به نام خدا

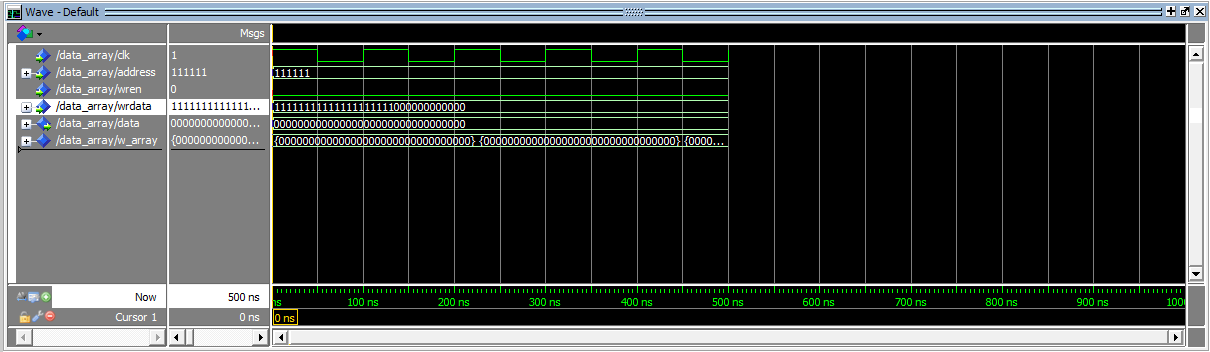
**گزارش پروژه معماری کامپیوتر**

محمدمهدی آقاجانی

**9331056**

**آرایه داده ای (data array )**

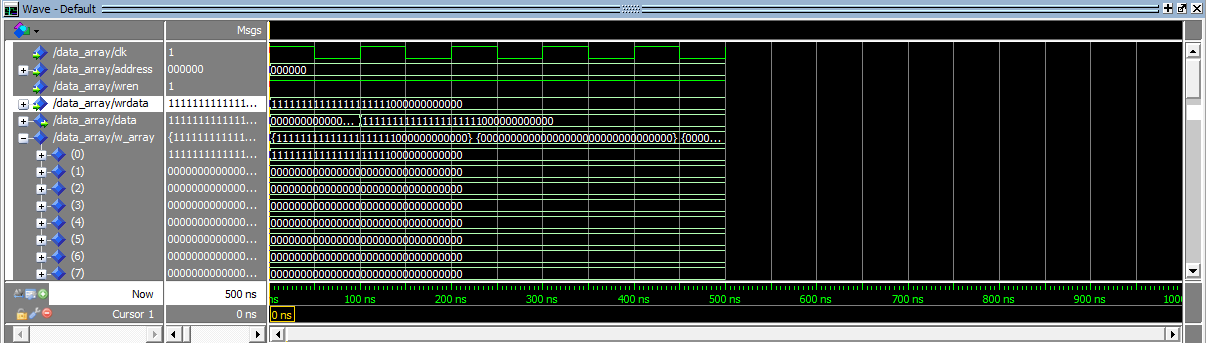
آرایه ای از std logic vector ها ست که در واقع 64 عنصر دارد. این ماژول حساس به لبه بالارونده کلاک است و اگر سیگنال wren فعال باشد اقدام به نوشتن می کند در غیر این صورت از همین آرایه ، داده مربوطه را می خواند و در خروجی قرار میدهد.

شکل موج خروجی وقتی که wren = 0 :

در این حالت داده در آدرس 111111 که در واقع 00000000000000000000000000000000 است در خروجی قرار گرفته است :

شکل موج خروجی وقتی wren = 1 است :

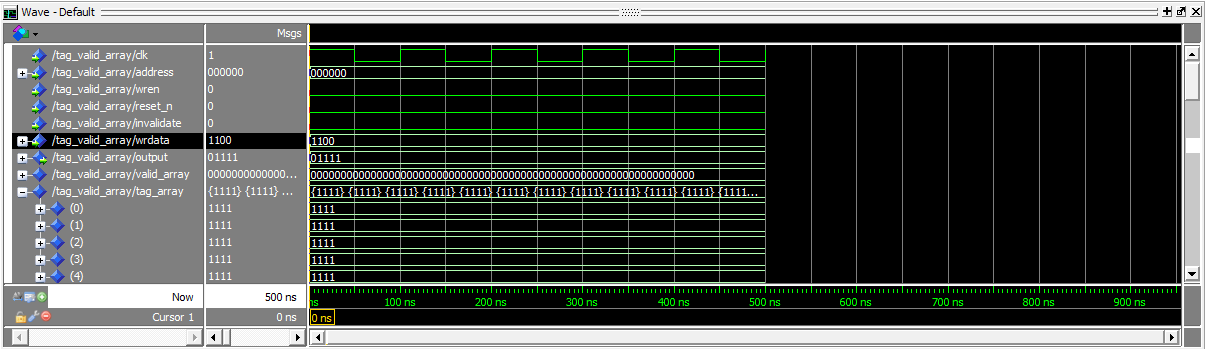
در این حالت داده wrdata در خانه آدرس که 000000 است قرار گرفته همچنین به خاطر این جایگزینی موفق داده در خروجی نیز قرار گرفته است :



**Tag valid array** :

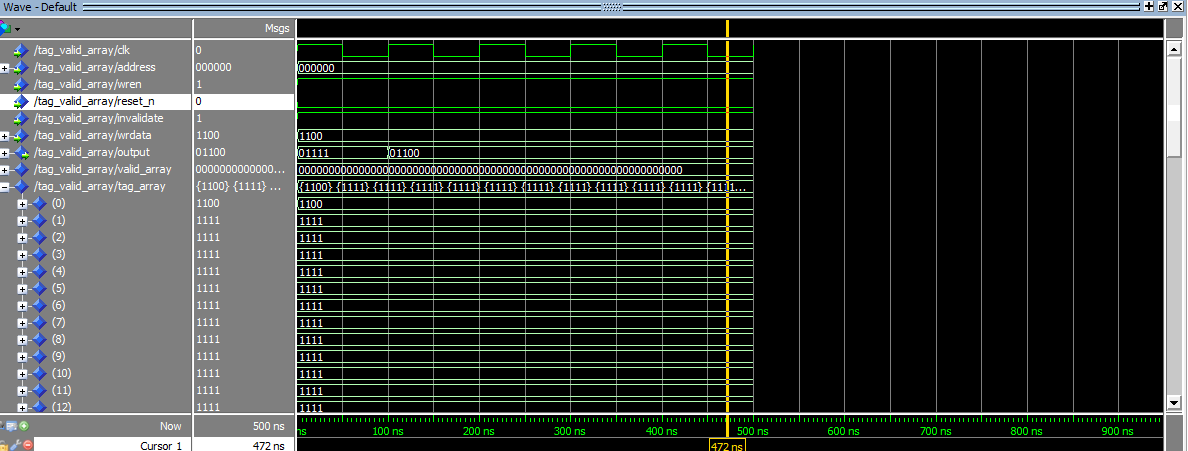
نحوه ساخت این ماژول مانند آرایه داده ای می باشد و حساس به لبه بالارونده کلاک است توجه کنید که در ابتدا به صورت پیش فرض تمامی داده های tag ها برابر 1111 تمامی داده های valid برابر صفر قرار داده شده است

در زیر شکل موج به ازای wren = 0 , reset = 0 را مشاهده می کنید :

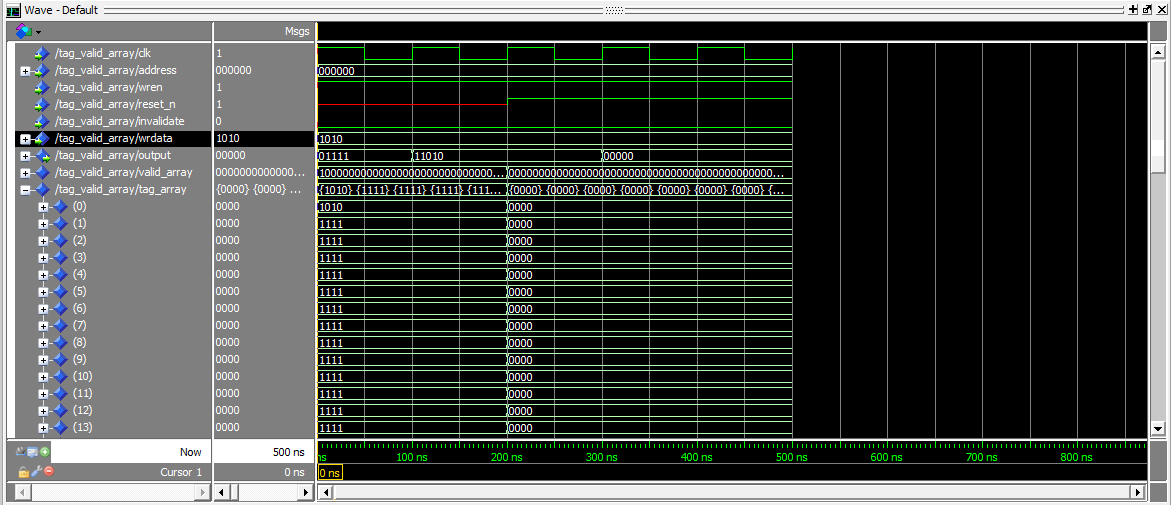
 در این حالت بدلیل اینکه سیگنال invalidate برابر صفر است valid خانه مورد نظر تغییری نکرده و برابر همان مقدار قبلی ست به همین دلیل در خروجی سیگنال 01111 را مشاهده میکنیم

در این حالت شکل موج را به ازای wren = 1 , reset = 0 مشاهده می کنید

دقت کنید که به این علت valid مربوط به آدرس داده شده صفر شده است که invalidate برابر 1 در نظر گرفته شده و الا باید valid مربوط به آدرس داده شده برابر یک قرار میگرفت



در این حالت reset برابر یک قرار داده شده :

دقت کنید که wren برابر یک است و invalidate برابر صفر است به همین خاطر در ابتدا خروجی برابر داده نوشته شده در آرایه قرار میگیرد و سپس بعد از اینکه ریست برابر یک قرار داده می شود تمامی آرایه برابر صفر قرار میگیرد

**منطق مطابقت ( miss-hit logic ) :**

این مدار چون حساس به کلاک نیست باید در سطح گیت پیاده شود که از گیت xnor برای مقایسه استفاده شده است و در نهایت با valid مقایسه می شود تا نتیجه را مشخص کند.

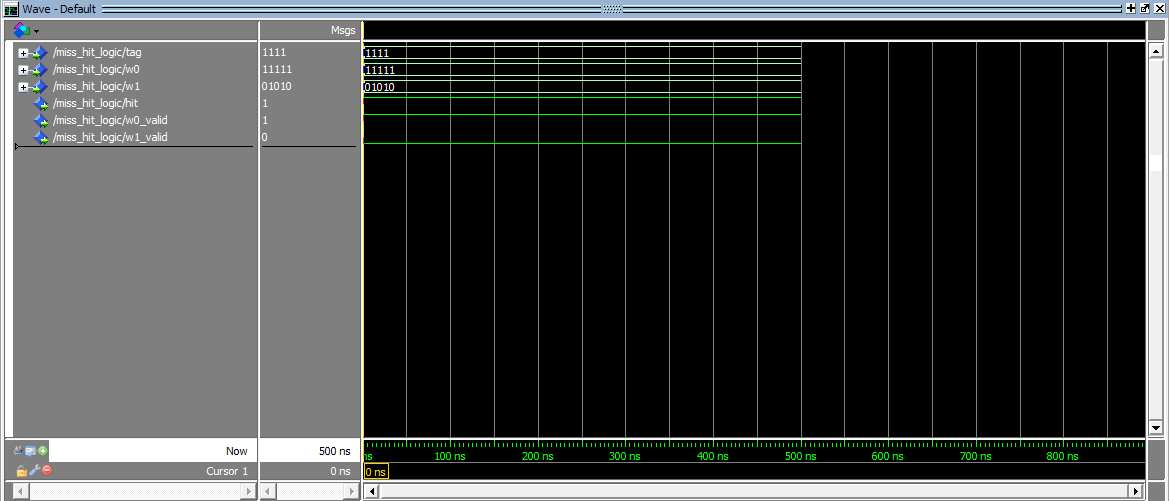
در زیر جدول درستی و جدول کارنو را می کشیم . توجه کنید که برای هر بیت ورودی یک جدول می کشیم و بعد آن ها را ادغام میکنیم. این کار بدین دلیل است که تعداد ورودی های بیتی خیلی زیاد می باشد و جدول کارنو دچار پیچیدگی می شود.

|  |  |  |  |
| --- | --- | --- | --- |
| Equal\_w(i) | valid | W(i) | Tag(i) |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| valid | 00 | 01 | 11 | 10 |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |

با توجه به جدول بالا خواهیم داشت :

با توجه به بالا و با توجه به این نکته که باید برای اینکه یک مطابقت رخ دهد تمامی equal\_w(i) ها برابر یک باشند پس خواهیم داشت :

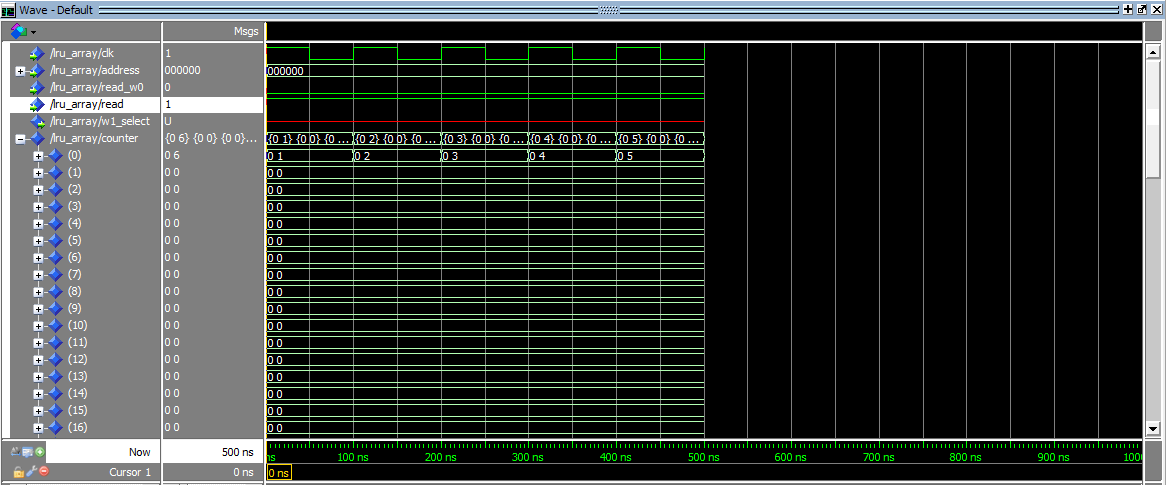
در این حالت شکل موج را در حالتی که hit رخ داده است مشاهده می کنید:

در این حالت miss رخ داده است و علت این است که w0 دارای بیت valid مساوی صفر می باشد و دیگری نیز اصلا تگ برابر با تگ ورودی ندارد:

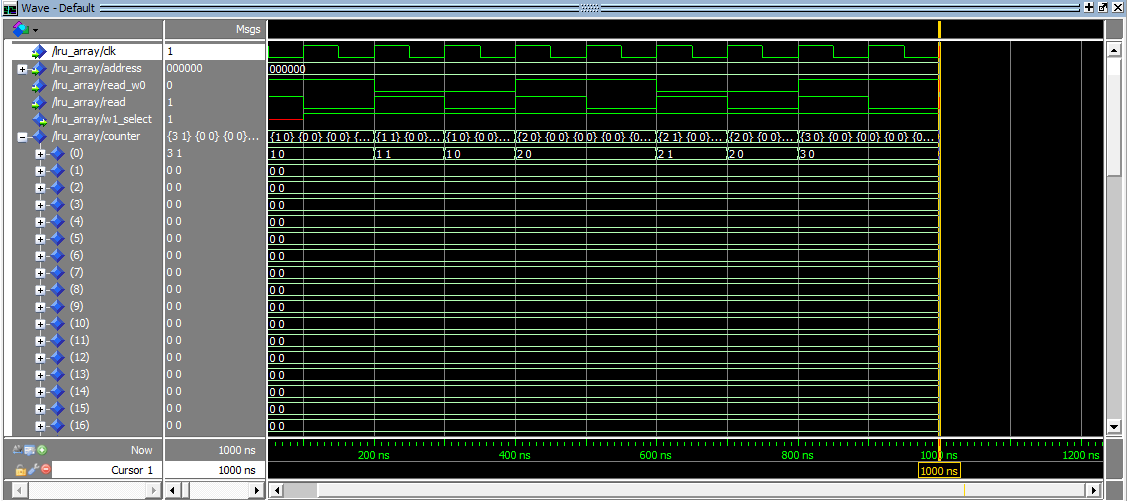


**سیاست جایگزینی ( lru array ) :**

این ماژول حساس به کلاک است و دارای سه سیگنال کنترلی read و read\_w0 و read\_w1است که هر وقت read برابر یک باشد به این معناست که cache در حال خواندن است و شمارنده به ازای آدرس ورودی یکی زیاد می شود که این افزایش برمبنای سیگنال کنترلی دیگری به نام read\_w0 و read\_w1 است که اگر اولی یک باشد شمارنده مربوط به w0 زیاد میشود و اگر دومی یک باشد شمارنده مربوط به w1 یکی زیاد می شود

در زیر شکل موج مربوط به حالت read = 1 , read\_w0 = 0 را مشاهده میکنید :

در شکل زیر شکل موج در حالت کلی تر را برای این ماژول مشاهده میکنید :



**نحوه کارکرد controller :**

این ماژول دارای یک سری خروجی هایی ست که تماما سیگنال های کنترلی بقیه ماژول ها هستند که عبارتند از : invalidate , ram\_write ,ram\_read ,read\_cache,wren برای کارکرد آن یک FSM طراحی کردیم که به صورت زیر است

Invalidate : i , ram\_write : rw , ram\_read : rr, read\_cache : rc , wren : wr

read\_request = 1

**Start**

i=0 , rw = 0 , rr = 0 ,

rc = 0 , wr = 0

Ram\_ready = 1

Is\_hit = 0

**read**

i=0 , rw = 0 , rr = 1 ,

rc = 0 , wr = 0

**miss**

i=0 , rw = 0 , rr = 1 ,

rc = 0 , wr = 0

**Read\_from\_ram**

i=0 , rw = 0 , rr = 0 ,

rc = 0 , wr = 1

Cache\_ready = 1

Cache\_ready = 1

Is\_hit = 1

Ram\_reaedy = 1

Cache\_ready = 1

write\_request = 1

**Write\_to\_ram**

i=1 , rw = 1 , rr = 0 ,

rc = 0 , wr = 0

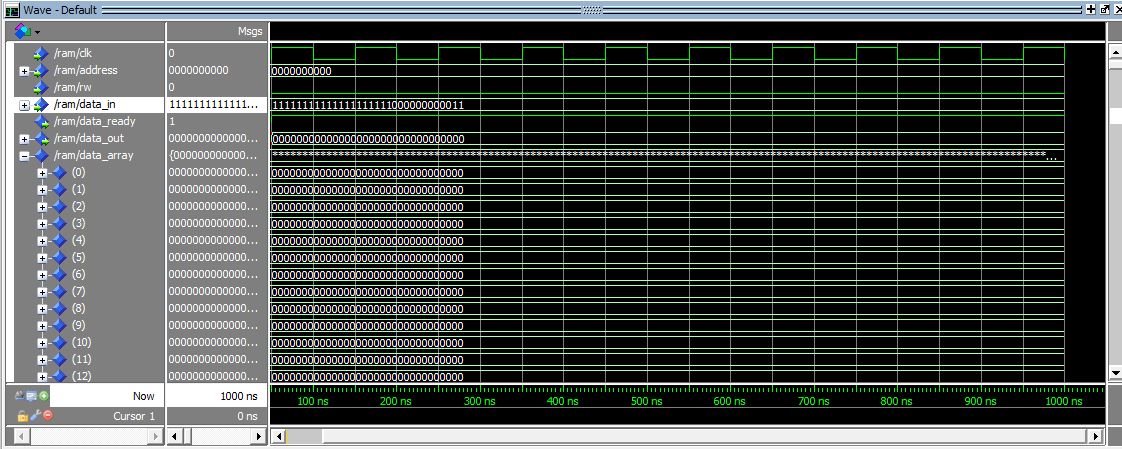
**hit**

i=0 , rw = 0 , rr = 0 ,

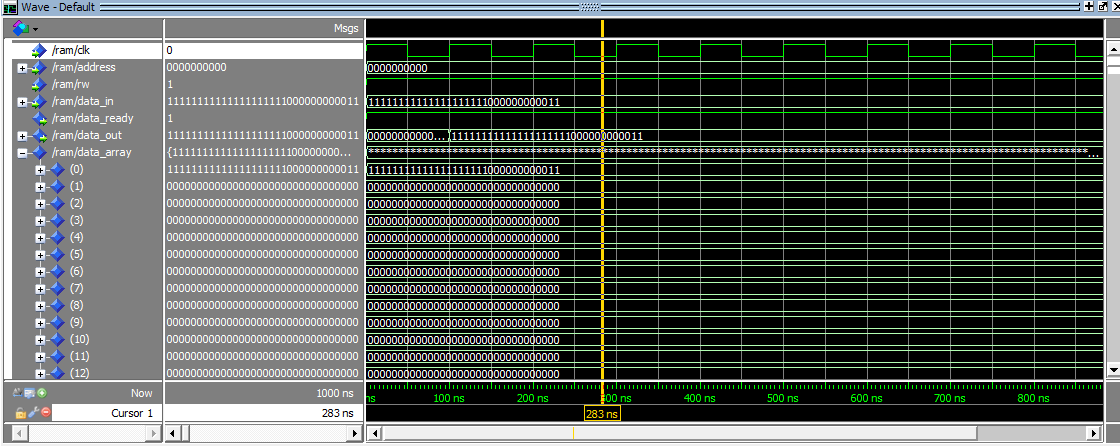
rc = 1 , wr = 0

البته توجه کنید که در پیاده سازی FSM بالا برای صرفه جویی در زمان و کم کردن استفاده از زمان کلاک برخی state ها در هم ادغام شده و سیگنال ها بعضا با گذاشتن شروطی در یک state تایین می گردند که البته علت این کار این است که برخی از سیگنال های ورودی این ماژول مانند is\_hit چون توسط مدار های ترکیبی تعیین میگردند خیلی زود مشخص می شوند و ربطی به کلاک ندارند

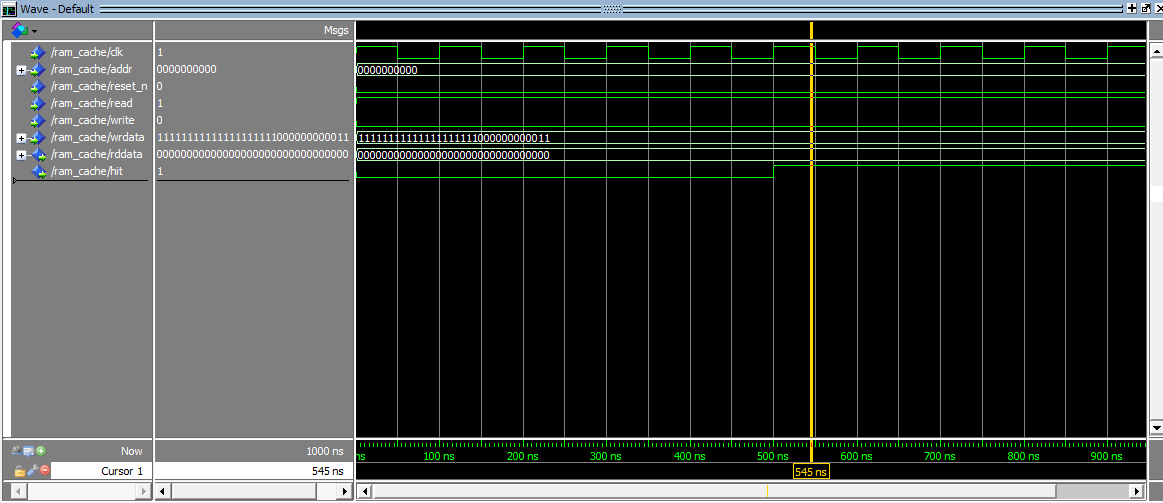
**کارکرد ram :**

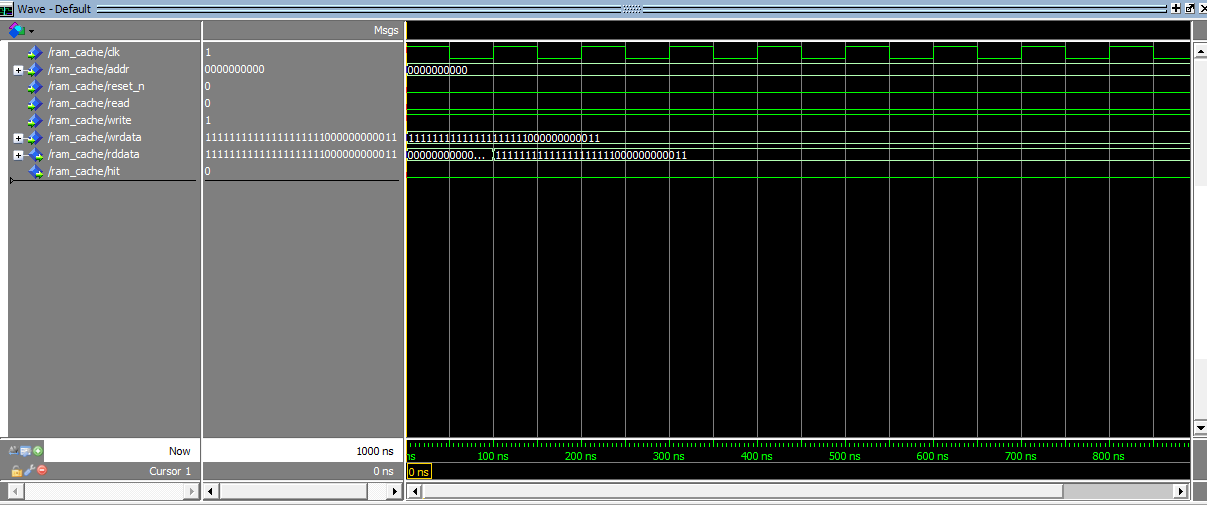
این ماژول همانند data array عمل می کند ولی تنها سیگنال کنترلی آن همان rw است که در زیر شکل موج را به ازای حالت rw = 0 مشاهده میکنید :

در زیر شکل موج به ازای حالت rw = 1 می بینید که باید دیتا در آدرس مربوطه نوشته شود :

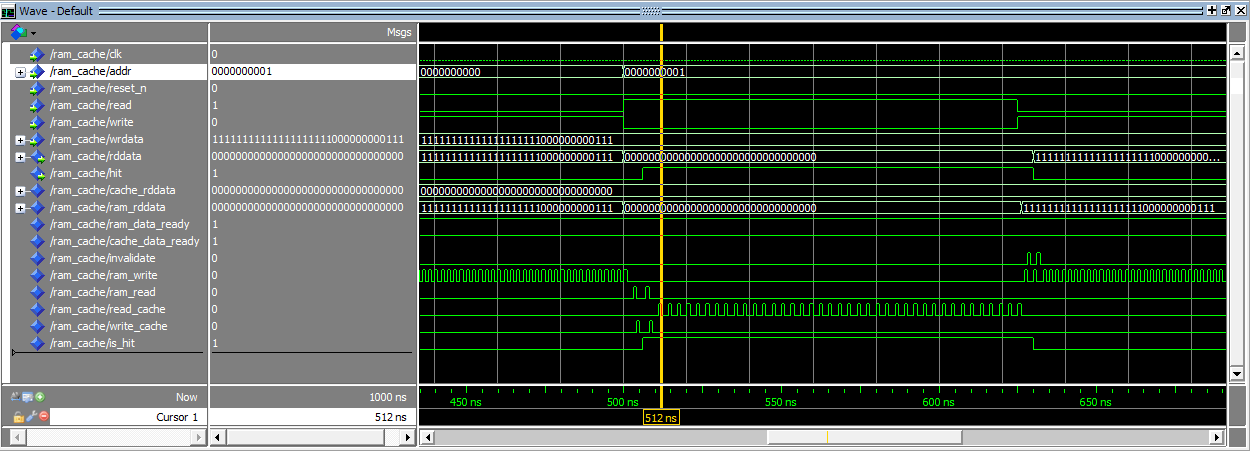


**آزمایش ماژول کلی ram\_cache :**

Addr = 0000000000 , reset\_n = 0 ,read = 1 , write = 0 , wrdata = 11111111111111111111000000000011

Addr = 0000000000 , reset\_n = 0 ,read = 0 , write = 1 , wrdata = 11111111111111111111000000000011

شکل موج حالت کلی ماژول ram\_cache :



**نحوه اتصال ماژول ها به یکدیگر**

**W0\_data\_array**

clk

address W0\_data\_array\_rddata

W0\_wren

wrdata

نحوه اتصال ماژول های cache :

**Miss\_hit\_logic**

Tag hit

W0\_tag w0\_valid

W1\_tag w1\_valid

**W1\_data\_array**

clk

address W1\_data\_array\_rddata

W1\_wren

wrdata

**Lru**

Clk

Address

Read\_w0

Read\_w1 w1\_select

Read

Write

invalidate

**W1\_tag\_valid\_array**

Clk

Address

W1\_wren w1\_tag\_valid\_output

Reset

Invalidate

wrdata

**W0\_tag\_valid\_array**

Clk

Address

W0\_wren w0\_tag\_valid\_output

Reset

Invalidate

wrdata