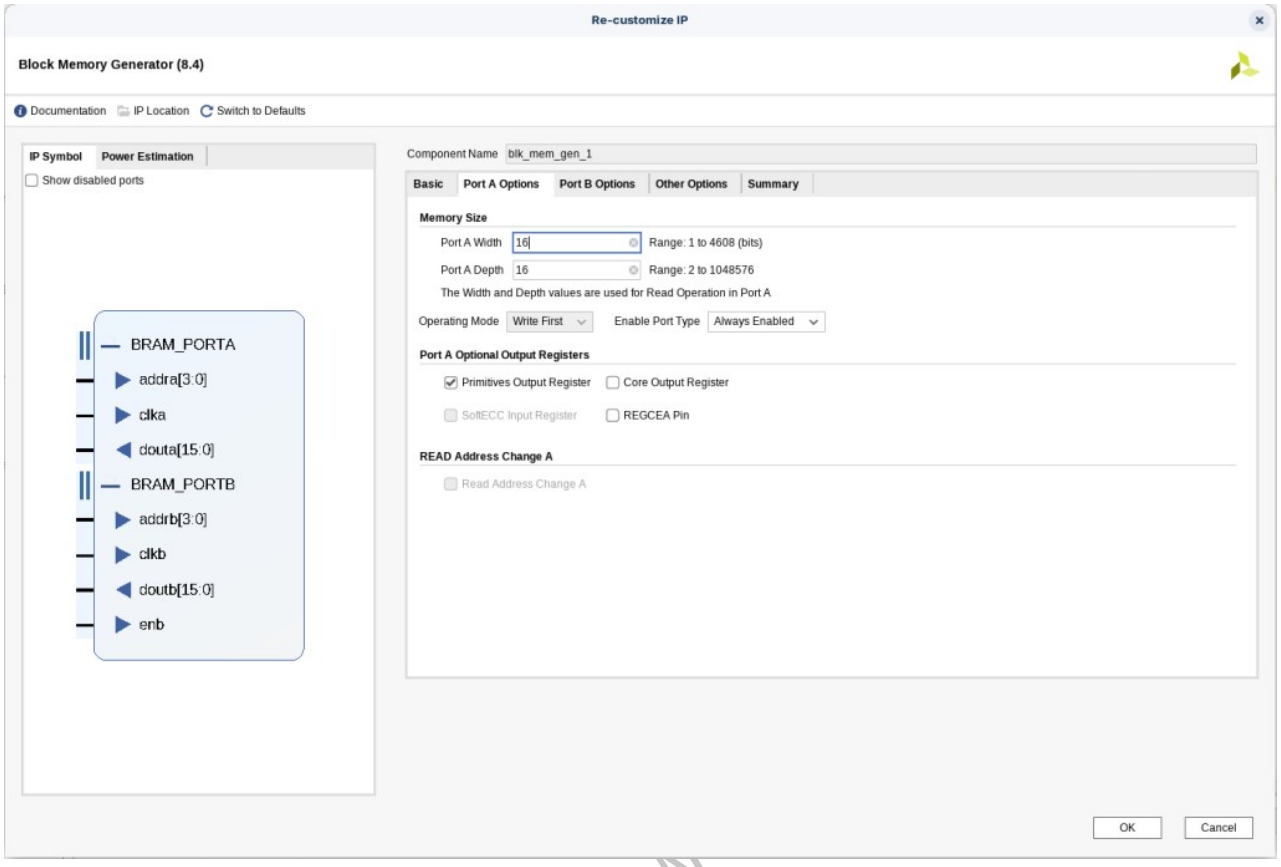
https://soceame.wordpress.com/2024/11/05/como-implementar-memorias-en-vivado-parte-2-memoria-rom/



Ahora hay dos pestañas, una para configurar el puerto A y otra para configurar el puerto B.

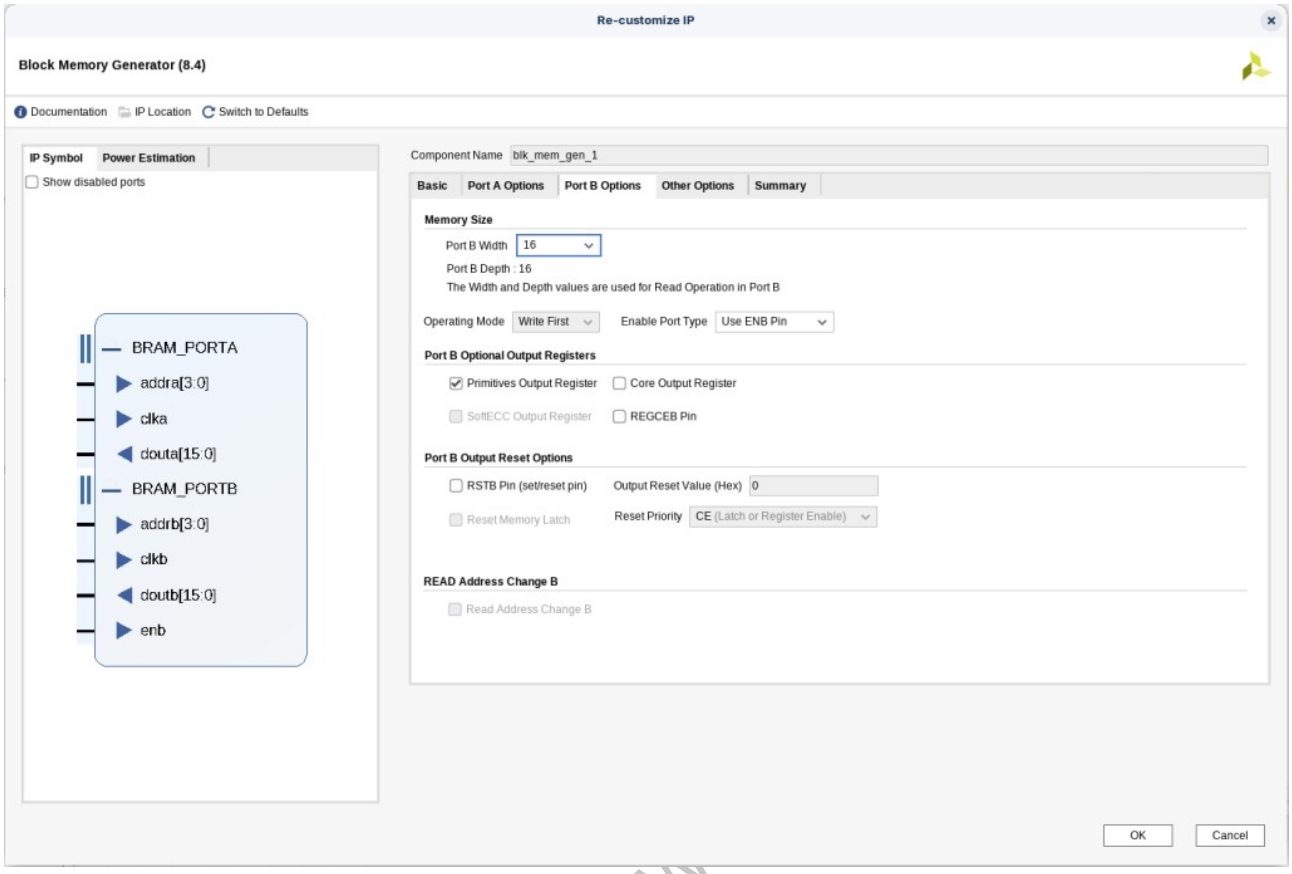
La configuración del puerto A, es igual que la del *Single Port ROM*, con las misma opciones.

https://soceame.wordpress.com/2024/11/05/como-implementar-memorias-en-vivado-parte-2-memoria-rom/



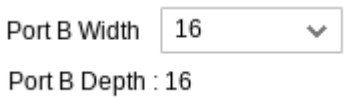
Y la configuración del puerto B esta condicionada en el tamaño por la profundidad de la memoria configurada en el puerto A, pero permite seleccionar el tamaño de los datos de salida y si el puerto de habilitación está siempre activo.

https://soceame.wordpress.com/2024/11/05/como-implementar-memorias-en-vivado-parte-2-memoria-rom/



NOTA: si el tamaño de los datos de salida es distinto del puerto A, las direcciones de memoria se ven modificadas.

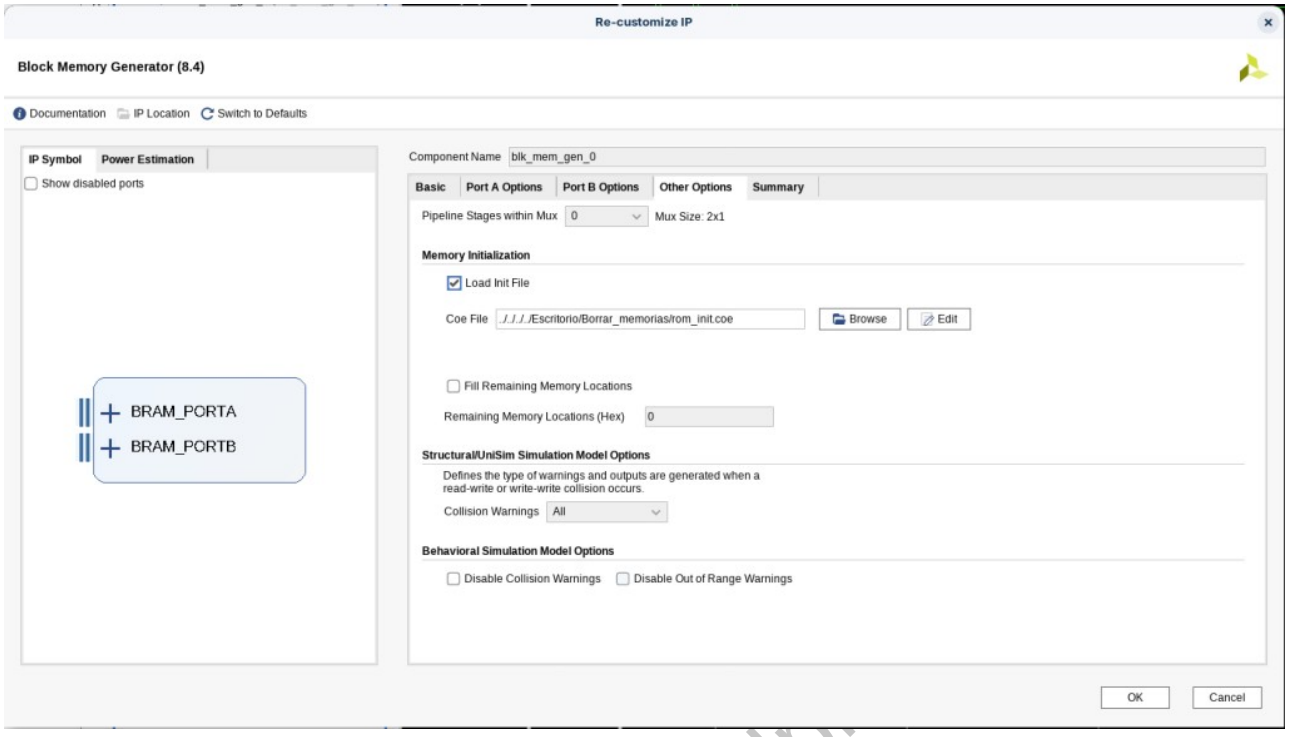
Por ejemplo, si se configura una memoria para una profundidad de 16 datos. Si los datos son de 16 bits, la profundidad se mantiene constante.



Si ahora los datos son de 32 bits, el tamaño de la memoria se reduce.

Para terminar la configuración es necesario darle un fichero COE con los datos al bloque IP.

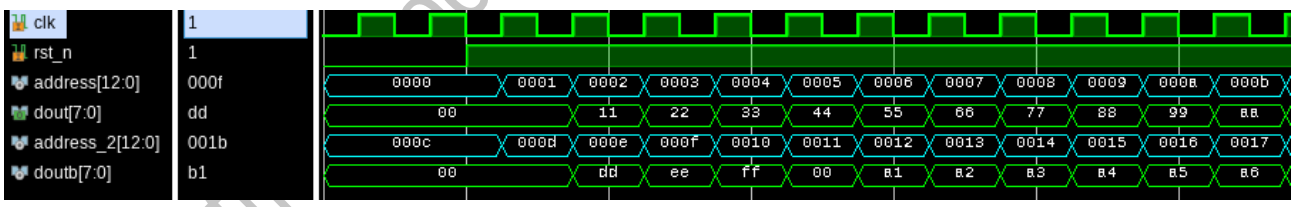
https://soceame.wordpress.com/2024/11/05/como-implementar-memorias-en-vivado-parte-2-memoria-rom/



Simulación de Dual Port ROM

Ahora lo que se va es a hacer una simulación igual que en la *Single Port ROM*. En esta simulación se va a leer por los dos puertos en el mismo instante de tiempo. El fichero utilizado es el mismo que en la simulación anterior.

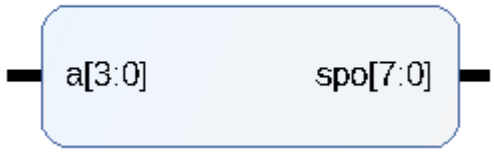
Entonces, se puede ver que al igual que en la anterior simulación, solo tarda 1 ciclo de reloj en escribir los datos guardados a la salida.



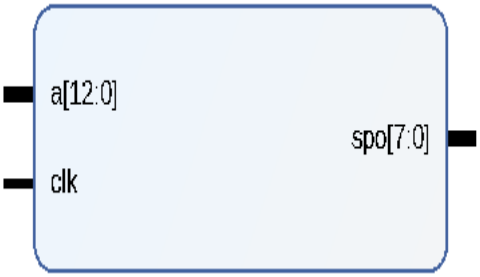
Distributed Memory Generator

Esta es la última opción que ofrece Vivado para generar una memoria ROM. Esta memoria ROM que genera la genera utilizando lo medios propios de la lógica de Vivado.

Este bloque genera una memoria que solo tiene un puerto de direcciones de entrada y un puerto de datos de salida. Por lo que es una memoria asíncrona por defecto *(en la configuración te explico cómo se puede hacer de ella una memoria síncrona)*.

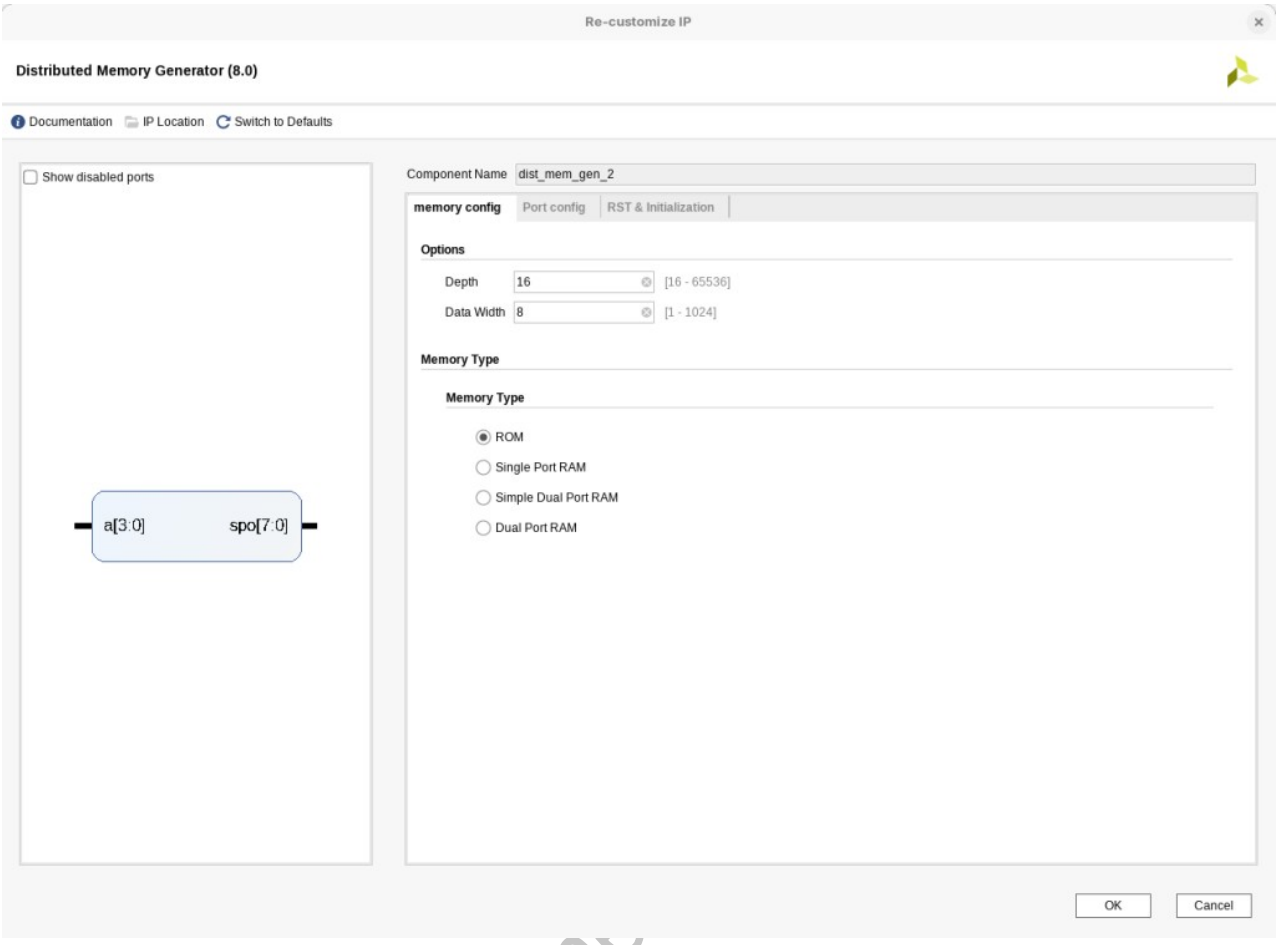


(si se quiere que sea síncrono, los puertos quedan así)



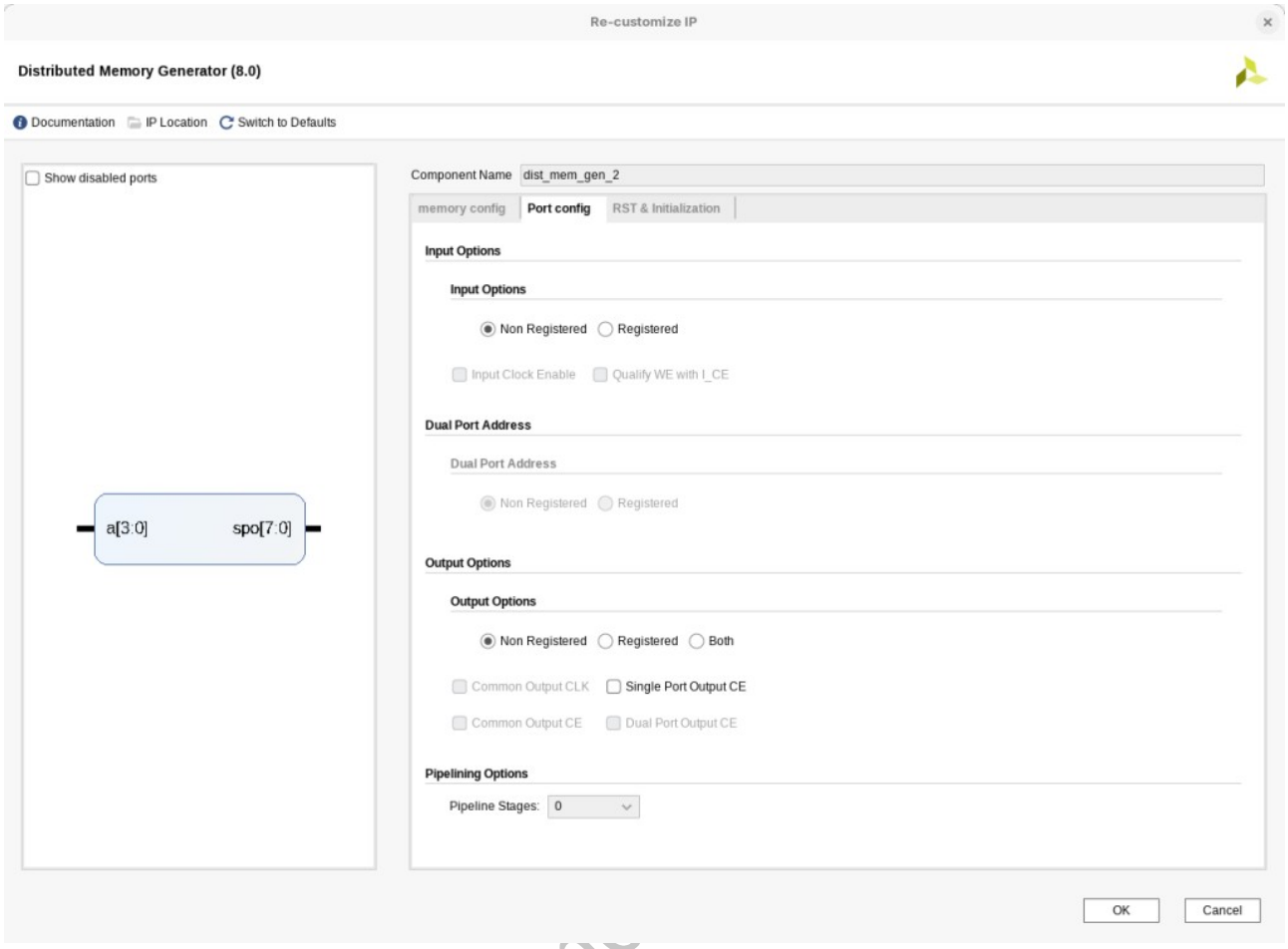
Configuración de la ROM

Para la configuración de la ROM se requiere de marcar la casilla *ROM* en el *Distributed Memory Generator.* También se tiene que elegir el tamaño de la salida y la profundidad de la memoria.



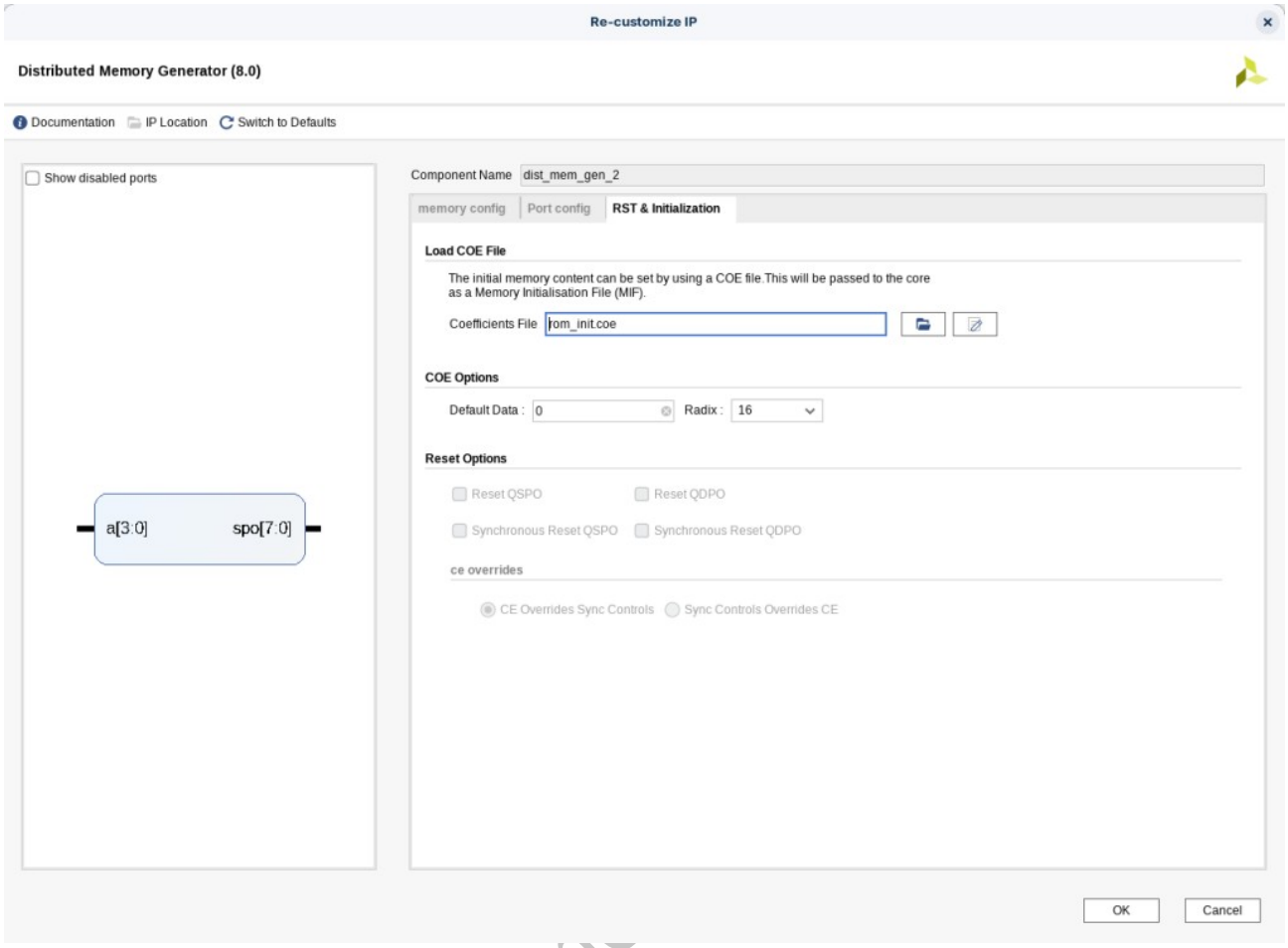
En *Port config* es dónde se elije el tipo de funcionamiento de la memoria. La memoria tiene dos tipos de funcionamiento, *asíncrono*, por defecto, en el que la escritura se hace al instante, y *síncrono* en el que se depende de un reloj, en esta pestaña es dónde se puede seleccionar si se quiere usar un reloj. Para ello se tiene que marcar la casilla *Registered* de *Input Options*.

https://soceame.wordpress.com/2024/11/05/como-implementar-memorias-en-vivado-parte-2-memoria-rom/



Y por último, en la pestaña *RST & Inicialization* se añade el fichero de datos.

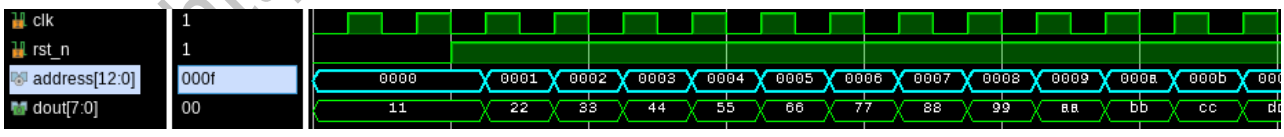
https://soceame.wordpress.com/2024/11/05/como-implementar-memorias-en-vivado-parte-2-memoria-rom/



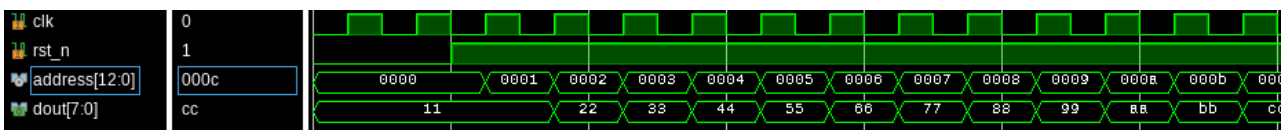
Simulación de la ROM

El último paso es simular el funcionamiento de esta memoria *ROM*, para ello lo único que se hará es pasarle una dirección de memoria al bloque para ver la salida que genera. Los coeficientes son los mismos que en las simulaciones anteriores.

Este bloque IP por defecto no requiere de un reloj para funcionar, por lo que la escritura de los datos en la salida es inmediata.



Si se quiere que funcione de forma *síncrona*, se marca la opción de *Registered*. Y entonces, se convierte en un *memoria síncrona*, con un retardo en la escritura de 1 ciclo de reloj.



Nota final

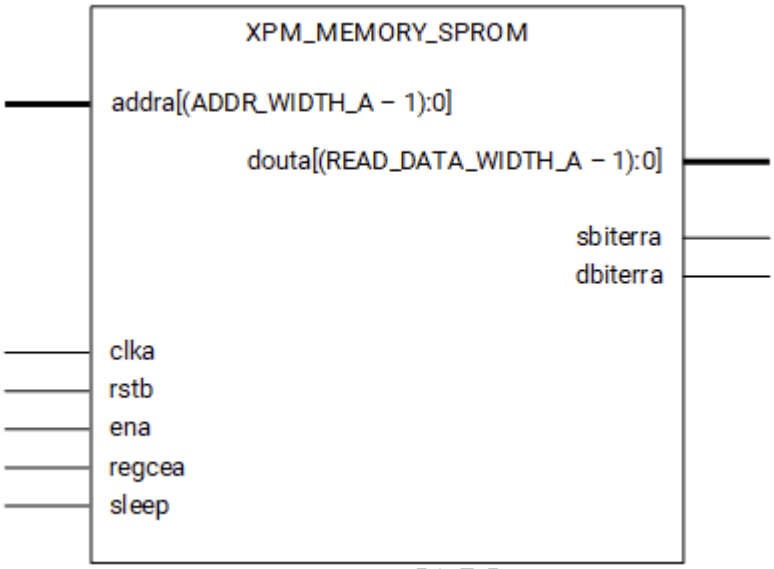
Muchas de memorias están creadas mediante macros internas de Vivado (o sea, llamadas internas a módulos que Vivado no enseña, pero que en la documentación sí que documenta).

La macros utilizadas pertenecen a la librería *XPM*. La librería de macros se instancia de la siguiente manera:

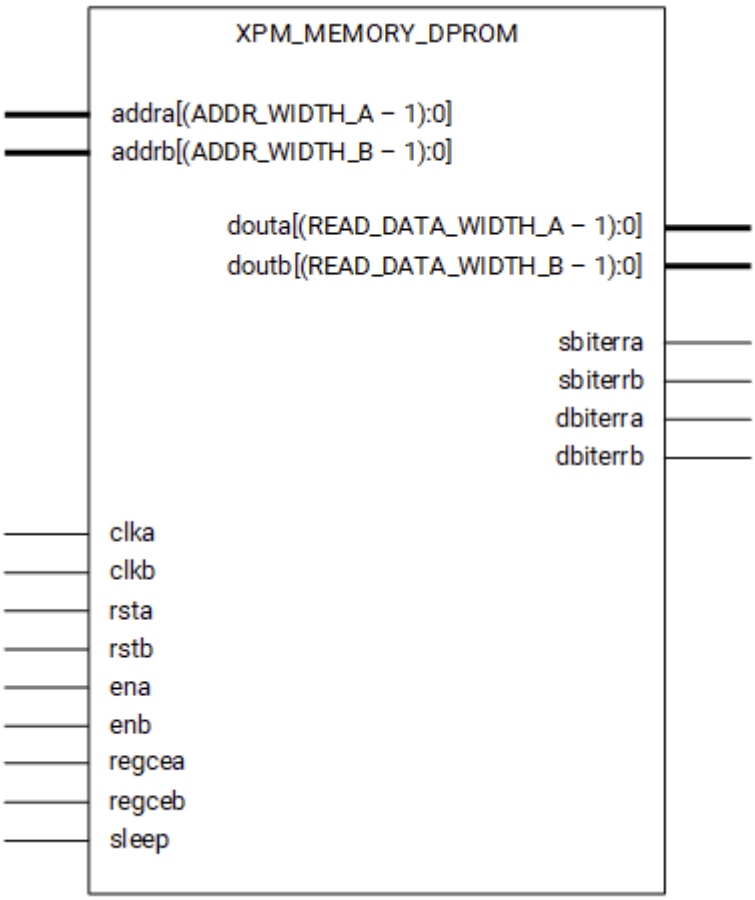
```
Library xpm;
use xpm.vcomponents.all;
```

Y son las siguientes:

- **XPM_MEMORY_SPROM**: esta macro hace referencia a las memorias *Single Port ROM*



- **XPM_MEMORY_DPROM**: esta macro hace referencia a las memorias *Dual Port ROM*



Bibliografía macros

- UG953: https://docs.amd.com/r/en-US/ug953-vivado-7series-libraries/XPM_MEMORY_SPRAM
- UG974: https://docs.amd.com/r/en-US/ug974-vivado-ultrascale-libraries/XPM_MEMORY_SPRAM