# Informe de progreso – Fernando Capellán Pizarroso

1 – La primera tarea realizada fue crear un repositorio en GitHub donde poder subir el código y la documentación y llevar un control de versiones y de los módulos modificados y probados. También es necesario para poder trabajar desde casa en caso de no contar con el equipo de trabajo habitual. El repositorio se puede acceder a través de la siguiente URL: <https://github.com/FernandoCapellan/RAM_DMA_BusMaster>

2 – La segunda tarea realizada fue crear un selector de direcciones. Este módulo se iba a encargar en un principio de distribuir la dirección a la RAM o al DMA según se quisiera leer o escribir un dato o escribir sobre el banco de registros. Sin embargo, dicho módulo fue descartado, y el bus de direcciones fue conectado a ambos módulos a la vez. Mediante el uso de instrucciones condicionales se lee el dato presente en dicho bus cuando sea necesario.

3 – A continuación se diseñó la memoria RAM que almacenará los datos. En un principio se pensó en utilizar un array de vectores, pero esta idea tuvo que ser descartada cuando se intentó sintetizar el módulo y la herramienta tuvo que acaparar todos los recursos del ordenador de trabajo para mapear los componentes necesarios en la FPGA que se había seleccionado. La siguiente idea fue utilizar la herramienta Xilinx CoreGenerator para producir un módulo de memoria Block RAM con los puertos y mecanismos internos necesarios y optimizados para su funcionamiento en este proyecto.

Cuando se intentó establecer la capacidad de la memoria en 2 MiB, o 2.097.152 posiciones de 1 Byte (8 bits) cada una, la herramienta advirtió de que la FPGA objetivo no disponía de los recursos suficientes para alojarla. En un principio se seleccionó la Virtex-6 XC6VLX240T, por ser un modelo ya conocido para mí, al haber investigado sus propiedades para mi TFG y TFM. Como se puede observar en el datasheet que proporciona Xilinx:

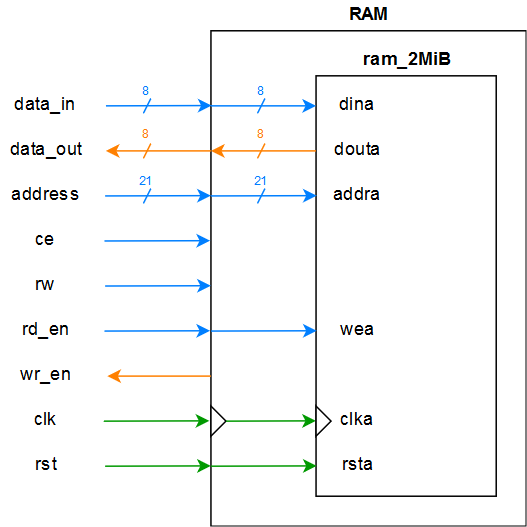
<https://www.xilinx.com/support/documentation/data_sheets/ds150.pdf>

En la segunda página aparece la cantidad de memoria Block RAM de la que dispone cada modelo de placa. El modelo previamente seleccionado cuenta con 14.976 Kb, lo cual se traduce en 14.976 Kb / 8 = 1.872 KB, 1.872.000 B / 1.024 = 1.828,125 KiB y 1.828,125 KiB / 1.024 = 1,785 MiB. Dado que una memoria de 2 MiB no cabe dentro de la FPGA sería necesario utilizar una memoria externa o una FPGA de mayor capacidad. Se optó por esta opción dado que se va a realizar una simulación del sistema, no a volcar el diseño sobre la placa. Se seleccionó un modelo superior, la Virtex-6 XC6VLX550T, la cual cuenta con 22.752 Kb de Block RAM, lo que se traduce en 2,712 MiB, lo que permite introducir la memoria en ella, aun ocupando gran cantidad del área disponible.

La memoria creada es de tipo Single Port, cuenta con un puerto de direcciones de 21 bits, puertos de entrada y salida de datos de 8 bits, un puerto de habilitación de escritura y puerto de reset.

El siguiente paso en esta etapa consistió en crear un contenedor para la memoria. En un principio el contenedor utilizaba un puerto de datos bidireccional a través del cual se leían los datos provenientes del exterior y se servían los datos presentes en la memoria. Debido a problemas para adaptar la memoria con el resto del diseño, la interfaz se modificó para utilizar puertos separados de entrada y salida, lo cual también simplificó la lógica de control de la memoria. Además, se consideró necesario introducir los puertos rd\_en y wr\_en, cuya función es indicar al receptor de sus señales que el dato proveniente del bus de datos es el correcto y puede ser usado. Dado que la memoria RAM tiene una latencia de aproximadamente un ciclo de reloj, la señal de notificación que produce se retarda un ciclo.

En la siguiente figura se muestra la interfaz del módulo de memoria:

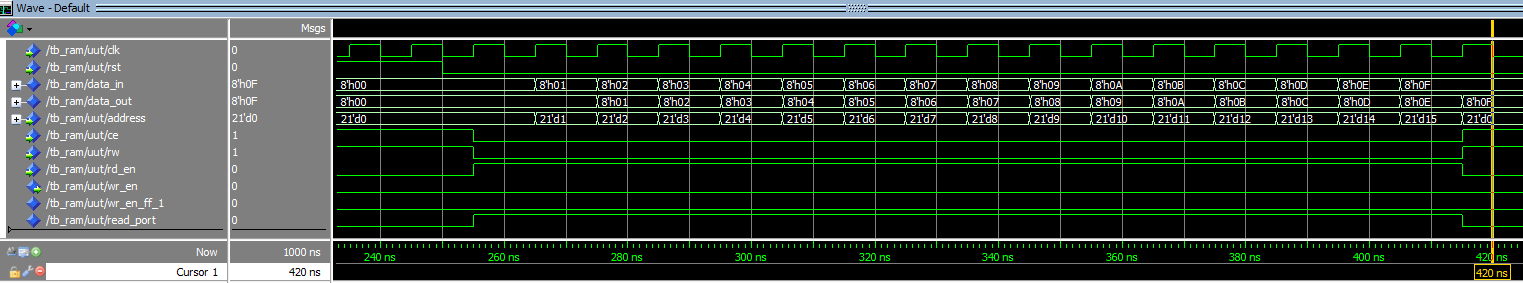


Donde RAM es el módulo contenedor y ram\_2MiB es el módulo obtenido con CoreGenerator.

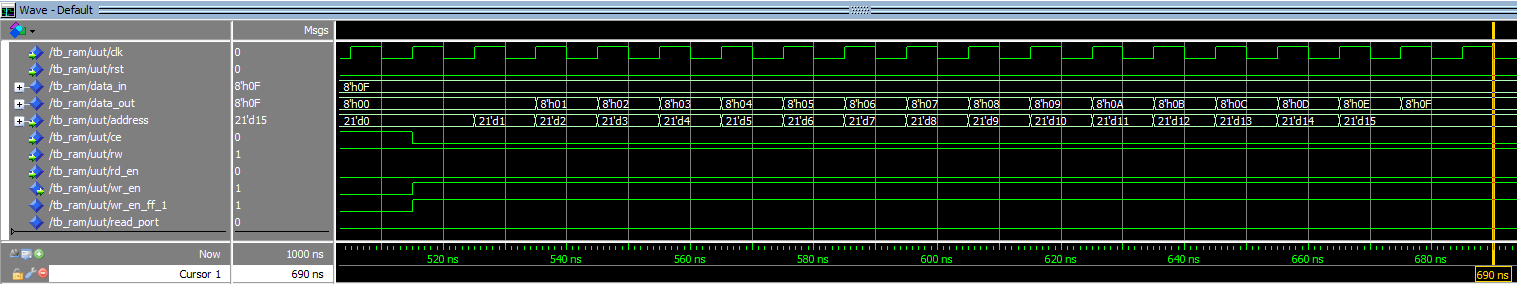
Se ha implementado un testbench para probar su correcto funcionamiento, tanto en escritura de datos como lectura de la memoria:

Para la escritura se introdujo una secuencia de datos en direcciones consecutivas. En la lectura se leyó el contenido de esas mismas posiciones. Se puede apreciar que los datos son devueltos un ciclo de reloj más tarde de lo que se proporciona la dirección de acceso.

Escritura



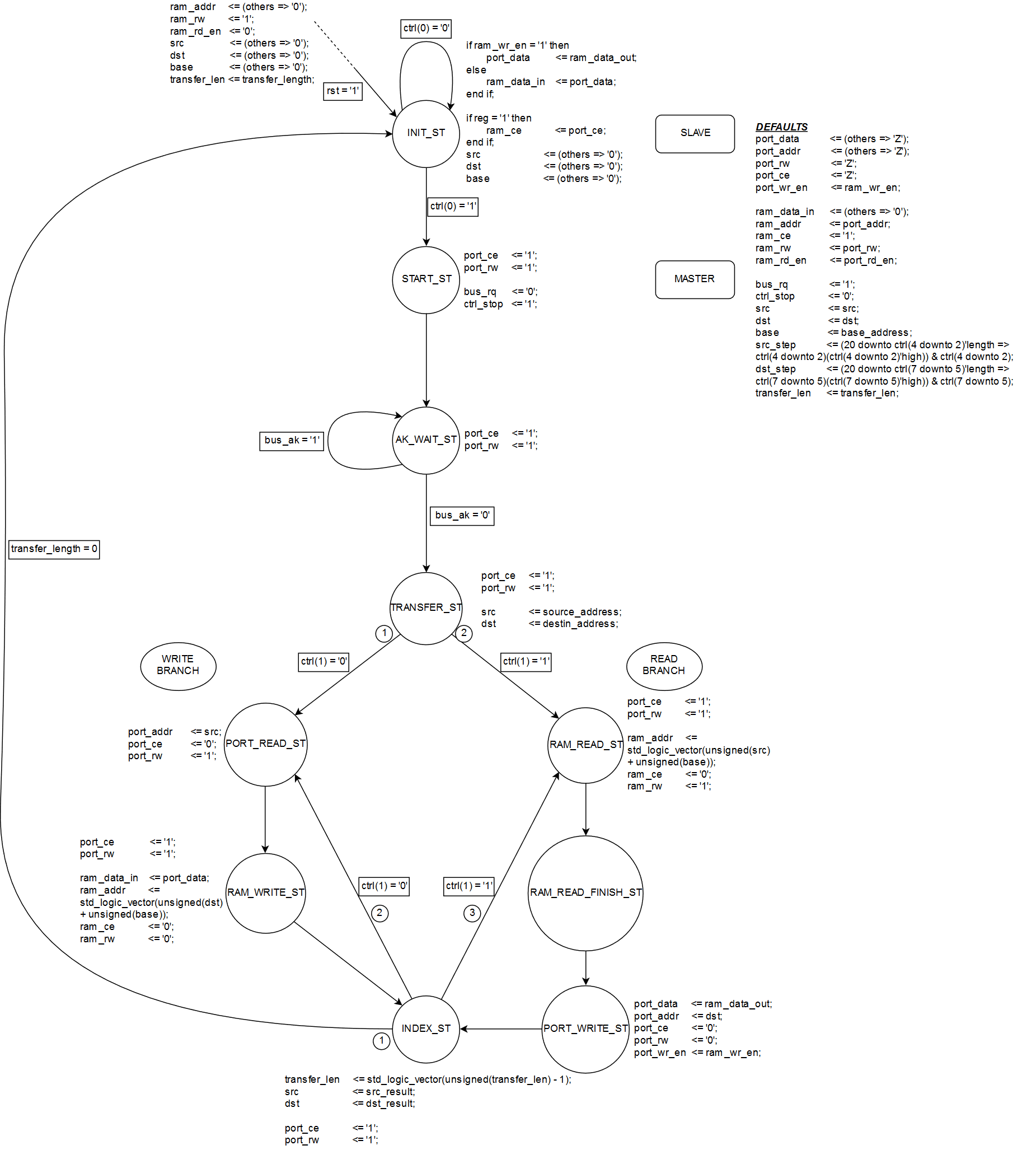
Lectura:



4 – El siguiente paso fue empezar el diseño del DMA. Se creó el esqueleto del módulo, con los registros especificados y la lógica para escribir sobre ellos. Tras analizar el enunciado del problema se hizo evidente la necesidad de que el DMA fuese controlado por una máquina de estados para automatizar la transferencia y recepción de datos, por lo que se procedió a su diseño en papel. Tras cuatro diseños descartados se procedió a usar la página <https://www.draw.io/> para crear los diagramas de forma más clara y poder realizar cambios sobre ellos más rápidamente. Tras diferentes iteraciones sobre el esquema de estados, la FSM se puede resumir de la siguiente forma:

|  |  |
| --- | --- |
| **Estado** | **Función** |
| INIT\_ST | Modo esclavo del DMA. El DMA puede recibir peticiones de escritura de datos que le manden o de lectura de los datos que contenga su memoria. Puede además recibir valores a escribir en sus registros, cuando bit 0 del registro de control o bit START sea activado, procederá a tomar control del bus y realizar una transferencia. |
| START\_ST | El DMA realiza una petición del bus para tomar el rol de maestro. Además, desactiva el bit START del registro CTRL. |
| AK\_WAIT\_ST | El DMA queda a la espera de confirmación de que puede usar el bus como maestro. |
| TRANSFER\_ST | Una vez recibida la confirmación de maestro de bus, se cargan las direcciones fuente y destino en sendas señales para poderlas incrementar o decrementar con los valores de autoincremento y transmitirlas a los módulos de almacenamiento de datos |
| PORT\_READ\_ST | En este estado se manda la dirección fuente al exterior para que el módulo conectado al nuestro envíe los datos necesarios. Es decir, realizar una lectura sobre él |
| RAM\_WRITE\_ST | Se escribe el dato entrante en la dirección destino que se manda a la memoria RAM. Se proporcionan también las señales de control necesarias para realizar la escritura, CE y RW |
| RAM\_READ\_ST | La dirección fuente es enviada a la RAM para que produzca el dato que será enviado al exterior. |
| RAM\_READ\_FINISH\_ST | Dado que la memoria tiene una latencia de 1 ciclo de reloj + 0.1 ns, es necesario esperar un ciclo más para que las señales de control se sincronicen a la salida del módulo. |
| PORT\_WRITE\_ST | Se envía la dirección de destino al exterior. El dato y las señales de control llegan sincronizadas. |
| INDEX\_ST | Decrementa el contador de transferencias y autoincrementa o autodecrementa las direcciones fuente y destino con los valores asignados en el registro CTRL. Si el contador de transferencias llega a 0, la transferencia se da por terminada y la FSM vuelve al estado de inicio, y el DMA a modo esclavo. |

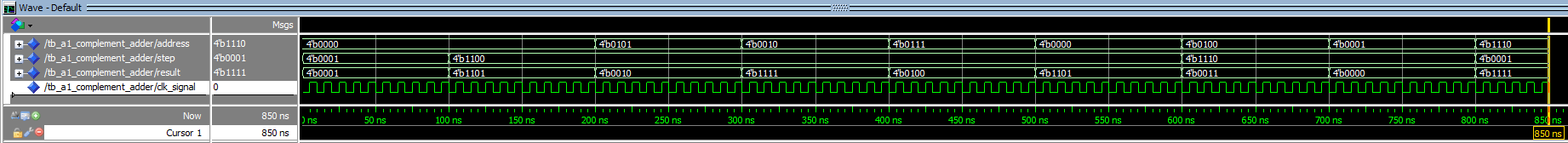
El diagrama de transiciones entre estados queda representado en la siguiente figura, junto a las condiciones de cada transición, el orden en que estas se evalúan en caso de haber diferentes ramas para un estado y las instrucciones que se ejecutan en cada estado:



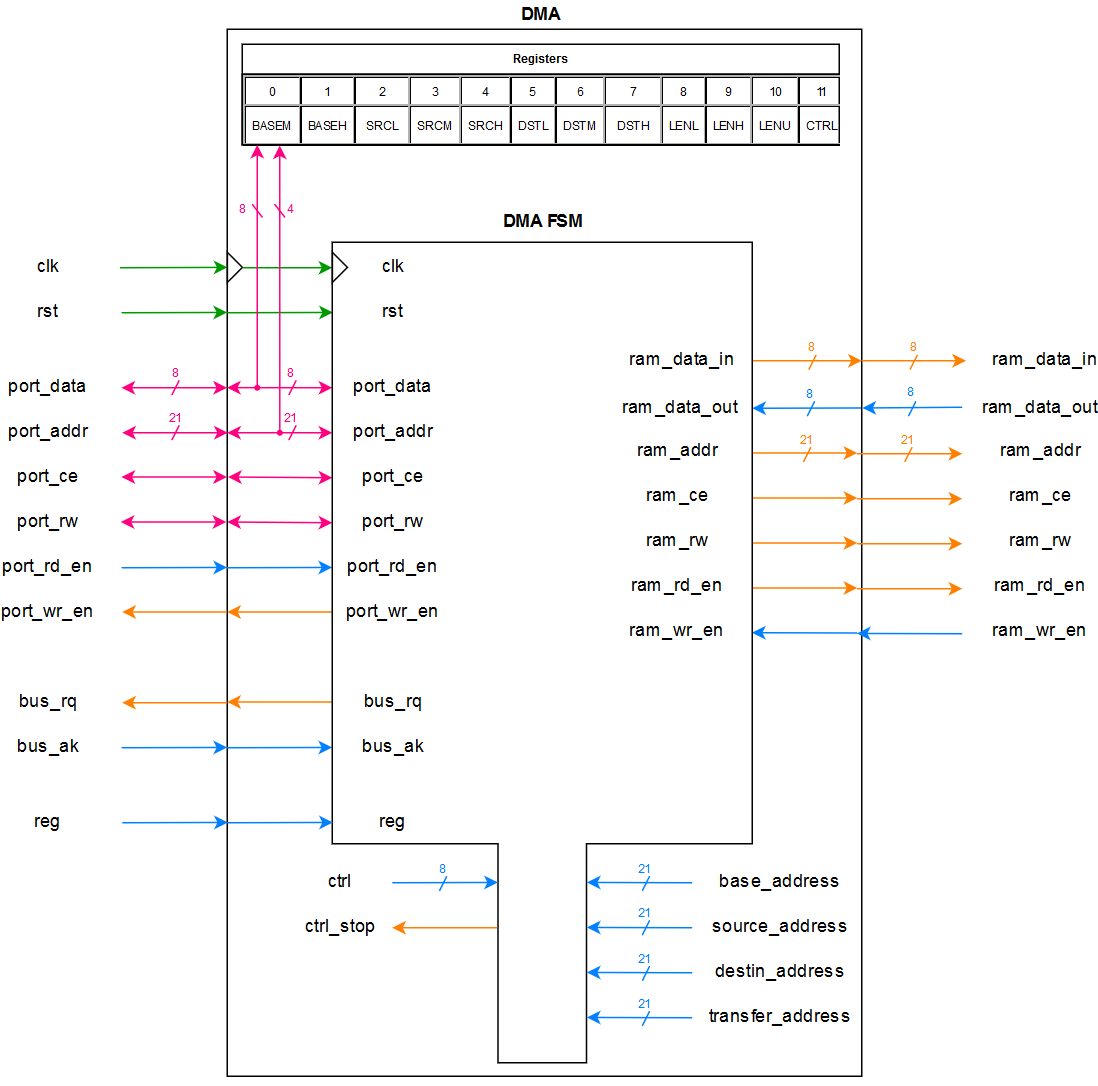
Realizando pruebas, en una fase todavía temprana del desarrollo, se comprobó que los datos llegaban antes que las señales de control. Esto era debido a que el bus de datos estaba conectado directamente a la memoria, mientras que dichas señales eran generadas por la FSM y por tanto dependían del reloj. Por ello se tomó la decisión de incluir este bus en la máquina de estados.

En un principio se implementó la máquina de estados en tres procesos diferentes: uno que realizaba la transición al siguiente estado dependiendo únicamente del reloj, en cada ciclo se realizaba una transición. El siguiente proceso evaluaba las condiciones de transición y establecía cual era el siguiente estado al que saltar. El último proceso se encargaba de ejecutar las instrucciones asociadas al estado en el que se encontrara la máquina. Este diseño se hizo funcionar en la simulación, pero la herramienta de síntesis proporcionaba una amplia lista de advertencias, entre las que se encontraban algunos como detección de múltiples señales que debían ser colocadas en listas de sensibilidad o latches en señales y buses. Para solucionar estos warnings, la FSM se reconstruyó en un único proceso, encargado de todas las funciones anteriormente mencionadas.

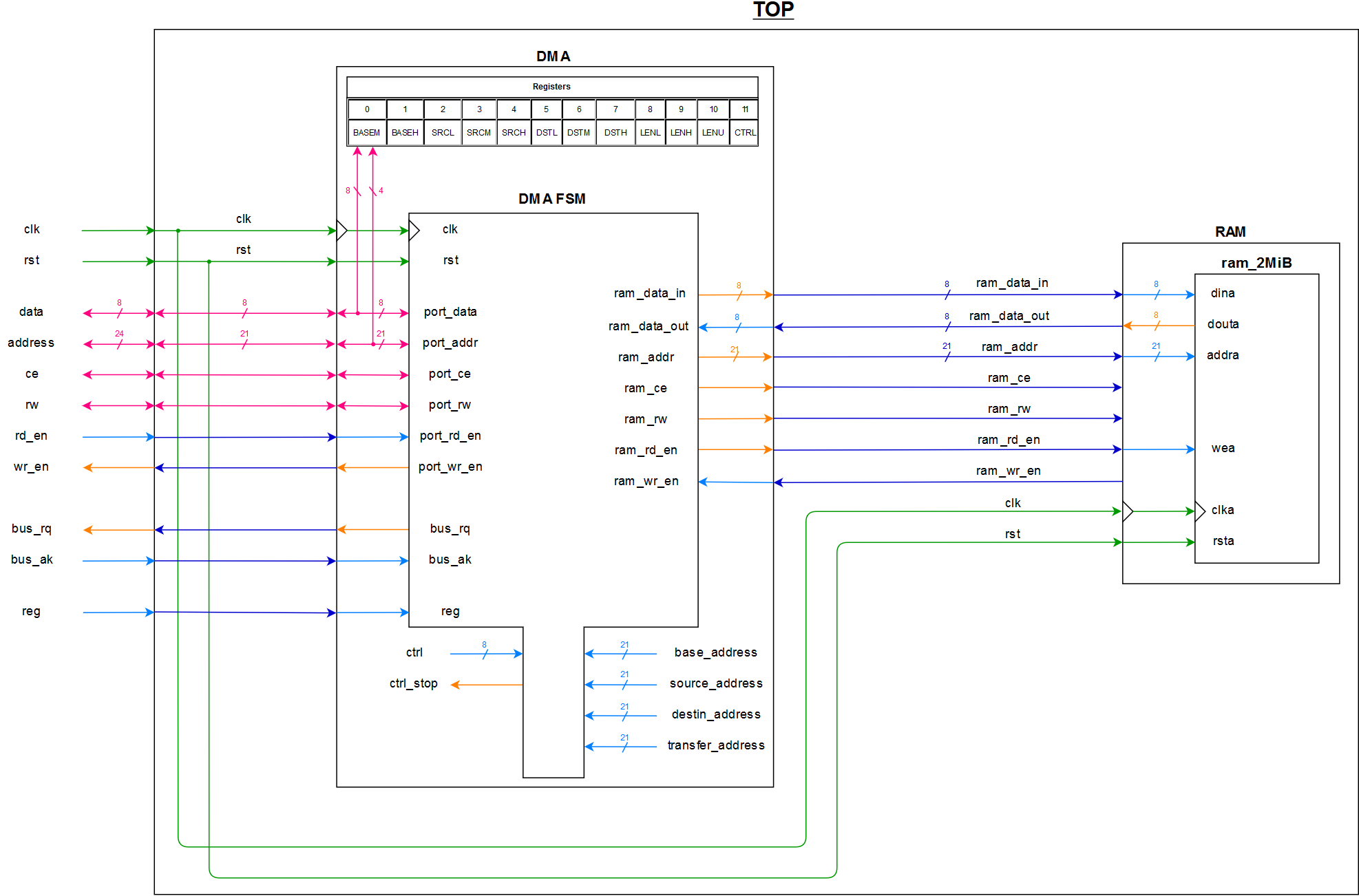
Para incrementar y decrementar las direcciones se indicaba que se usase un fragmento del vector de control como valor de autoincremento, además de interpretarlo como valor en complemento A1. Dado que el tipo Signed de VHDL funciona como complemento A2, se implementó un módulo sumador combinacional que ejecutase dicho autoincremento. Cuando el valor de la dirección se decrementa por debajo del valor mínimo representable o se incrementa sobre el máximo, el valor se desborda. Esto permite usar la RAM como una memoria circular. Por ejemplo, con 4 bits para el vector de direcciones podemos direccionar 16 posiciones, de la 0 a la 15. La posición previa a la posición 0000 o 0 es 1111 o 15. En la siguiente imagen se puede ver el correcto funcionamiento del módulo mediante un simple testbench:



El módulo del DMA queda pues como un contenedor del banco de registros, la lógica de control de éstos y la máquina de estados finita. En la siguiente imagen podemos ver su interfaz y contenido:



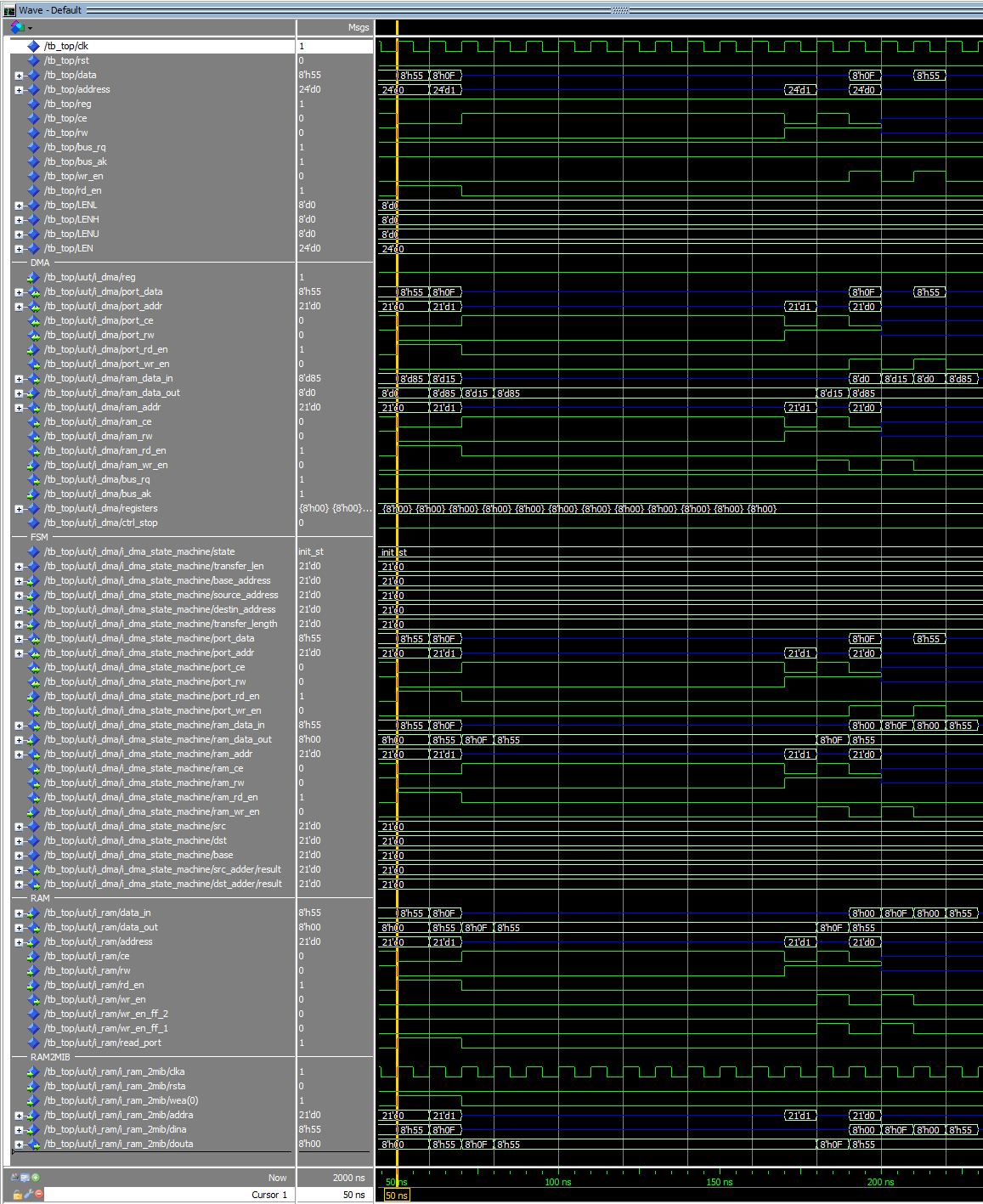
5 – Por último, el módulo principal Top contiene ambos módulos, la memoria RAM y el DMA. Este módulo no dispone de lógica de control, ya que de esa tarea se encarga la máquina de estados que hemos explicado antes. En este módulo se interconectan estos dos módulos y se conectan con los puertos del sistema. Este es el diagrama de bloques conjunto resultante:



Aunque se han realizado testbenches de cada módulo por separado, es decir, la máquina de estados finita, el DMA, la RAM y el sumador en complemento A1, el testbench del módulo principal aúna todos los resultados que se pueden extraer de ellos. En las siguientes páginas se explican los cronogramas resultantes. Las imágenes requieren ampliación para poder apreciar el texto, por lo que ocuparán gran parte de las páginas y quedarán muchos huecos vacíos:

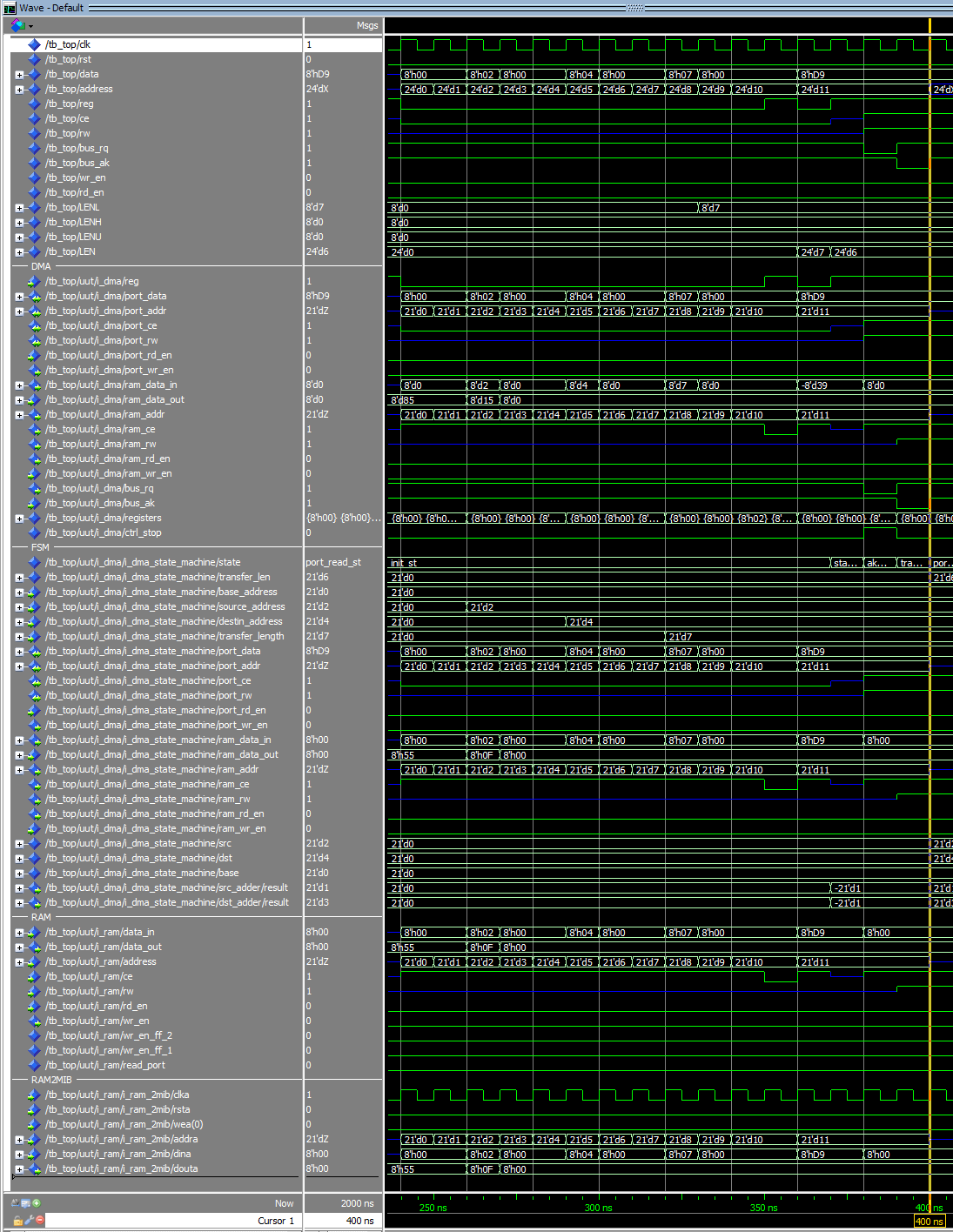
En la figura siguiente podemos ver al módulo funcionando como esclavo, ya que está a la espera de instrucciones en el estado inicial. Se realizan dos escrituras y dos lecturas. En las escrituras se proporciona el dato, la dirección, se activan (a nivel bajo) las señales CE y RW y se transmite la señal de enable para indicar que los datos son correctos y deben ser leídos. Estas señales son transmitidas en el mismo ciclo desde el Top, pasando por el DMA, la FSM, la RAM hasta la memoria, que se encarga de almacenarlos.

Para realizar la lectura, se debe proporcionar una dirección, CE = 0 y RW = 1. Todo ello llega en el mismo ciclo a la RAM, que busca los datos en la memoria. Un ciclo de reloj y 0.1 ns después, la memoria pone el dato en el bus, mientras que la señal de habilitación wr\_en ha sido generada tan solo un ciclo después. Cuando los datos llegan a la FSM, se sincronizan en el mismo ciclo y flanco de reloj. Dos ciclos de reloj después de que el dato haya sido solicitado, será proporcionado por el puerto de datos del módulo principal.

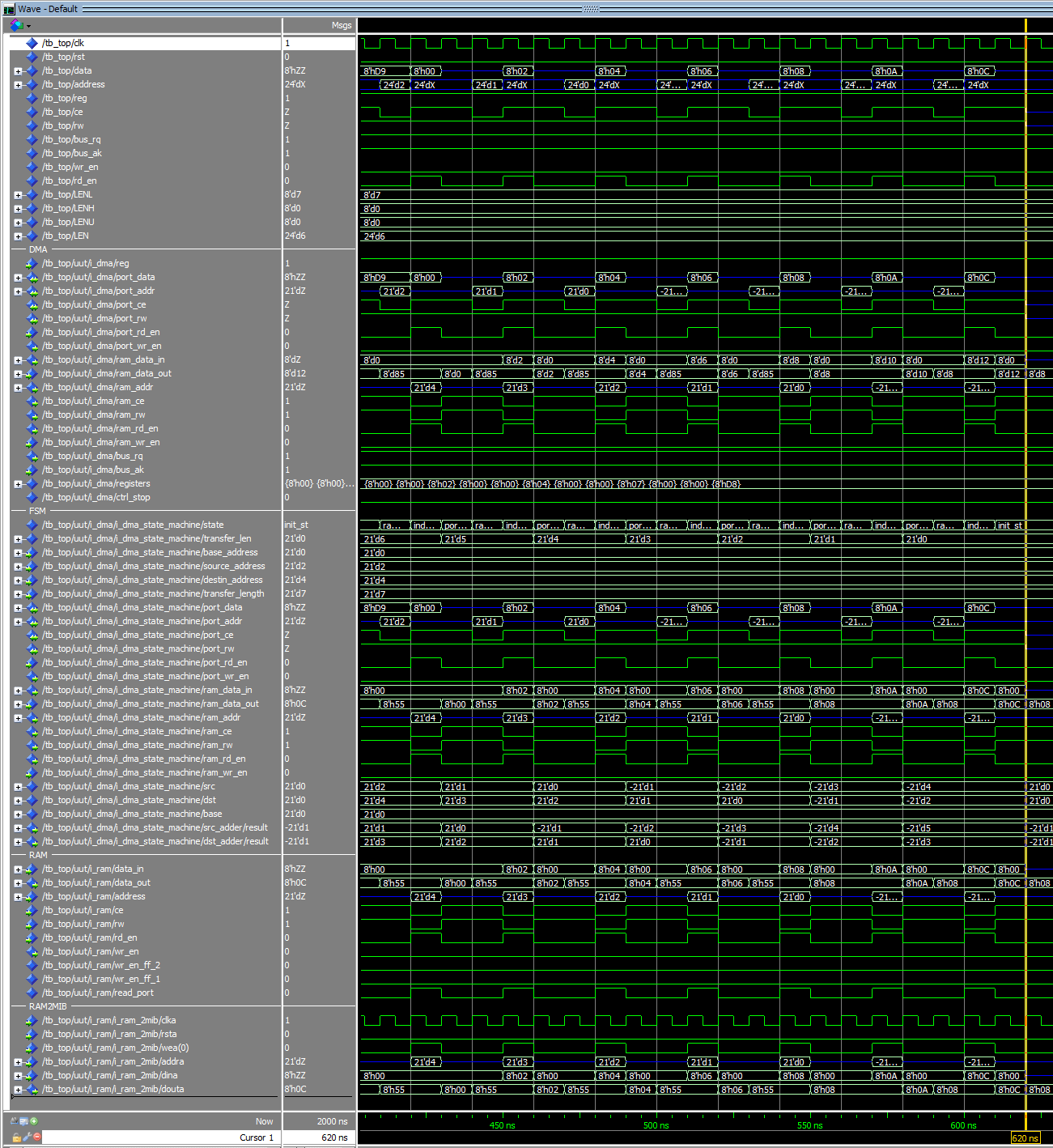


La siguiente figura muestra la escritura de los registros del DMA con los siguientes parámetros:

* Dirección base de mapeo de memoria RAM = 0
* Primera dirección fuente que se pedirá por el puerto al exterior = 2
* Primera dirección destino para escribir sobre la RAM = 4
* Longitud de la transferencia = 7
* Registro de control: SRC autoincremento: -1; DST autoincremento: -1; Modo: Lectura de datos del exterior; Start: Activado = “11011001”

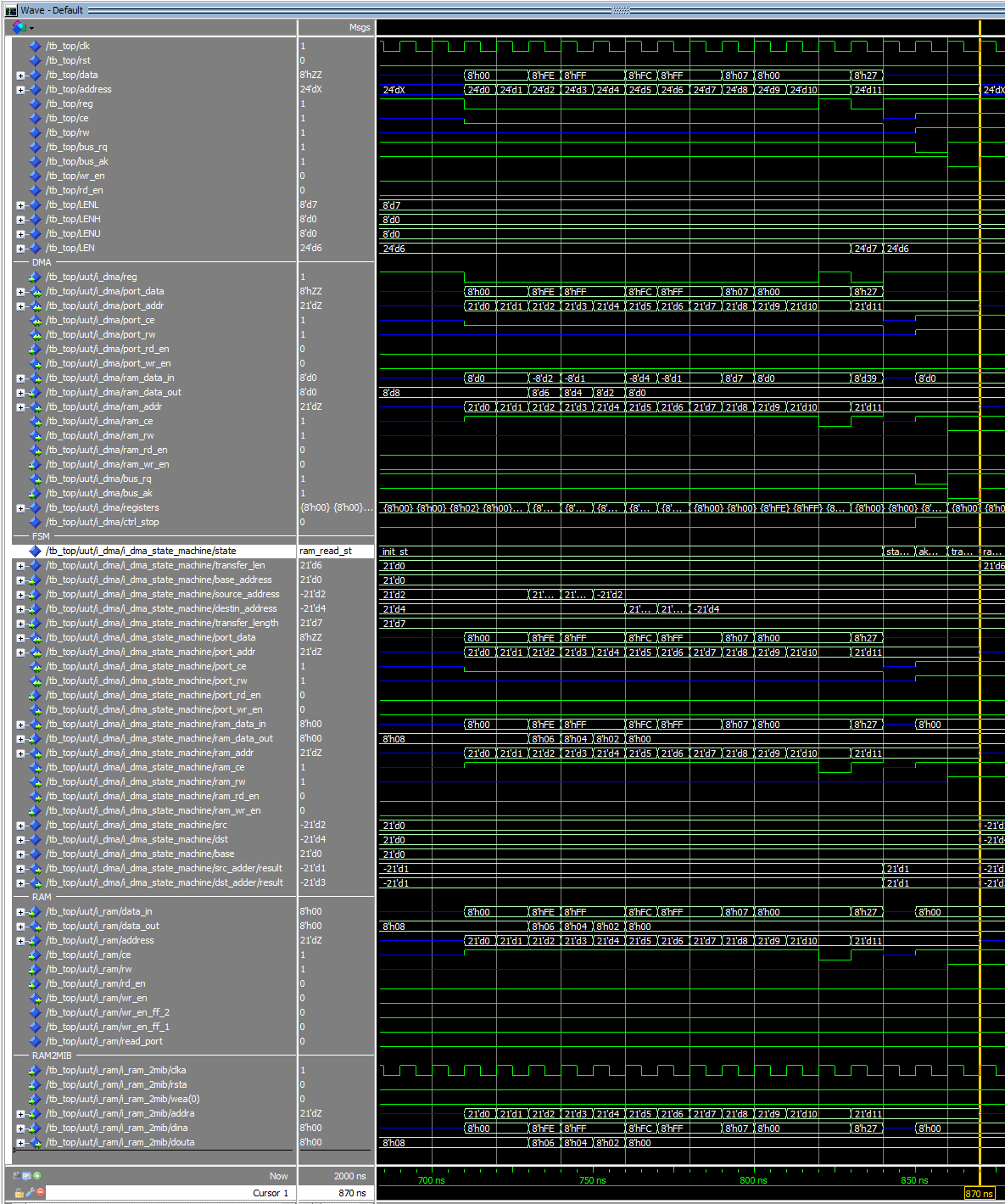


Tras estas escrituras, la FSM cambia al estado START\_ST para convertirse en maestro del bus y tomar el control de las transferencias. Manda una petición bus\_rq, que es respondida con un bus\_ak, se cargan las señales correspondientes y da comienzo la transferencia. En cada vuelta a la rama de lectura de la máquina de estados se proporciona una dirección al exterior, en el siguiente ciclo de reloj simulamos recibir el dato y se genera la dirección de la RAM. Esta almacena los datos que recibe en las posiciones que se le indican. Cuando el contador de transferencias llega a 0, se retorna al estado inicial.

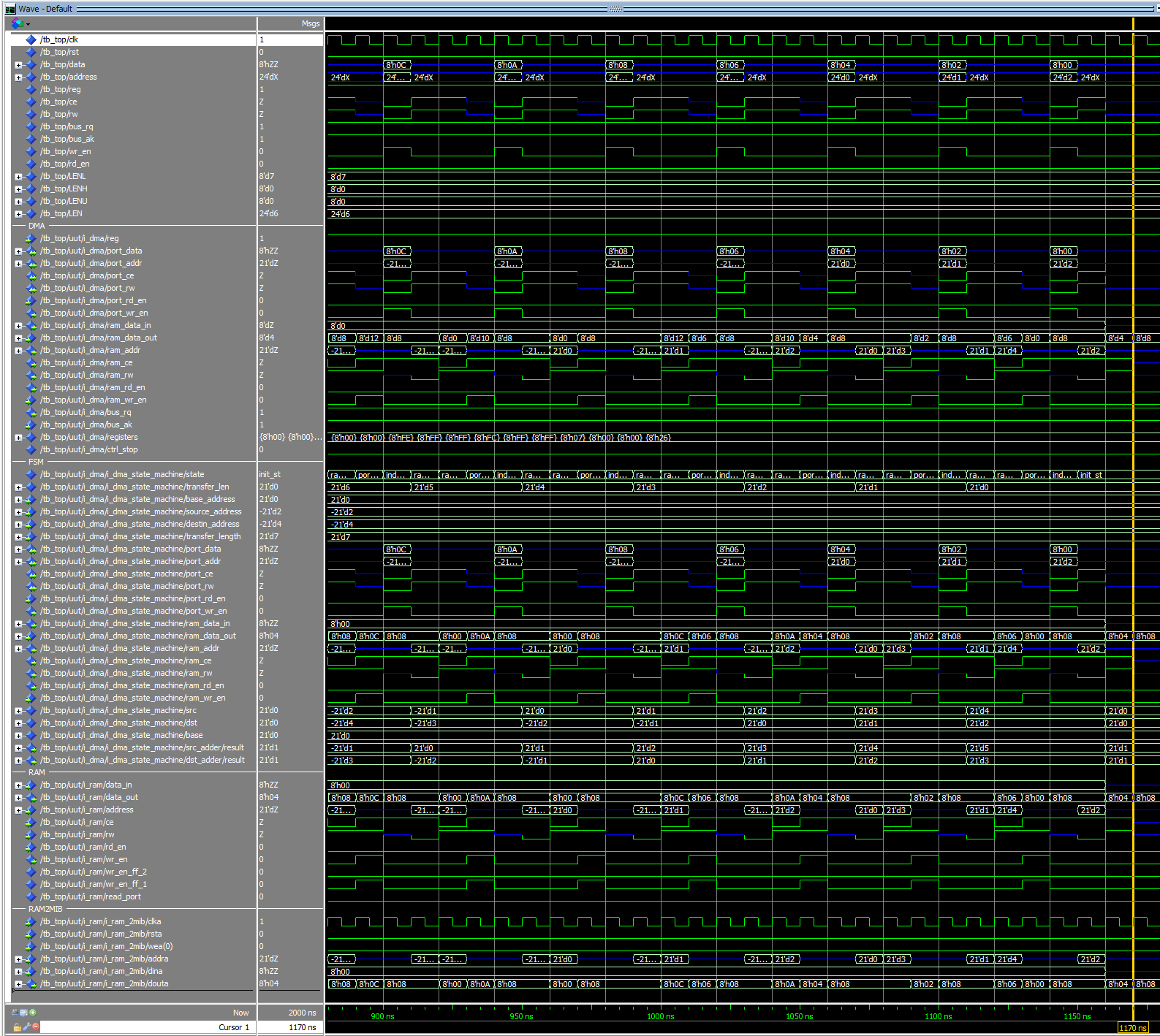


Tras unos ciclos de espera, se simula la lectura de dichos datos. En este caso, los valores escritos en los registros son los siguientes:

* Dirección base de mapeo de memoria RAM = 0
* Primera dirección fuente para la RAM = dirección en la que ha finalizado la escritura de datos en el proceso anterior = -2
* Primera dirección destino para escribir sobre el módulo externo = dirección en la que ha concluido la lectura de datos en el proceso anterior = -4
* Longitud de la transferencia = 7
* Registro de control: SRC autoincremento: 1; DST autoincremento: 1; Modo: Lectura de datos de la RAM; Start: Activado = “00100111”



El último cronograma muestra un comportamiento similar al que se ha explicado cuando se mostraba la lectura de la RAM con el dispositivo en modo esclavo: los datos se proporcionan con dos ciclos de reloj de latencia. Por lo demás el funcionamiento es similar al de la lectura de datos del puerto:



En la imagen se puede ver que el DMA manda una dirección a la memoria RAM, esta proporciona el dato 1 ciclo de reloj y 0.1 ns después, y la FSM sincroniza la transmisión del dato con la señal de confirmación del dato, para que lleguen al exterior al mismo tiempo.

Analizando el informe de síntesis podemos extraer las siguientes estadísticas:

Módulos inferidos por la herramienta de síntesis:

|  |  |  |
| --- | --- | --- |
| ***HDL Synthesis*** | | |
| Adders/Subtractors | 21-bit adder | 1 |
| 21-bit subtractor | 1 |
| 22-bit adder | 4 |
| Registers | 1-bit register | 12 |
| 21-bit register | 8 |
| 8-bit register | 14 |
| Comparators | 21-bit comparator greater | 3 |
| 4-bit comparator greater | 1 |
| Multiplexers | 1-bit 2-to-1 multiplexer | 5 |
| 21-bit 2-to-1 multiplexer | 12 |
| 22-bit 2-to-1 multiplexer | 2 |
| 8-bit 2-to-1 multiplexer | 4 |
| Tristates | 1-bit tristate buffer | 31 |
| FSMs | | 1 |
| Xors | 1-bit xor2 | 2 |

Tras realizar optimizaciones, la síntesis avanzada infiere los siguientes componentes:

|  |  |  |
| --- | --- | --- |
| ***Advanced HDL Synthesis*** | | |
| Adders/Subtractors | 21-bit adder | 1 |
| 21-bit subtractor | 1 |
| 22-bit adder | 2 |
| Accumulators | 21-bit up loadable accumulator | 2 |
| Registers | Flip-Flops | 241 |
| Comparators | 21-bit comparator greater | 3 |
| 4-bit comparator greater | 1 |
| Multiplexers | 1-bit 2-to-1 multiplexer | 55 |
| 21-bit 2-to-1 multiplexer | 8 |
| 22-bit 2-to-1 multiplexer | 2 |
| 8-bit 2-to-1 multiplexer | 3 |
| FSMs | | 1 |
| Xors | 1-bit xor2 | 2 |

Por último, la síntesis de bajo nivel simplifica el resultado de la siguiente manera

|  |  |  |
| --- | --- | --- |
| ***Low level Synthesis*** | | |
| Registers | Flip-Flops | 245 |

Podemos ver también cuanto ocupa la implementación de la memoria RAM sobre la FPGA:

Specific Feature Utilization:

Number of Block RAM/FIFO: 512 out of 632 81%

Number using Block RAM only: 512

Y un resumen del informe de tiempos de la implementación:

Periodo mínimo: 4.970ns (Frecuencia máxima: 201.207MHz)

Periodo mínimo de llegada de señales de entrada antes del reloj: 2.147ns

Periodo máximo necesario para señales de salida tras el reloj: 1.392ns

Máximo retardo combinacional: No se ha encontrado camino crítico

El retardo de 4.970ns consiste en el camino desde la memoria RAM hasta el Puerto de salida de datos del módulo principal, que en total son 5 niveles de lógica:

3.694ns lógica (74.3%), 1.276ns rutado (25.7%)