Cómo implementar memorias en Vivado. Parte 1: memoria RAM

Creador: David Rubio G.

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

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

GitHub: https://github.com/DRubioG

Fecha última modificación: 23/02/2025

Una memoria RAM, es una memoria de acceso aleatorio, es una memoria que no es capaz de almacenar los datos que contiene de forma duradera, entonces, en caso de que se corte la alimentación, la memoria se borra. No es como el caso de una memoria ROM, donde la información se guarda *(aunque en las FPGAs esto último es muy relativo, porque si se corta la alimentación en una FPGA de tipo SRAM, la ROM creada se borra a menos que se guarde externamente en una memoria ROM externa o una flash. En una FPGA de memoria Flash, la ROM se queda grabada, y la RAM también se quedaría grabada)*.

También, es importante distinguir que una memoria **RAM** se puede parecer a un **FIFO**, pero son elementos distintos. Un **FIFO** tiene un *apuntador de memoria incremental*, de forma que cada dato que se incorpora incrementa las direcciones del apuntador que poseen memoria. Mientras, que una memoria **RAM** tiene un *apuntador de memoria selectivo*, lo que significa que puede acceder a cualquier lugar de la memoria, sin necesidad de que esta pueda contener datos almacenados.

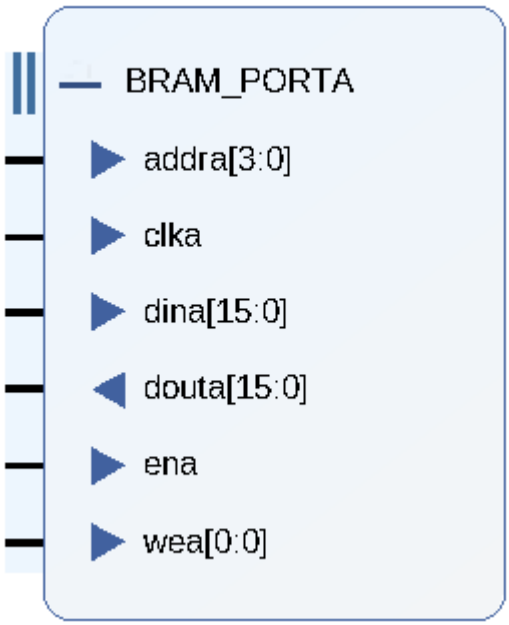
Para crear memoria RAM en una FPGA de Xilinx existen dos opciones de bloques IP, usar el *Block Memory Generator* o usar el *Distributed Memory Generator.* También existe la tercera opción que es crearte tu propia memoria RAM usando HDLs, pero en este caso voy a explicar las opciones que Xilinx leva incorporadas.

Block Memory Generator

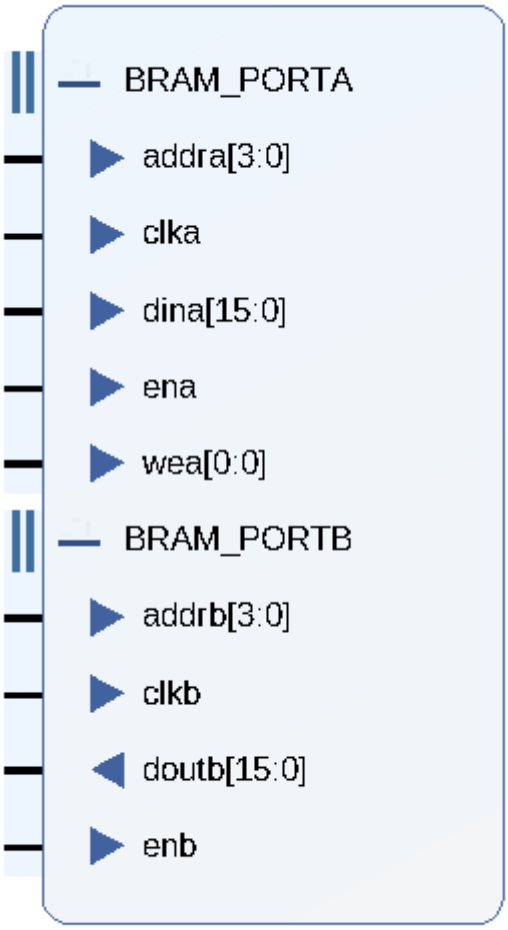
El *Block Memory Generator* es un bloque IP muy versátil que contiene diferentes opciones de configuración como memoria RAM (también tiene como memoria ROM que se comentarán en otra entrada). Este bloque IP se vale del uso de recursos internos de las FPGAs de Xilinx como son las BRAMs.

Este bloque se puede configurar como 3 opciones de memoria RAM.

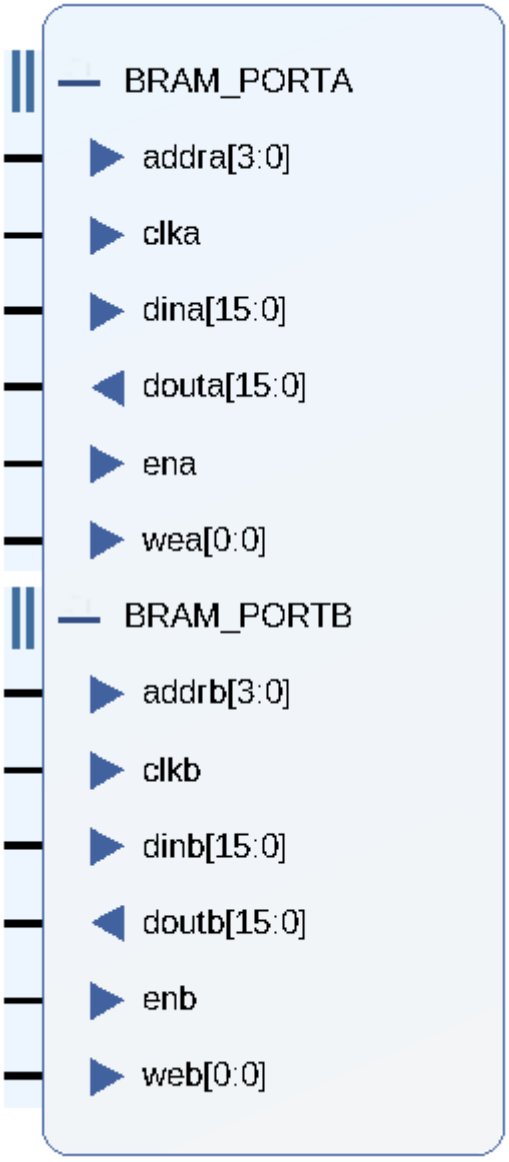
- *Simple Port RAM*: Es la opción de memoria RAM que te permite crear una memoria de lectura y escritura por el mismo puerto de la memoria.



- *Single Dual Port RAM*: es la opción que te permite crear una memoria RAM de dos puertos, solo que esta memoria está capada, para que el canal A solo sea de escritura y el canal B solo sea de lectura.



- *True Dual Port RAM*: esta opción es la memoria RAM de dos puertos más real que hay. Esta opción permite configurar ambos puertos como lectura y como escritura.

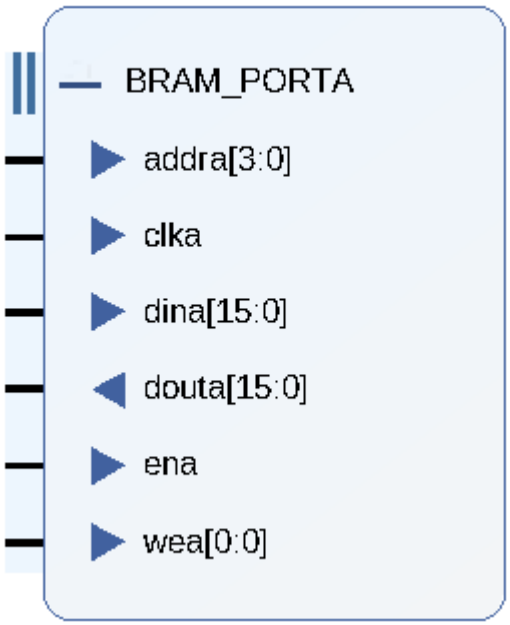


Ahora una vez visto, vamos a entrar en entender todas estas opciones más en detalle.

Single Port RAM

Como he comentado antes esta opción permite tener una memoria RAM de un puerto, con escritura y lectura en el mismo puerto.

El modelo más básico es el siguiente:



Los puertos básicos son:

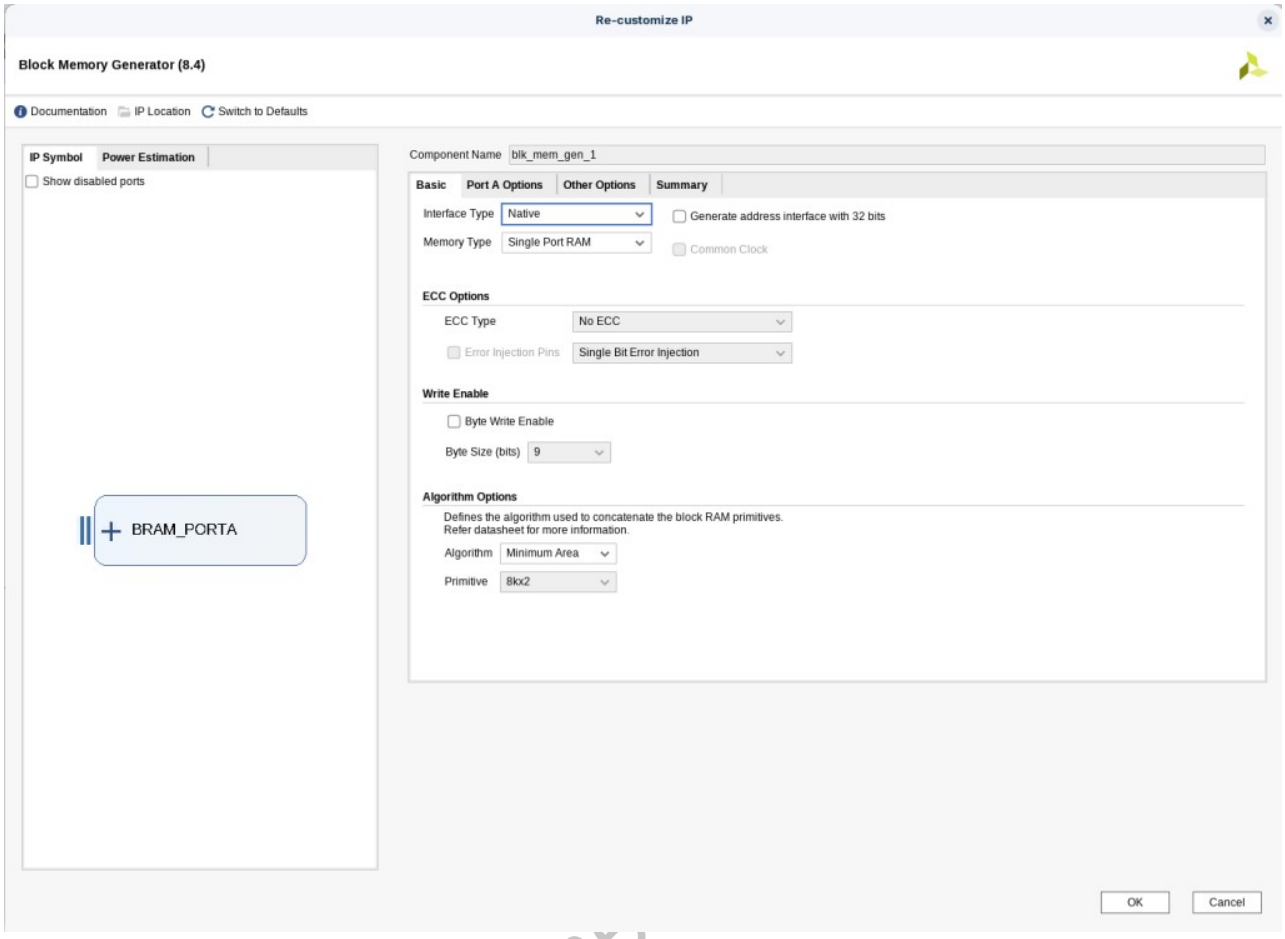
- *addra*: puerto de direcciones de memoria, depende del tamaño del que se quiera la memoria, se configura internamente.
- *clka*: reloj de la memoria, este reloj se utiliza para almacenar datos en cada flanco de reloj positivo.
- *dina*: este puerto es el puerto de entrada de datos, dependerá del tamaño de los datos que se quiera almacenar.
- *douta*: este puerto es el puerto de salida de datos, depende del tamaño de los datos almacenados.
- *ena*: es el puerto de habilitación de la memoria RAM, este puerto se puede dejar activo en la propia configuración del bloque. Activo a nivel alto.
- *wea*: este puerto es que permite seleccionar si se quiere escribir en la memoria o se quiere leer de la memoria. Para la escritura es a nivel alto, para la lectura a nivel bajo, para ninguno de los dos, deshabilitar la memoria RAM(*en el puerto ena*)

Configuración del Single Port RAM

Para configurar el *Block Memory Generator* se tiene que seleccionar la opción en *Memory Type* como «**Single Port RAM**» y la interfaz como «**Native**» (también se puede configurar con una interfaz tipo AXI). También, se puede configurar para optimizar recursos como el área, la potencia o utilizar primitivas fijas.

Nota: la opción *Generate address interface with 32 bits*, es una opción qu permite generar una memoria de tipo coherente, es decir que los datos que maneja son de 32 bits en bloques de 4 bytes, de tal forma que el write-enable es de 4 bits, uno por cada bloque.

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

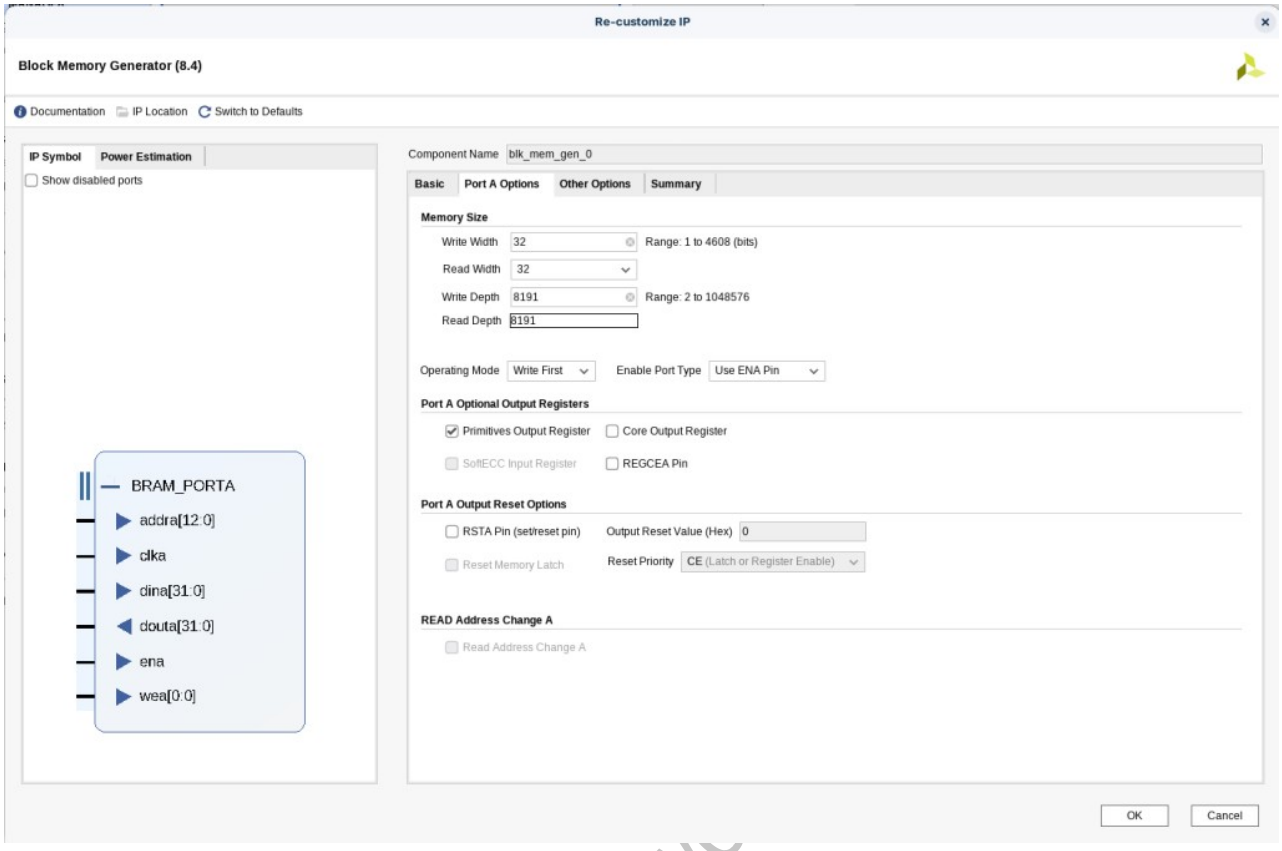


En la pestaña *Port A options* se configuran los tamaños de los datos y de la memoria. Además de otras propiedades accesorias.

El tamaño de los datos es el mismo para escritura y para lectura (*width*). El tamaño de la memoria en bits se calcula en función de la profundidad en datos de la memoria, por lo que en la casilla *Write Depth* se escribe el número de datos a almacenar.

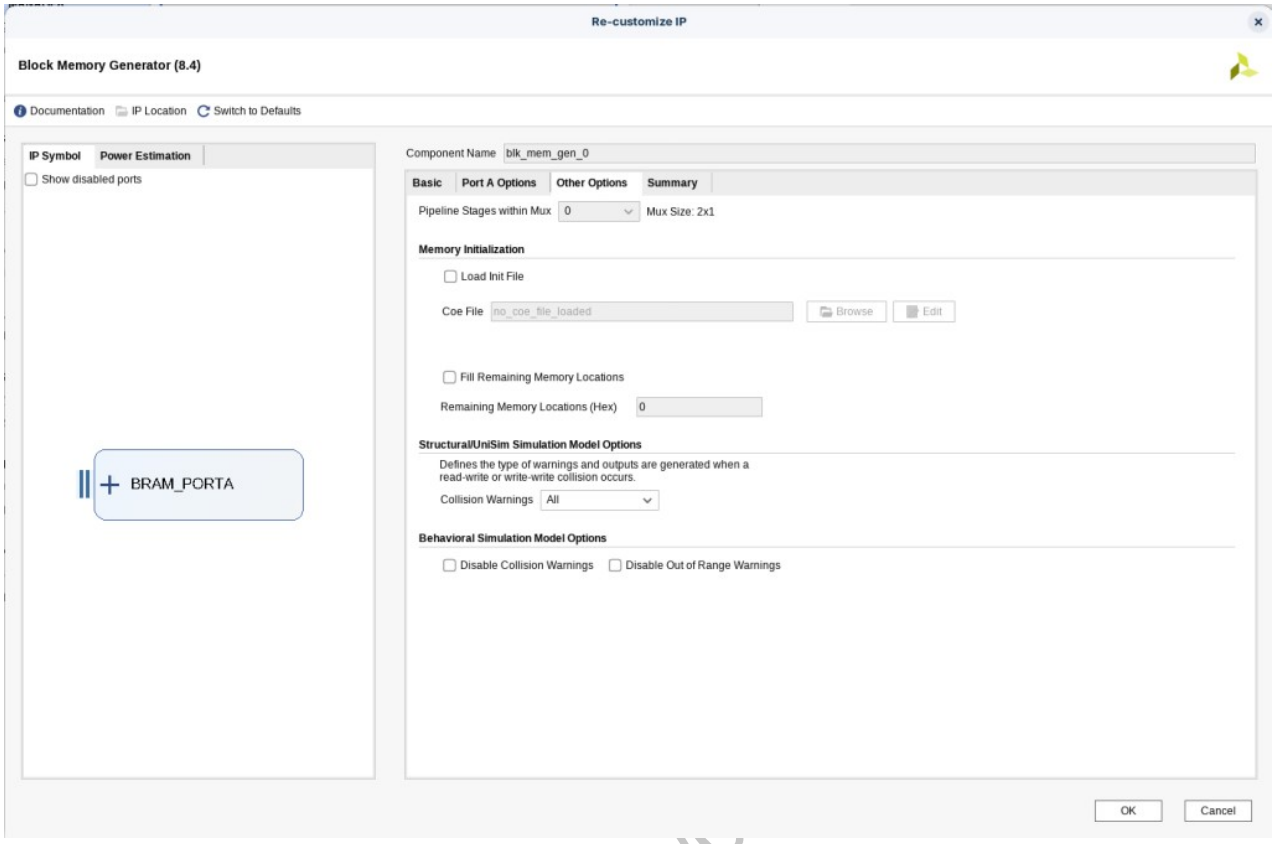
Luego aparecen dos opciones interesantes, *Operating Mode*, *(es una opción que tiene más sentido en memorias de doble puerto para evitar colisiones, cosa que hablaremos en su sección)* en las memorias de un solo puerto esta opción, si se configura, ya sea como **Write First** o **Read First**, hace que los datos que se le da a la entrada salgan por la salida en el siguiente ciclo de reloj. Para evitar que esto ocurra se tiene que configurar como **No Change**, así la salida queda bloqueada mientras se escribe en la RAM. Y luego la opción *Enable Port Type*, que la que permite dejar el bloque de la RAM siempre habilitado, **Always Enabled**, o que tenga un puerto para ello, **Use ENA Pin.**

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



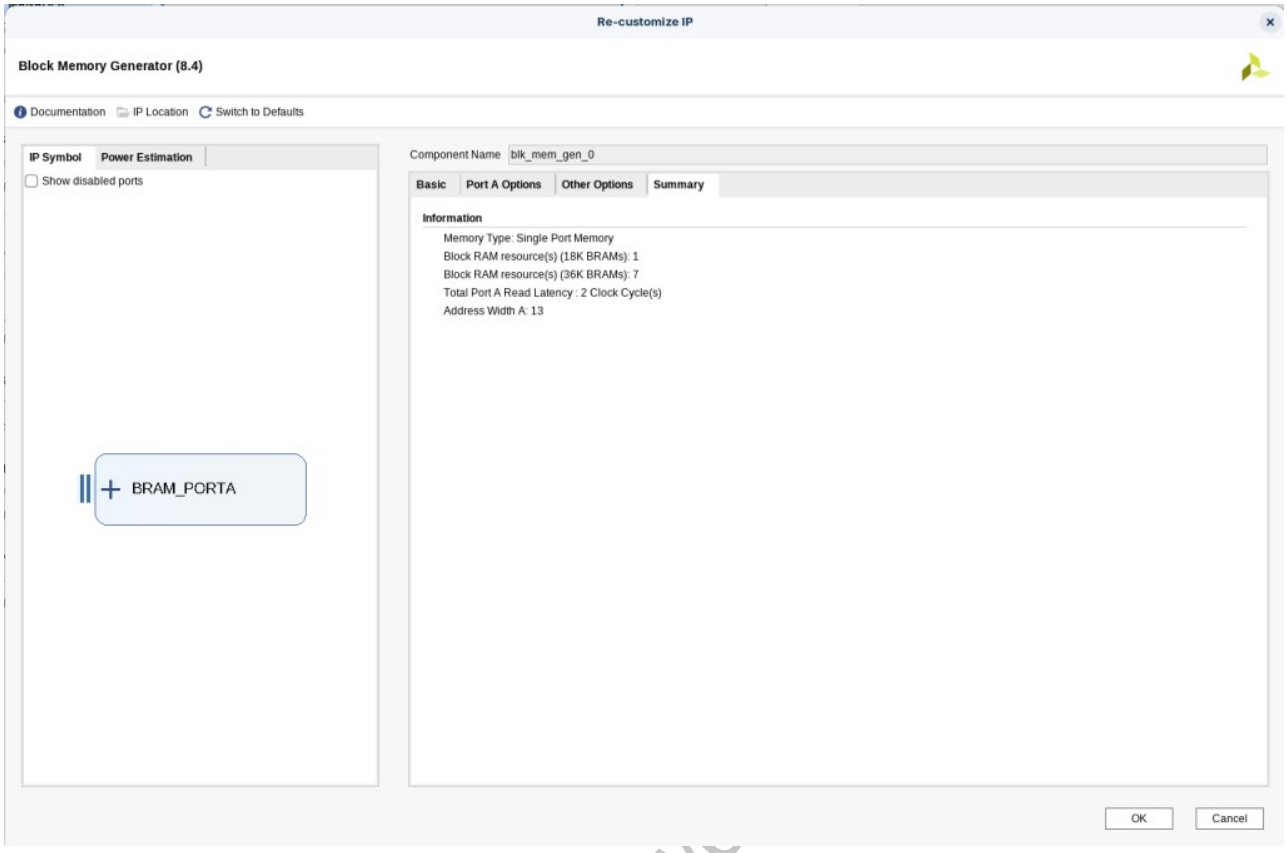
La pestaña *Other Options*, que permite dejar datos grabados en la memoria generada (solo para memorias ROM). También, permite rellenar aquellas direcciones que el usuario no va a utilizar, pero que al crear la memoria se crean. Y desactivar opciones de simulación.

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

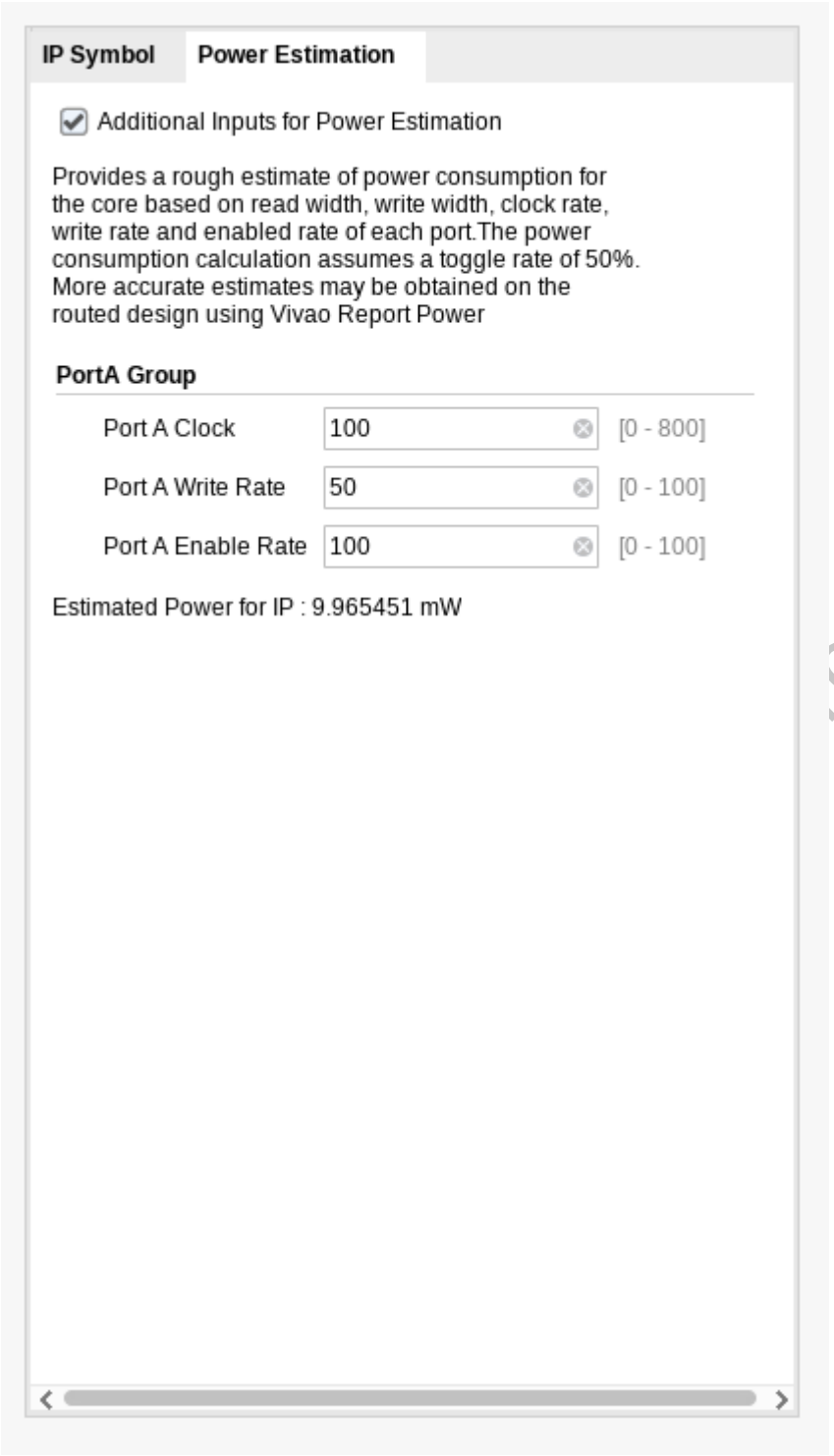


Y la última pestaña es un resumen de la memoria creada, es importante ver que hay un parámetro que pone ***Total Port A Read Latency***, este parámetro determina cuantos ciclos de reloj tarda la memoria en modo lectura en devolver los datos.

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



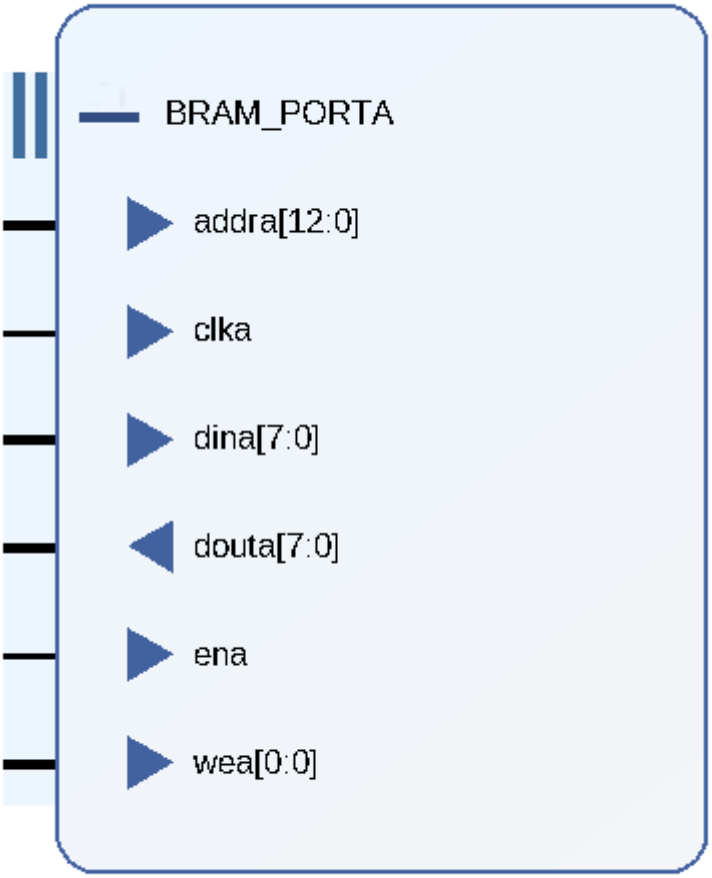
Y para terminar hay una opción que se llama *Power Estimation* que indica cuanta potencia consumiría la memoria. Pide parámetros como la frecuencia de reloj, la tasa de datos de escritura, o la tasa de habilitaciones del bloque (es una estimación).



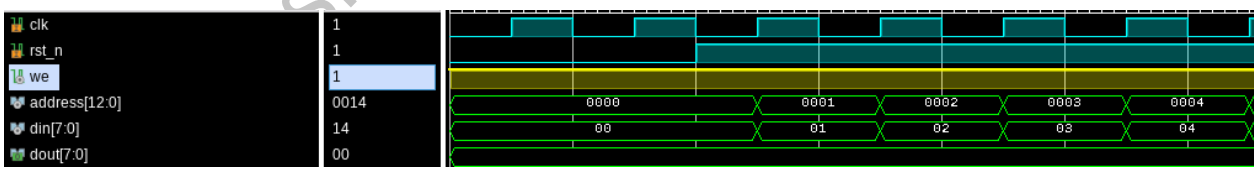
Ejemplo de Single Port RAM

Imaginemos que queremos escribir los datos de un contador en un memoria RAM de forma consecutiva. Empezando con el dato 0x00 en la posición 0x00, el dato 0x01 en la posición 0x01, así hasta completar una memoria de 13 bits.

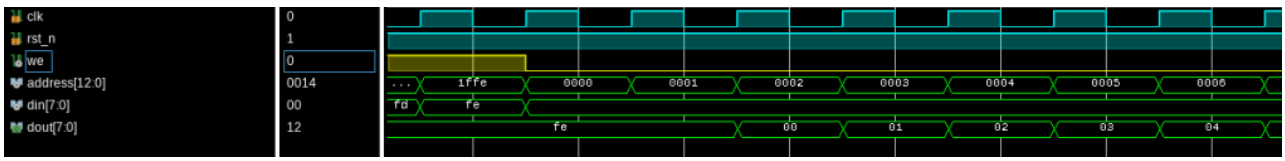
Configuramos el bloque de para tener 13 bits de memoria.



Y simulamos. Al simular se puede ver que el dato que queremos escribir, *din*, se tiene que cambiar durante el primer flanco de subida, y el we tiene que estar a nivel alto. Mientras escribimos el dato de la salida está fijo.

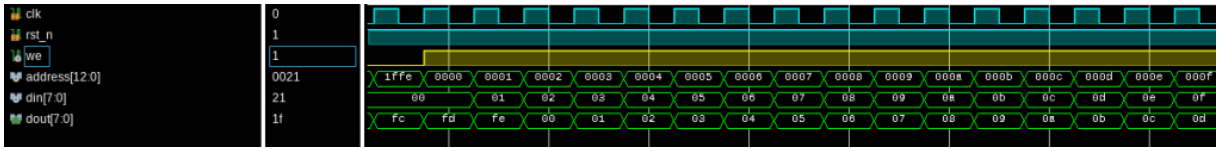


Y si ahora, miramos la lectura de los datos guardados, *dout*, coincide que los datos que se han escrito previamente. También, se puede ver que la memoria tarda dos ciclos de reloj en leer la memoria, cosa a tener en cuenta (este número de ciclos coincide con lo que ponía el resumen de la configuración del bloque IP).



NOTA: Otra cosa que se ve, es que durante la escritura la memoria deja fijo en la salida el último dato leído en la lectura anterior. Esto es debido a que se ha configurado la opción **No Change** en el bloque IP.

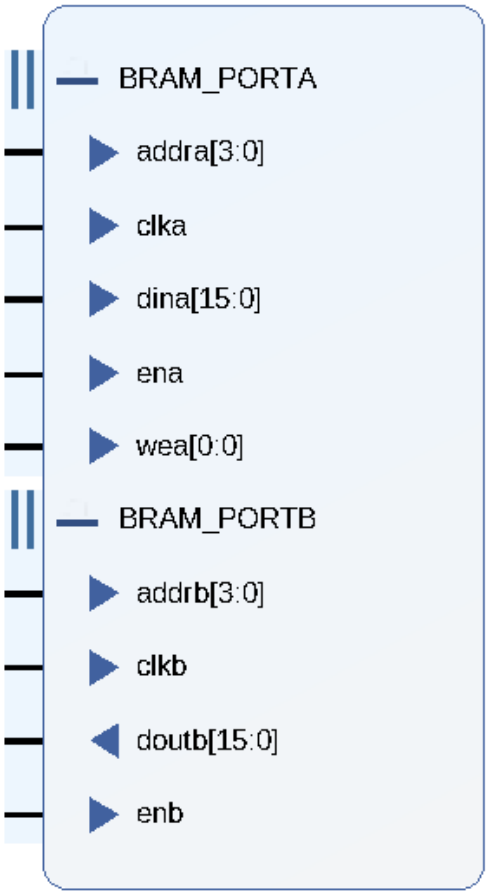
Si se configura con las opciones **Write First** o **Read First**, los datos que se escriben en la memoria se reflejan dos ciclos después a la salida *(como una especie de FIFO de dos posiciones).*



Simple Dual Port RAM

Esta opción es la que permite crear memorias RAM de doble puerto, pero con un solo puerto de escritura y el otro de lectura.

El modelo básico es el siguiente.



En este modelo se pueden ver los diferentes puertos:

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

BRAM_PORTA:

- *addra*: dirección de memoria de escritura
- *clka*: reloj de escritura, se escribe cada flanco de subida
- *dina*: puerto para la escritura del dato en memoria
- *ena*: habilitación del puerto de escritura
- *wea*: este puerto es el que habilita la escritura o la lectura de la memoria, de tal forma que no se puedan hacer a la vez

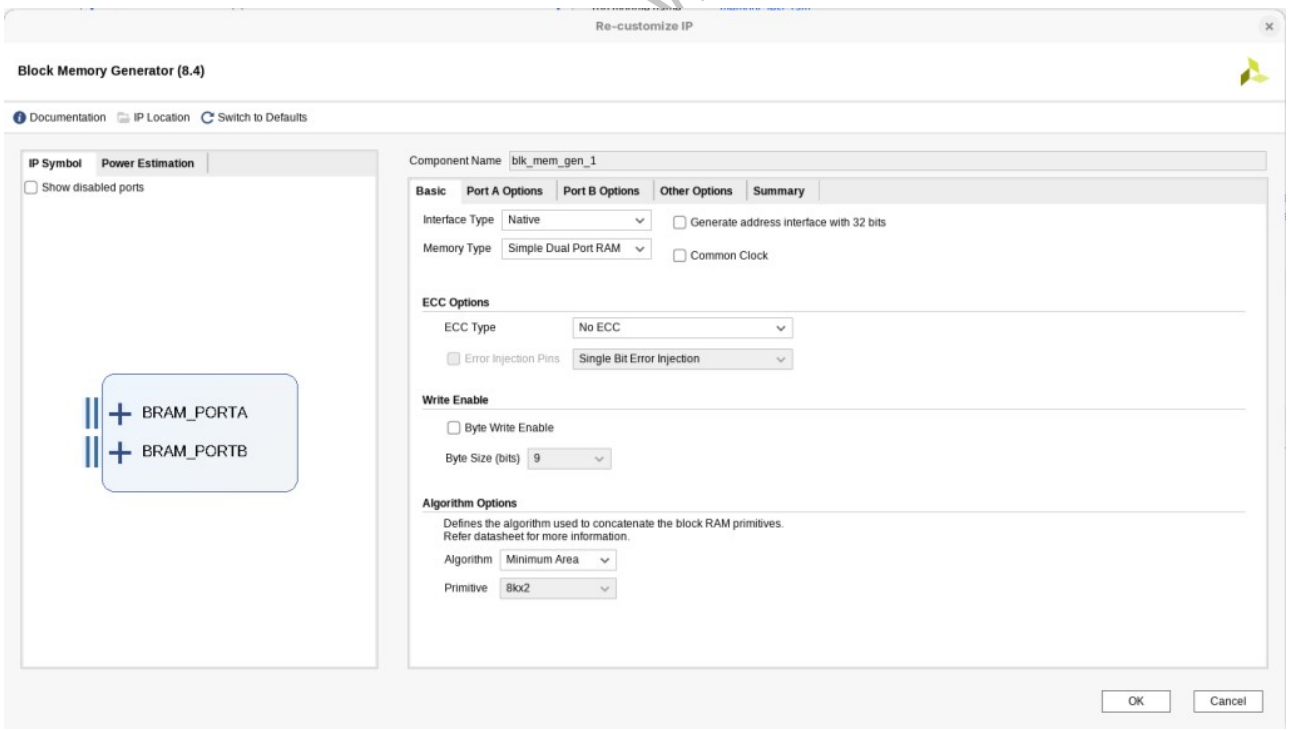
BRAM_PORTB:

- *addrb*: dirección de memoria de lectura
- *clkb*: reloj de lectura, se lee cada flanco de subida
- *doutb*: puerto para la lectura del dato de memoria
- *enb*: habilitación del puerto de lectura

Configuración del Simple Dual Port RAM

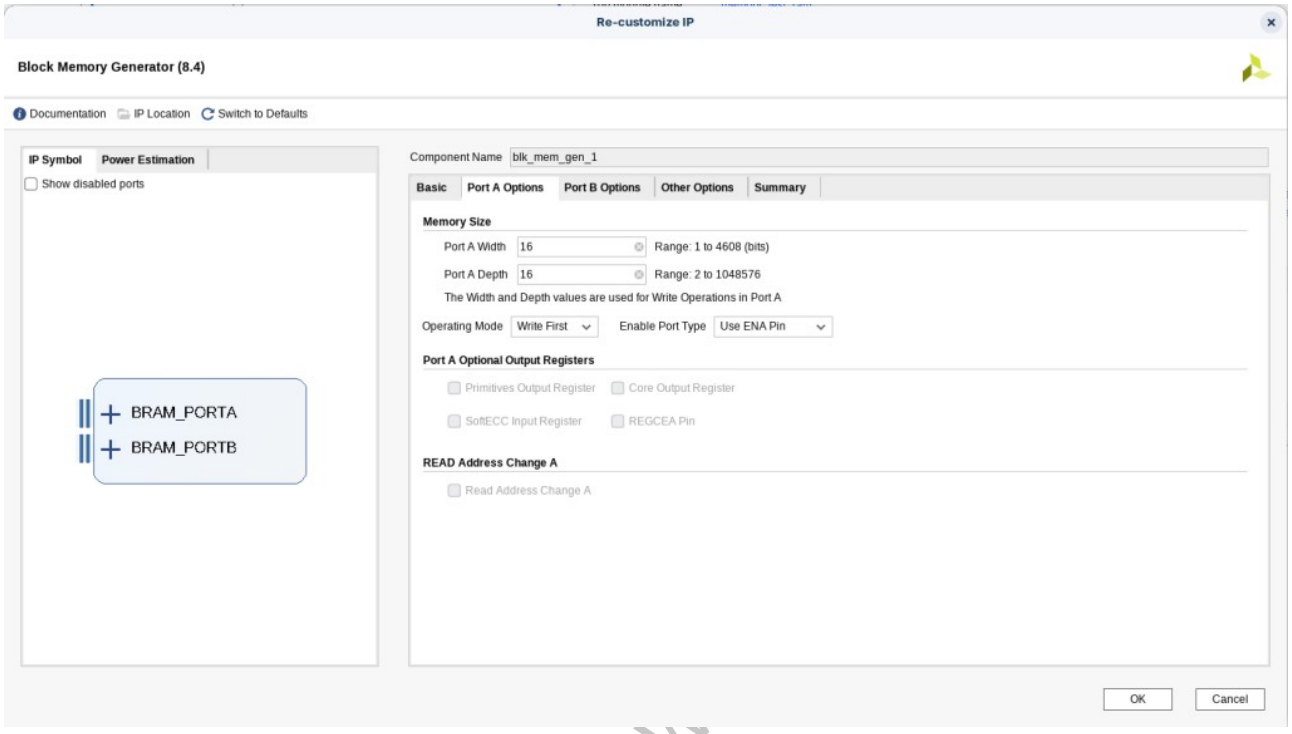
Para la configuración es igual que para la Single Port RAM, por lo que se omitirán detalles repetidos *(se recomienda mirar cómo se hace la configuración anterior)*.

Para a configuración se tiene que seleccionar en Memory Type «Simple Dual RAM». Si se marca la casilla de *Common Clock* todas las operaciones se realizan con el reloj del puerto A.

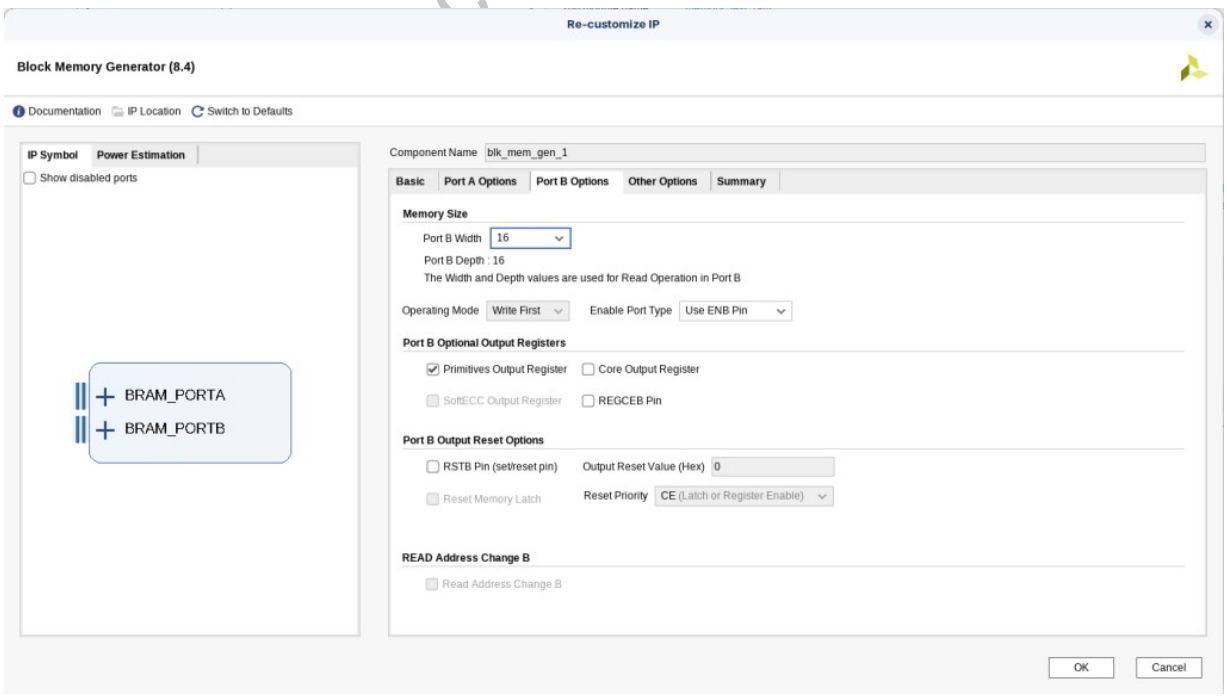


En la configuración del puerto A, se configuran los tamaños de los datos y la profundidad de la memoria. También, se configura si este puerto de la memoria está siempre habilitado. Y también se configura el *Operating Mode*, este modo es el que se utiliza para evitar colisiones en la memoria,

diciéndole a la memoria quién tiene prioridad en caso de que los dos puertos soliciten en el mismo instante acceder a la dirección de memoria, esta opción no tiene mucha utilidad en esta memoria, porque el puerto B está bloqueado para que tenga prioridad la escritura.



Para el puerto B se configura solo el tamaño de los datos de salida y si siempre está habilitada la salida. El tamaño de la memoria lo elige el puerto de escritura, y está configurada siempre como «**Write First**«, eso significa que si los dos puertos quieren acceder a la misma memoria en el mismo instante, el puerto de escritura, *puerto A*, tiene prioridad sobre el puerto de lectura, *puerto B*.



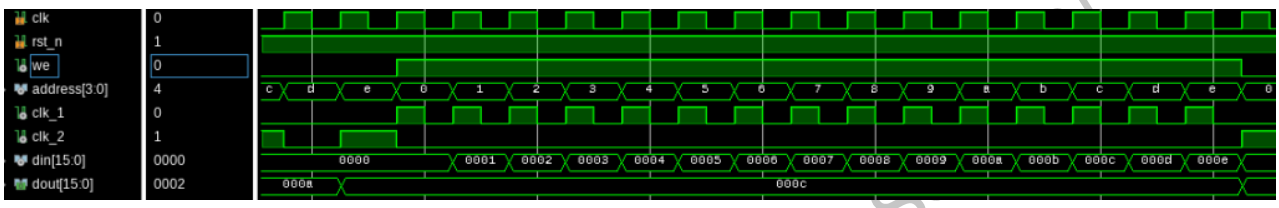
https://soceame.wordpress.com/2024/11/03/como-implementar-memorias-en-vivado-parte-1-memoria-ram/

El resto de las configuraciones de este bloque IP ya se han explicado en el Single Port RAM.

Ejemplo de Simple Dual Port RAM

Vamos a hacer un ejemplo como el anterior, pero esta vez con dos sistemas de reloj diferentes para cada puerto. El puerto A tiene el doble de frecuencia que el puerto B. Y la señal de direcciones de memoria a la que acceden los puertos es la misma. Solo que el reloj de lectura o de escritura solo están activos durante el momento en el que estén operativos.

Si miramos la escritura, *we='1'*, vemos como cada ciclo de reloj las direcciones y el dato a escribir(*din*), van a la par.



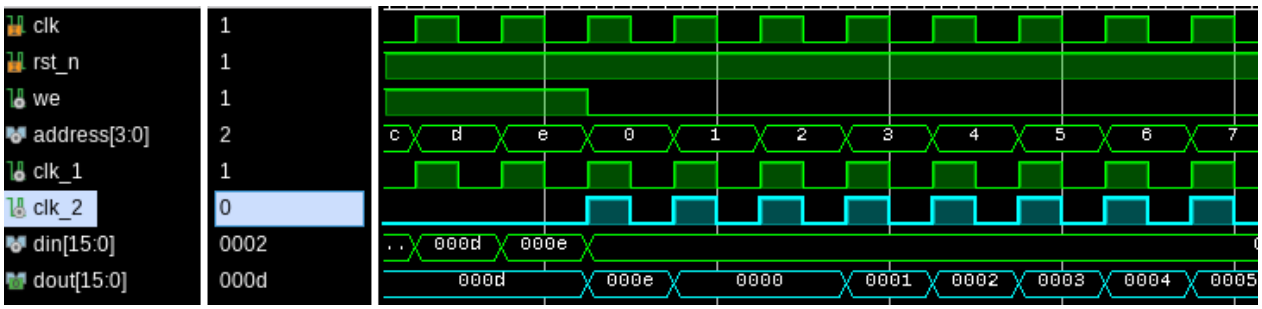
Y para la lectura, el reloj que se utiliza para la lectura es el *clk_2*, pero el reloj que cambia las direcciones de memoria es el reloj *clk*.

Se puede ver que hay un retardo de un ciclo de reloj de *clk_2*. Teóricamente, según el bloque son dos ciclos de reloj *(recomendable verificar el comportamiento real que puede tener con dos relojes de distinta frecuencia)*.

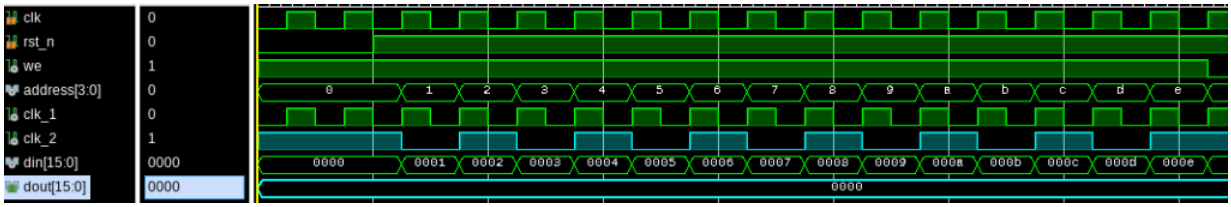
Information

Memory Type: Simple Dual Port RAM
Block RAM resource(s) (18K BRAMs): 1
Block RAM resource(s) (36K BRAMs): 0
Total Port B Read Latency (From Rising Edge of Read Clock): 2 Clock Cycle(s)
Address Width A: 4
Address Width B : 4

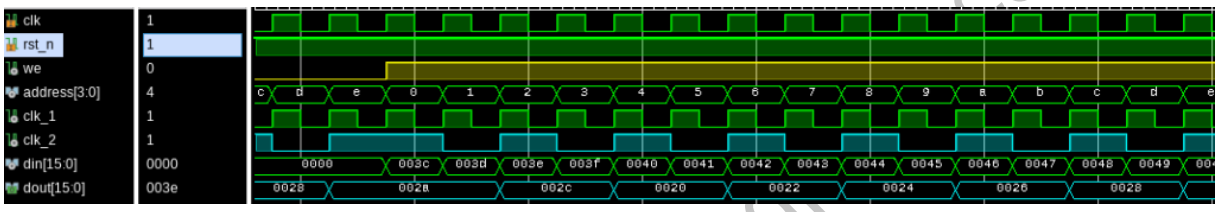
Aunque, si se utilizase el mismo reloj para la lectura y para la escritura, sí que son dos ciclos de reloj.



NOTA: si queremos que los dos puertos colisionen de tal forma que la escritura y la lectura vayan a la misma dirección de memoria al mismo tiempo, lo que ocurre es que al principio, como los datos de la memoria son ‘0’, devuelve todo ‘0’.



Pero cuando las direcciones están ya escritas, devuelve los valores almacenados previamente en memoria, mientras el puerto de escritura sigue escribiendo nuevos datos en memoria. Entonces, se produce un efecto en el que primero se lee el dato que está en la memoria, y después, se escribe en esa posición.

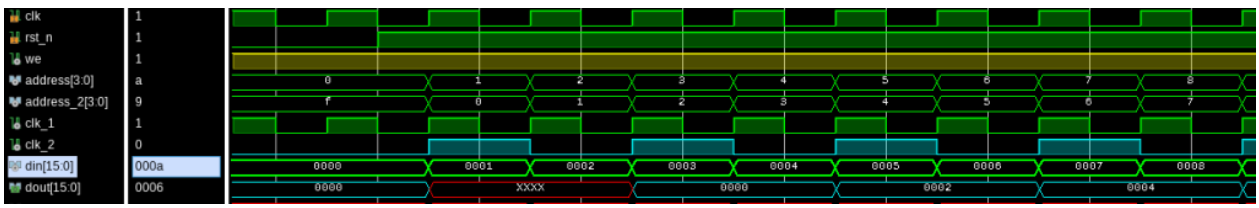


Esto es debido a que si se quiere utilizar una memoria **Simple Dual Port Memory**, con un **mismo reloj de entrada y salida**, el puerto A se tiene que configurar como **Read First**, para evitar posibles problemas. En caso de utilizar un reloj **asíncrono**, Xilinx, recomienda configurar al puerto A como **Write First**.

> For Synchronous Clocking and during a collision, the Write mode of port A can be configured so that a Read operation on port B either produces data (acting like READ_FIRST), or produces undefined data (Xs). For this reason, it is always advised to use READ_FIRST when configured as a Simple Dual-port RAM. For asynchronous clocking, Xilinx recommends setting the Write mode of Port A to WRITE_FIRST for collision safety. For detailed information about this behavior, see Collision Behavior, page 51.

NOTA: Esto último es un poco caótico, entonces, se recomienda en la medida de lo posible, no leer la memoria mientras está se está escribiendo, debido al riesgo de colisión. Y dejar algún ciclo de reloj previo antes de leer el dato recién grabado.

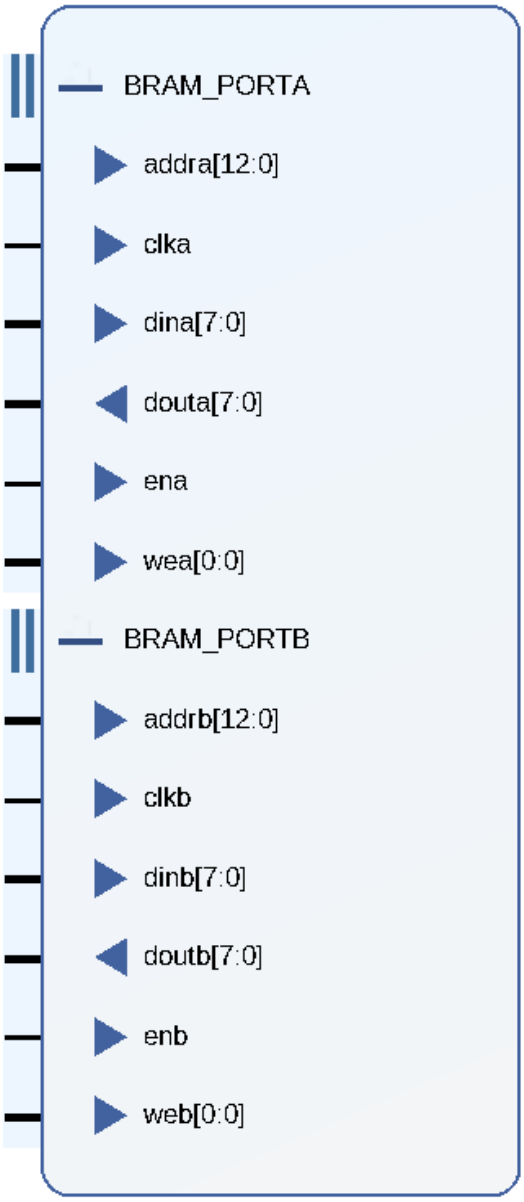
El mínimo tiempo que se tiene que dejar entre el momento en el que se escribe y el que se puede leer ese dato es de 1 ciclo de reloj de entrada, aunque se puede producir una colisión en la señal de salida al comenzar a funcionar.



True Dual Port RAM

Esta es la última memoria de tipo RAM que nos deja configurar el *Block Memory Generator*. Esta es una memoria RAM de dos puertos real. Eso significa que hay dos puertos que tienen entradas y salidas diferentes.

Esta memoria se aproxima más a las memorias RAM que existen en la realidad.



Esta memoria consta de dos puertos, el A y el B, con sus respectivos puertos de señal.

BRAM_PORTA:

- *addra*: puerto de direcciones de memoria del puerto A
- *clka*: reloj del puerto A

- *dina*: puerto de entrada de datos del puerto A
- *douta*: puerto de salida de datos del puerto A
- *ena*: puerto de habilitación del puerto A
- *wea*: puerto de habilitación de la escritura del puerto A

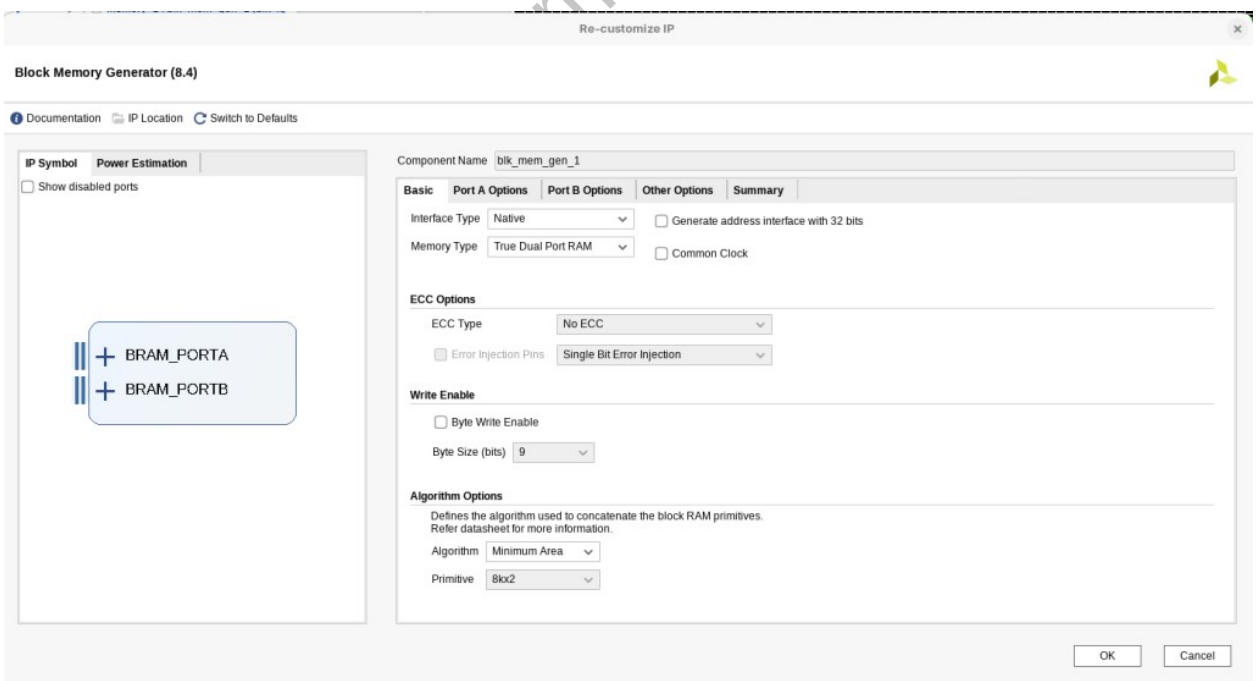
BRAM_PORTB:

- *addrb*: puerto de direcciones de memoria del puerto B
- *clkb*: reloj del puerto B
- *dinb*: puerto de entrada de datos del puerto B
- *doutb*: puerto de salida de datos del puerto B
- *enb*: puerto de habilitación del puerto B
- *web*: puerto de habilitación de la escritura del puerto B

Configuración de True Dual Port RAM

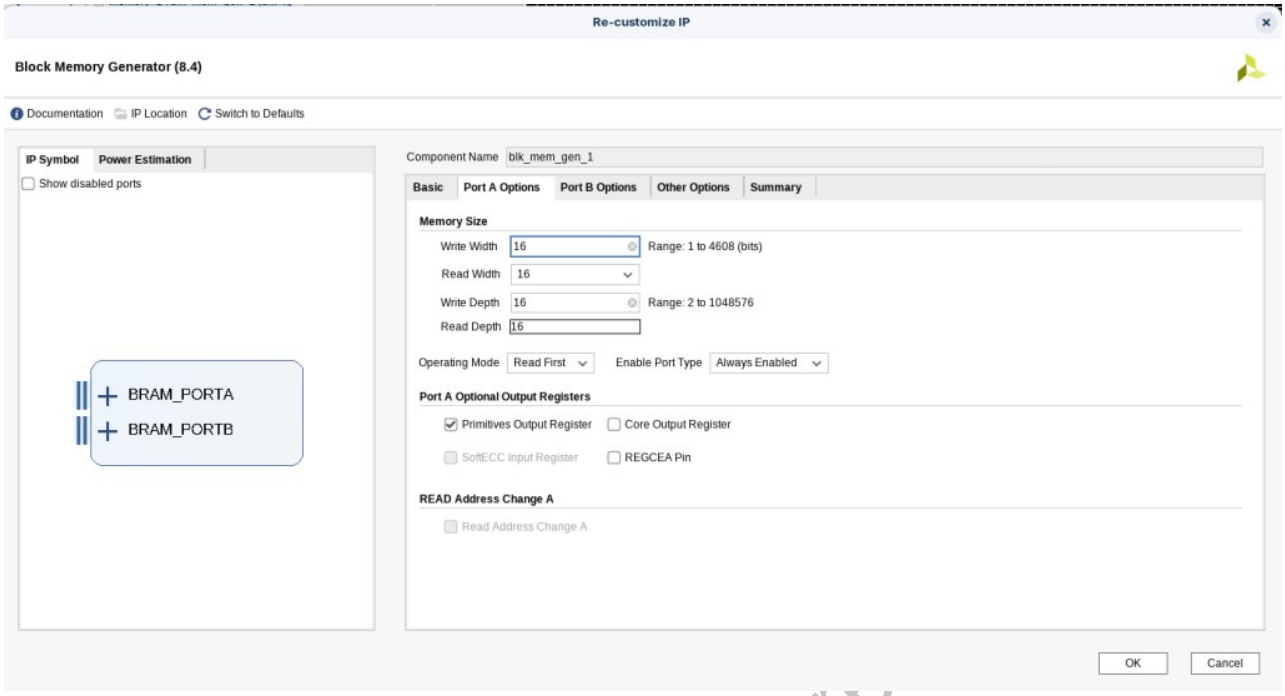
La configuración de una **True Dual Port RAM** es igual que la un **Simple Dual Port RAM**, solo que cambiando algunos detalles.

El *Memory Type* se configura como **True Dual Port RAM**. Si se marca la casilla de *Common Clock* todas las operaciones se realizan con el reloj del puerto A.

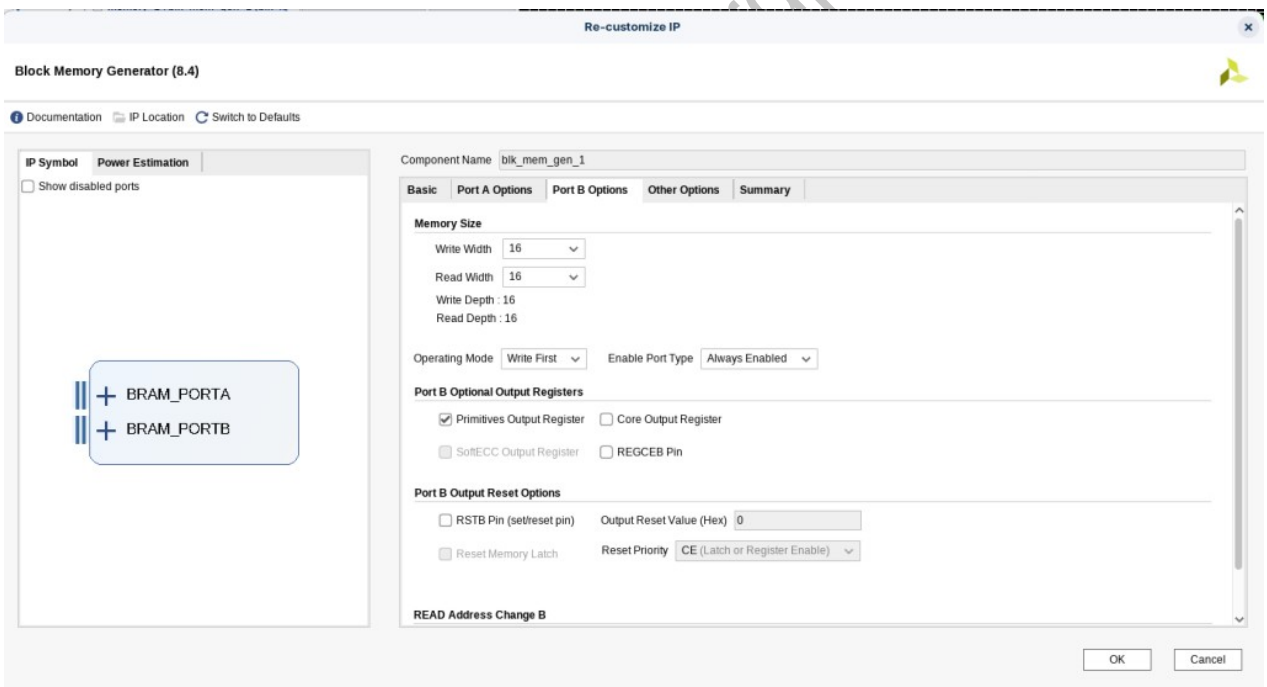


Y para el puerto A, es igual que todas las RAM anteriores.

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



Y para el puerto B, también es igual.

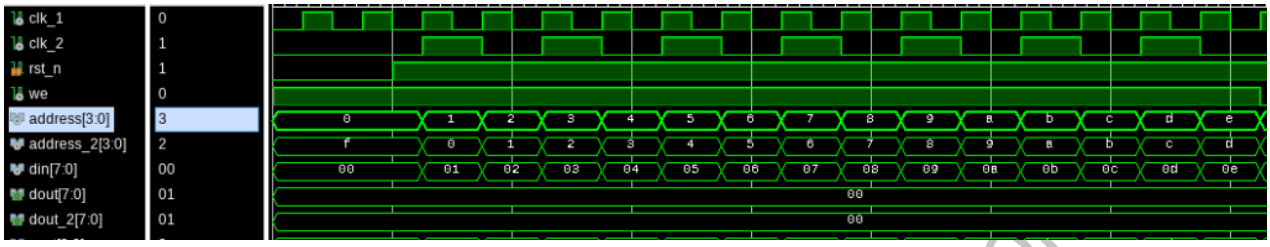


Pero cobra para los dos puertos, especial relevancia debido a que ahora hay dos direcciones de memoria que pueden escribir en la misma posición, cosa que antes estaba muy limitado. Por eso, **es imperativo que ninguna de las dos direcciones de memoria escriban en el mismo sitio en el mismo instante de tiempo**.

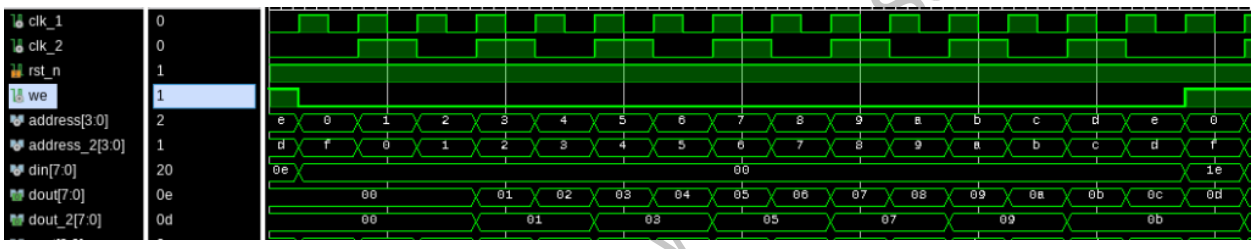
Ejemplo de True Dual Port RAM

Se va a realizar el mismo ejemplo que es los casos anteriores *(esta vez no se va a hacer colisionar porque pueden pasar cosas catastróficas)*.

Para la escritura de datos se puede ver que se puede escribir de forma simultánea en direcciones de memoria distintas sin generar problemas.

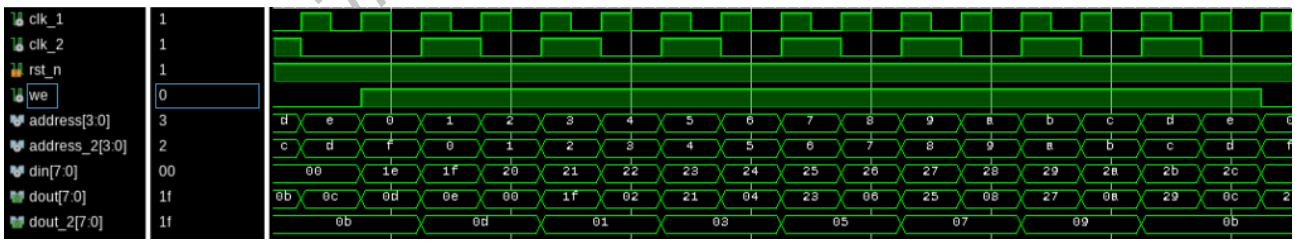


Y para la lectura se puede ver que los datos pueden ser leídos con facilidad, con un retardo de 2 ciclos de reloj.



NOTA: es posible que el primer dato de lectura genere un error, eso es debido a la configuración de la memoria. Si se configura como **No Change**, no genera ese error.

También, es recomendable que se configure como **No Change**, debido a que ahora ocurre lo mismo que en la *Single Port RAM*, si no se configura así los datos de entrada salen a la salida, debido a que se utiliza la misma dirección de memoria para la entrada y para la salida. Entonces, los datos afloran de forma *no deseada* a la salida, mientras se escribe la memoria.



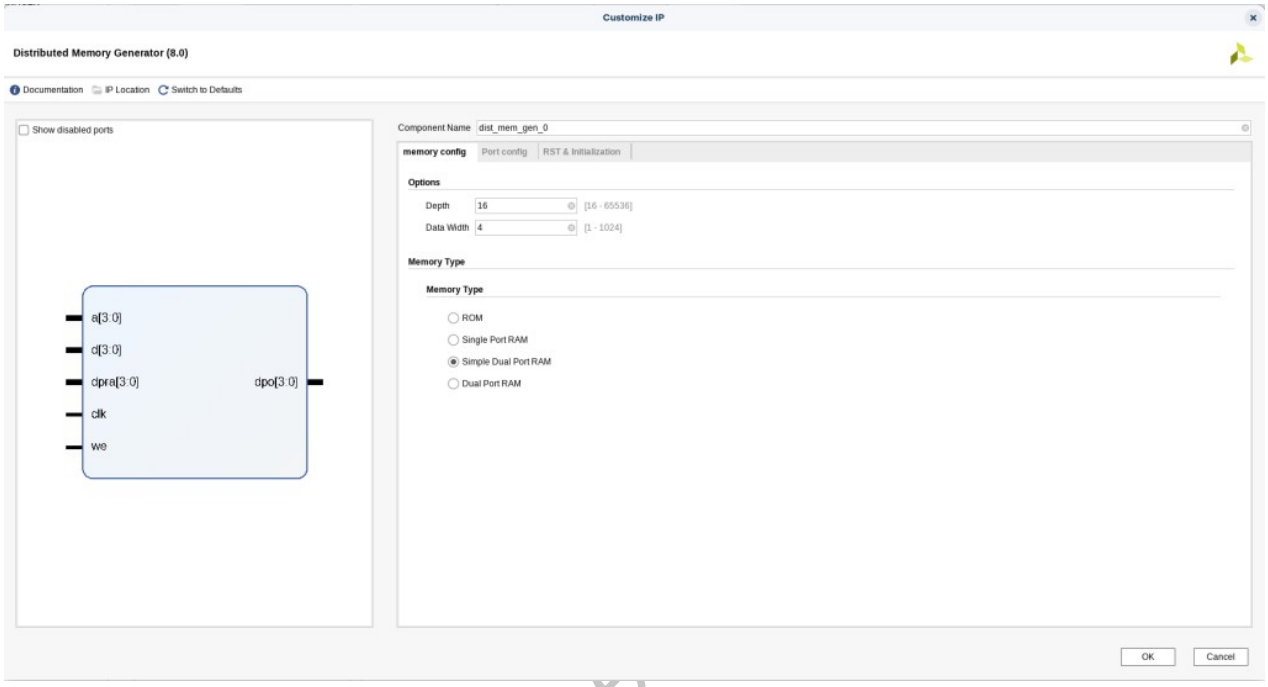
Distributed Memory Generator

Y para finalizar esta entrada queda el último generador de memoria que da Vivado, el *Distributed Memory Generator.* Este bloque IP genera una memoria RAM utilizando la lógica integrada de la FPGA.

La ventaja que tiene este bloque es que es más rápido para hacer lecturas, **tarda 1 ciclo** de reloj mientras que el otro bloque IP tarda 2 ciclos de reloj.

El problema de este bloque es que **tiene varios «males»** que el otro bloque IP permite corregir en su configuración.

Para configurarlo es más simple que el *Block Memory Generator.* Solo admite 4 opciones, 3 de ellas de memoria RAM, cuyo funcionamiento es el mismo que las memorias que se han explicado anteriormente.

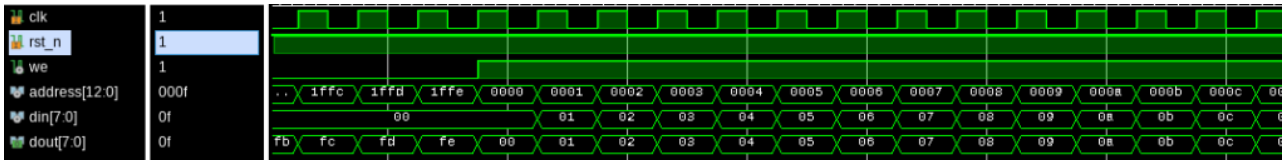


Para configurar la RAM solo es necesario hacerlo desde la pestaña principal, el resto son accesorias.

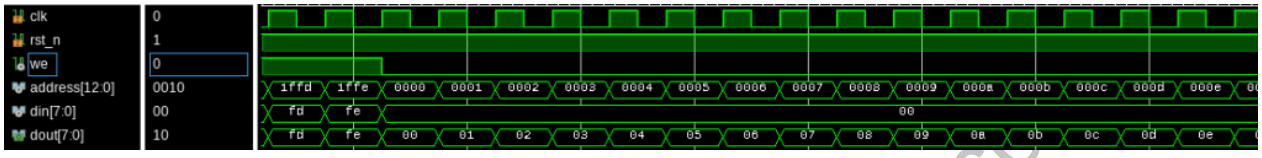
Single Port RAM

Como se puede ver tiene los mismos puertos que tenía el *Single Port RAM* del *Block Memory RAM*.

Si se comprueba el funcionamiento en modo *escritura*, se puede ver que la escritura se realiza, pero también, el dato que se escribe sale también por la salida, como ocurría en el *Single Port RAM* del *Block Memory RAM*, solo que aquí no se puede configurar el modo para que esto no ocurra.



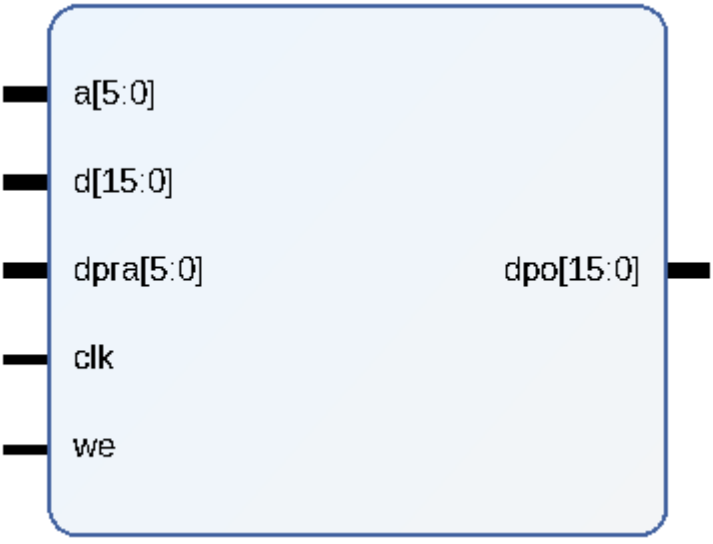
Y para la *lectura*, se puede ver que solo tarda 1 ciclo de reloj en leer la memoria.



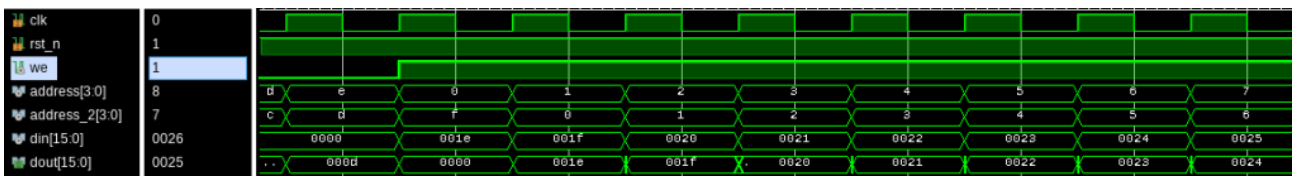
Simple Dual Port RAM

Esta configuración es casi la misma que tiene el *Block Memory RAM*. Solo que solo hay un puerto de direcciones, por lo que solo se puede escribir o leer desde un sitio. La señal *we* es la encargada de controlar si se lee o si se escribe. También, solo hay un reloj.

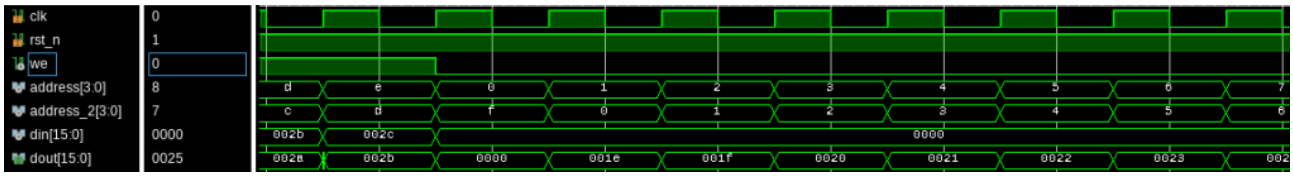
El puerto *dpra* es el que define la dirección del dato de lectura por *dpo*, y *a* define la dirección de memoria sobre la que va a escribir *d*.



Para ver el funcionamiento en *escritura*. Igual que en el Single Port RAM.

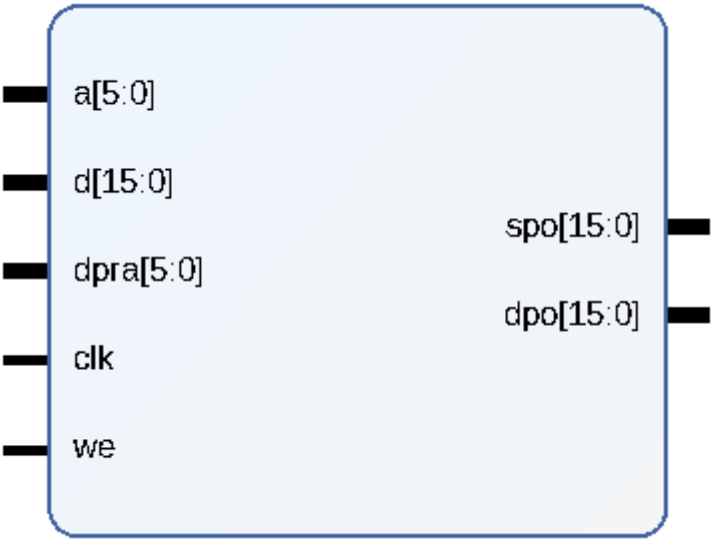


Para ver el funcionamiento en *lectura*. Se puede ver que se tarda un ciclo de reloj en realizar la lectura.

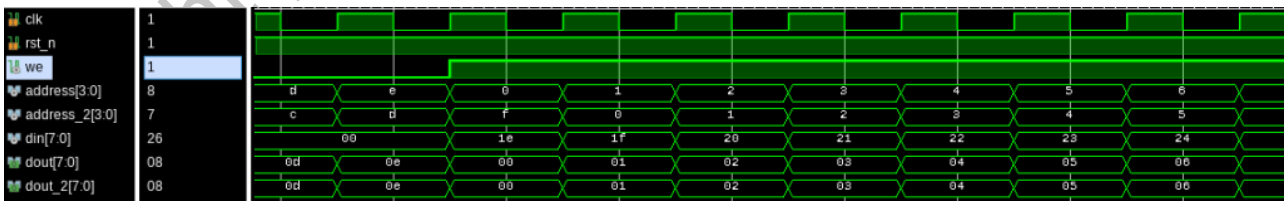


Dual Port RAM

Para esta configuración sí que se producen cambios como el número de puertos, dónde solo hay un write-enable(*we*) para los dos puertos de memoria, y también un solo puerto de escritura de memoria, pero dos salidas de memoria, una para cada puerto de la memoria. Y un solo reloj, por lo que a la memoria solo se accede un sitio para escribirla, pero se puede leer desde dos sitios diferentes.

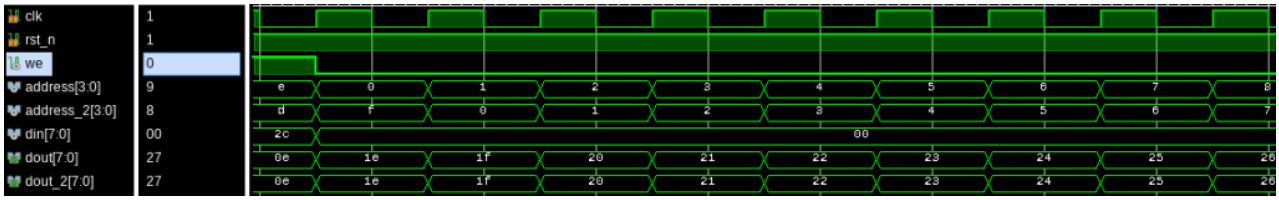


Si queremos ver la *escritura*, también, se puede ver que los datos de entrada salen por la salida un ciclo después como en el Single Port RAM.



Y para ver la *lectura* podemos ver que el dato que sale por la salida sale con un retardo de un ciclo de reloj. El dato para leer es el mismo en ambos puertos, esto es debido a que los dos puertos de salida entregan el mismo dato.

Entonces, el puerto de direcciones *dpra*, emite los mismos datos por los puertos de salida *spo* y *dpo*, y el puerto de direcciones *a* permite escribir los datos en memoria.



NOTA: No es que se emitan los datos con un retraso, es que se emiten los mismos datos sin importar como vayan las direcciones de memoria retrasadas o cualquier otra cosa. Para evitar esto se recomienda *solo leer por uno de los dos puertos.*

Ampliación

(para entender esta ampliación se recomienda haber leído la parte 2 de estas entradas y saber cómo se hace un fichero COE)

Es posible utilizar un COE como fichero base para el inicio de la memoria RAM, de tal forma que si se quiere leer la RAM antes de que se haya grabado un dato, lo primero que se lee es el fichero COE grabado, entonces, este fichero COE tiene que estar dimensionado para contener las posiciones suficientes de la RAM, si el fichero COE es más grande que la RAM, dará error. Luego los valores del fichero COE son sobreescritos en la escritura.

Ejemplo

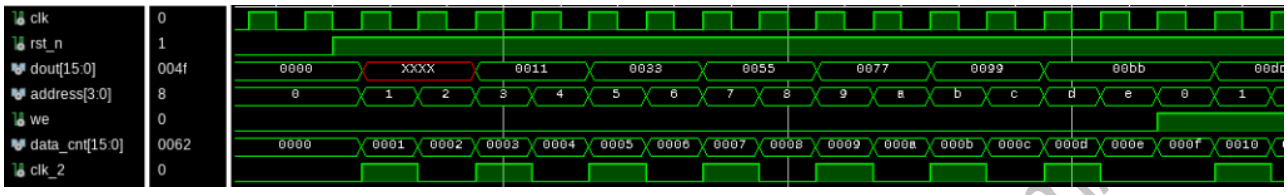
Si yo cargo un COE como el siguiente en la memoria Dual Port RAM.

```
MEMORY_INITIALIZATION_RADIX=16;
MEMORY_INITIALIZATION_VECTOR=
11,
22,
33,
44,
55,
66,
77,
88,
99,
aa,
bb,
cc,
dd,
ee,
ff,
00;
```

Y le digo a la RAM que empiece leyendo (WE=’0′), sin tener ningún dato escrito mandando la orden WE=’1′, lo primero que hace es sacar los datos del fichero COE (0x11, 0x33, 0x55).

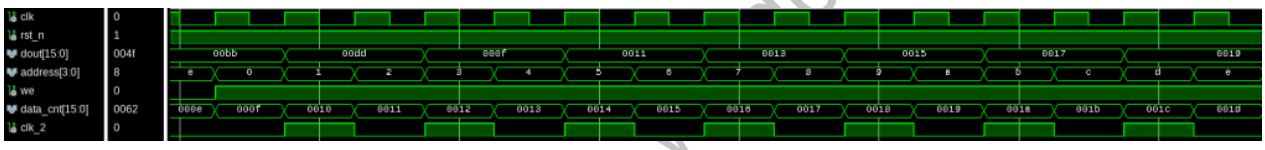
NOTA: el desfase de los datos que entran con los que salen es porque el reloj de salida es más lento que el de escritura, y el contador para los datos de lectura funciona con el reloj de entrada y no con el de salida.

NOTA 2: El primer dato leído es fallido porque en la simulación hay dos relojes que no están lo suficientemente desfasados, eso se tiene que ajustar bien, o utilizar el mismo reloj para entrada/salida.

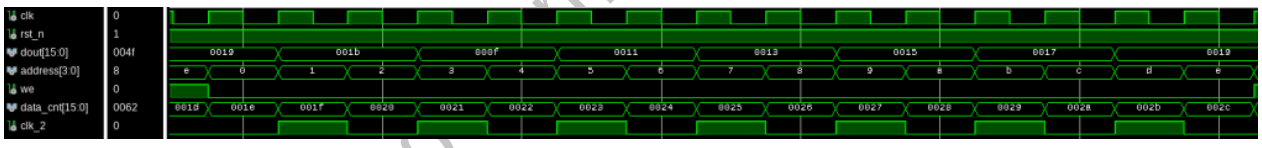


Y si ahora analizamos la lectura y escritura que se produce en los siguientes ciclos de escritura y lectura, se puede ver que primero empieza escribiendo (WE=’1′) bien los valores(0x11, 0x13, 0x15) y cuando se pasa a lectura (WE=’0′) se puede ver que los valores que saca son correctos (0x11, 0x13, 0x15), por lo tanto está sobreescribiendo los datos de arranque del COE.

Escritura



Lectura



Nota final

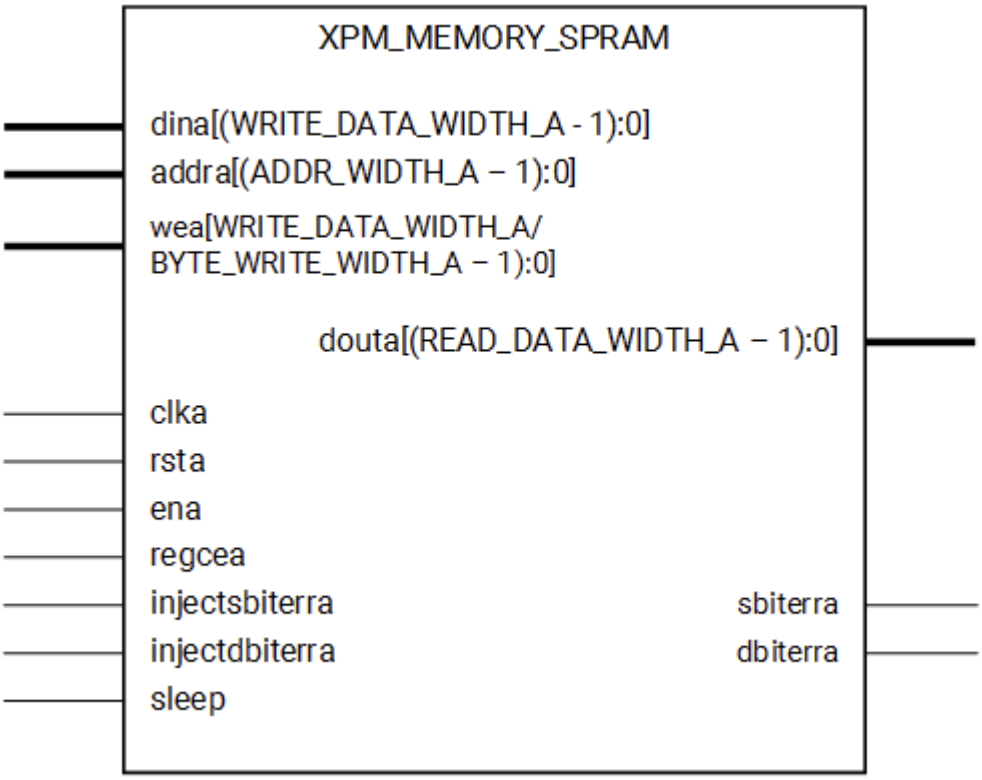
Muchas de memorias están creadas mediante macros internas de Vivado (o sea, llamadas internas a módulos que Vivado no enseña, pero que en la documentación sí que documenta).

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

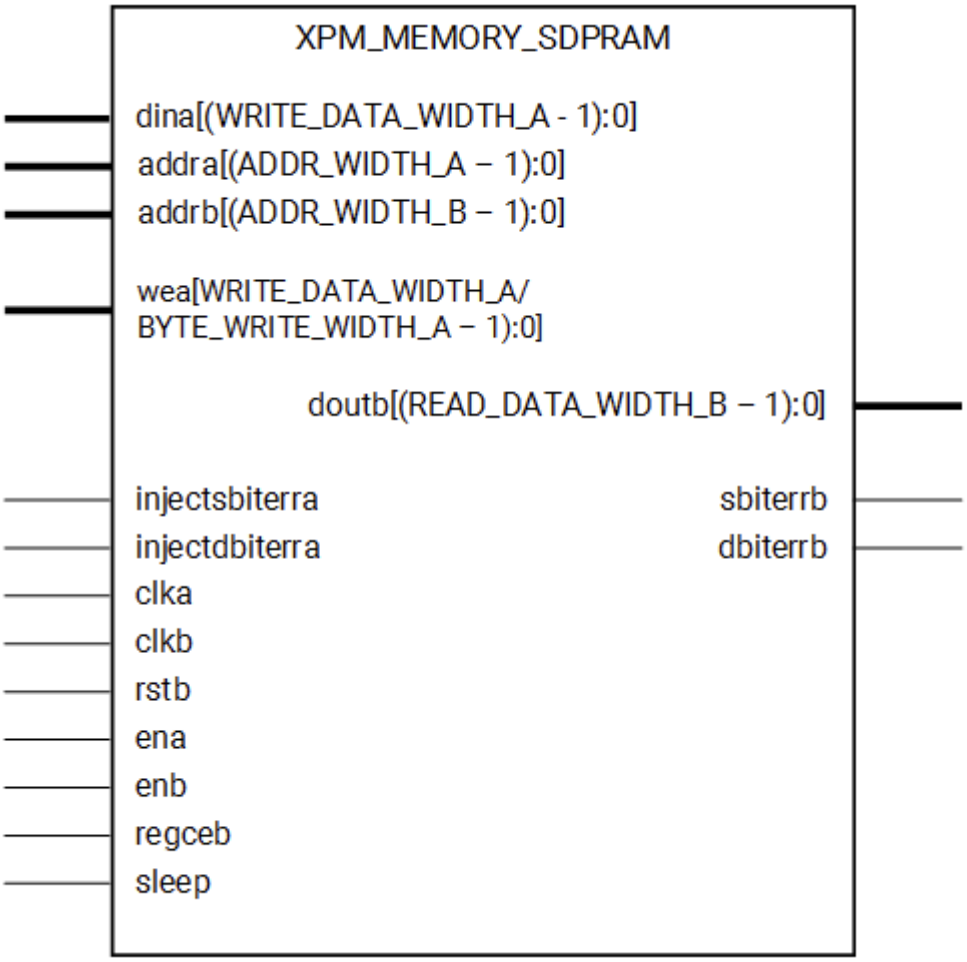
```
Library xpm;
use xpm.vcomponents.all;
```

Y son las siguientes:

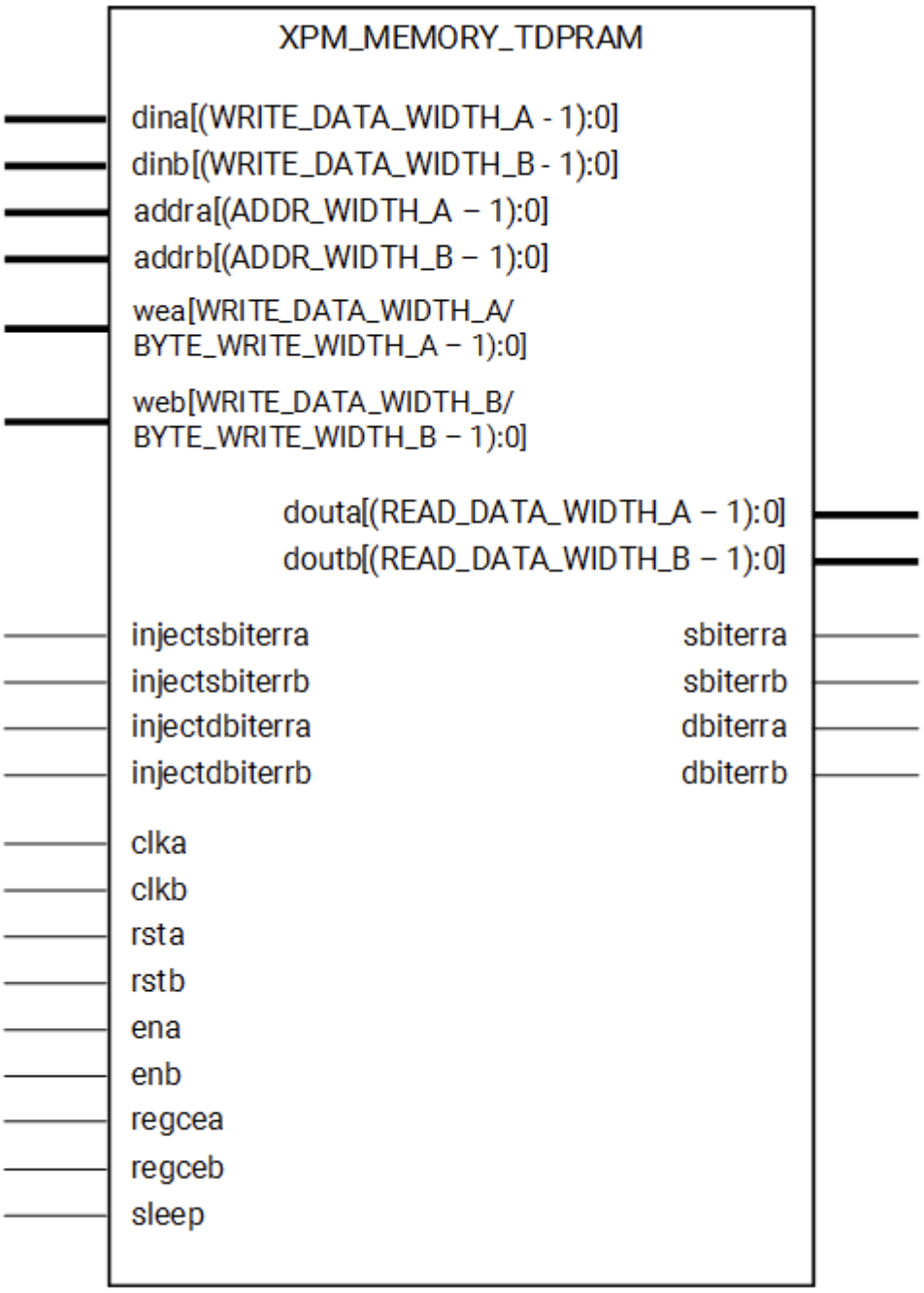
- **XPM_MEMORY_SPRAM**: esta macro hace referencia a las memorias *Single Port RAM*



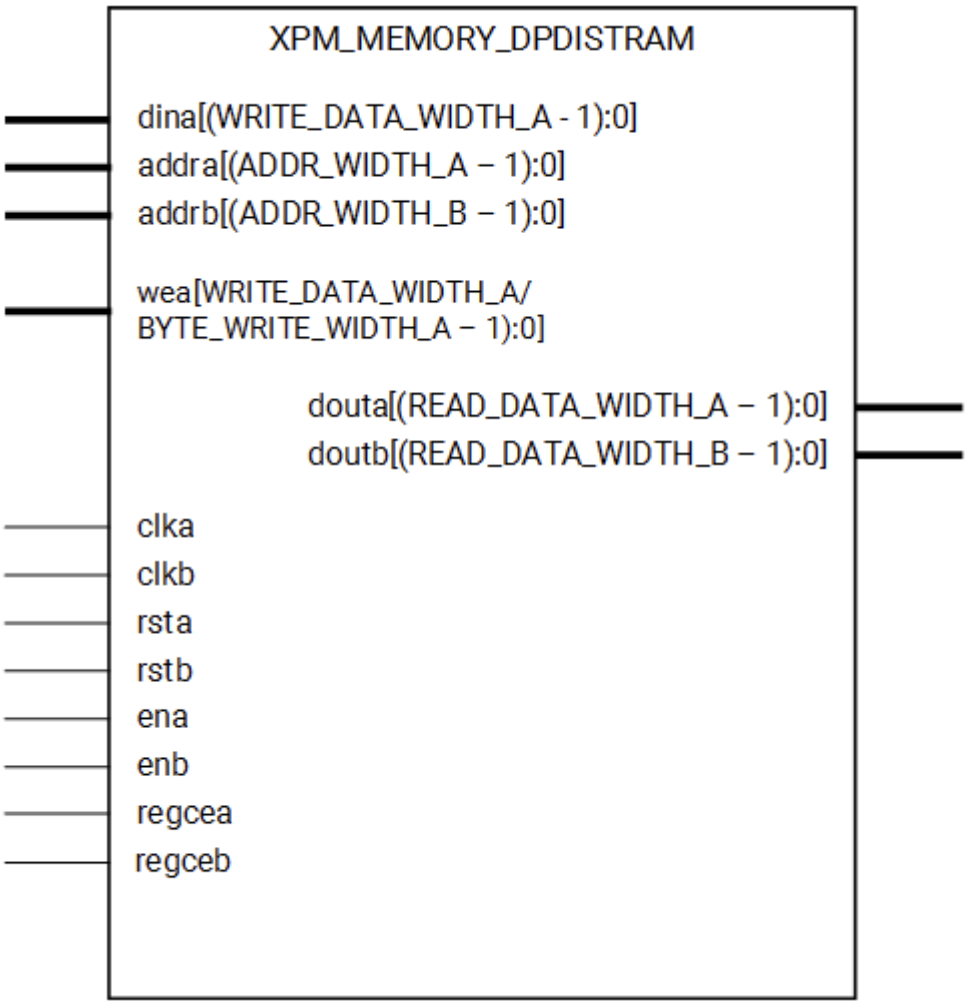
- **XPM_MEMORY_SDPRAM**: esta macro hace referencia a las memorias *Simple Dual Port RAM*



- **XPM_MEMORY_TDPRAM**: esta macro hace referencia a las memorias *True Dual Port RAM*



- **XPM_MEMORY_DPDISTRAM**: esta macro hace referencia a las memorias *Dual Port Distributed RAM*



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