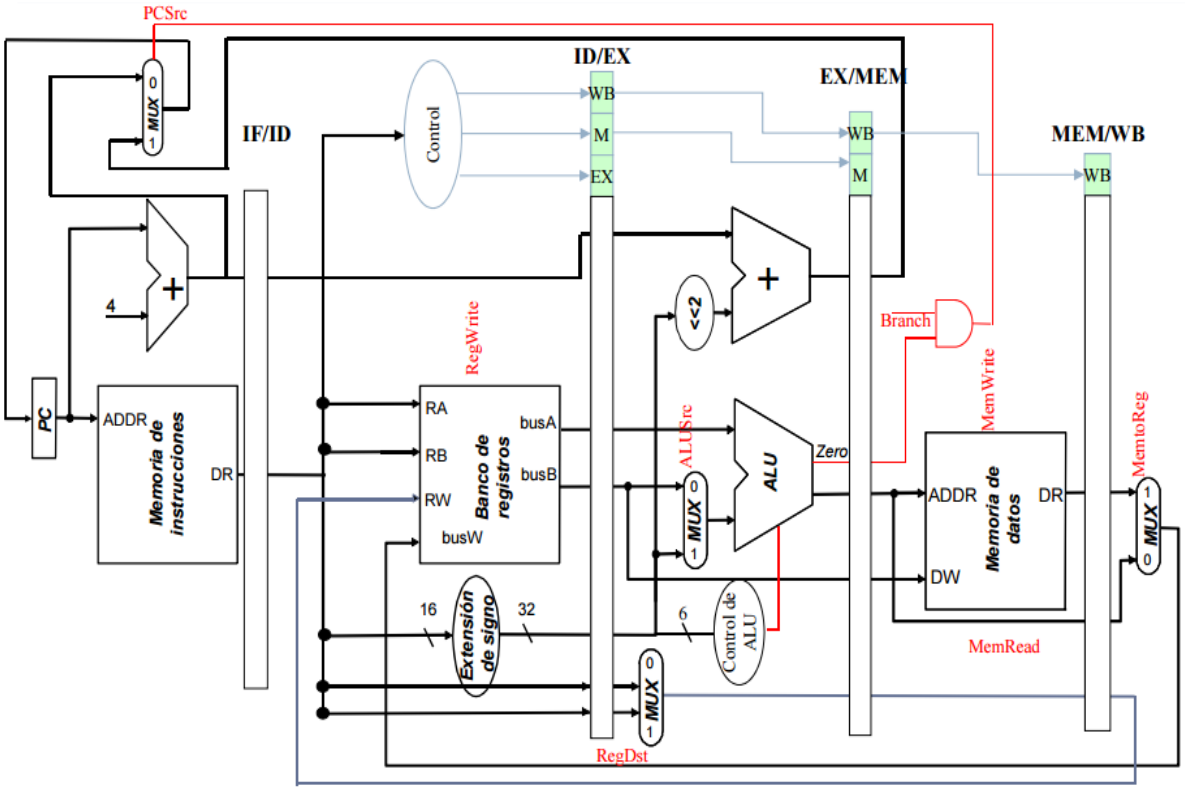
Proyecto Optativo 2:

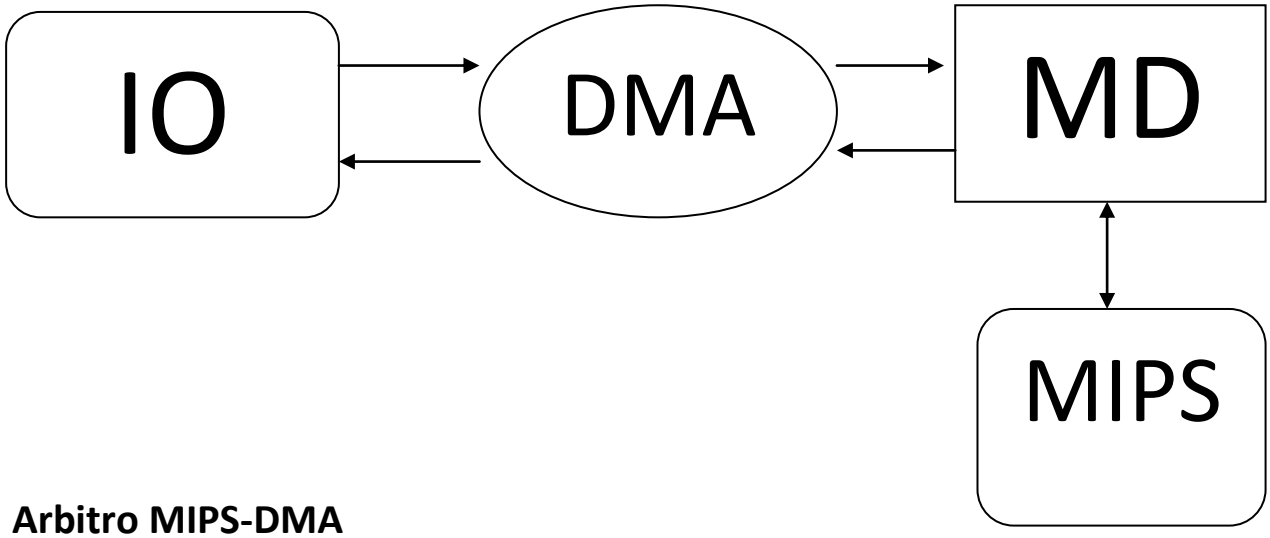


Bus compartido y DMA.

Miguel Allué Barón - 593599

Cristian Simón Moreno - 611487

Esquema del nuevo diseño de memoria con el bus:



Arbitro MIPS-DMA

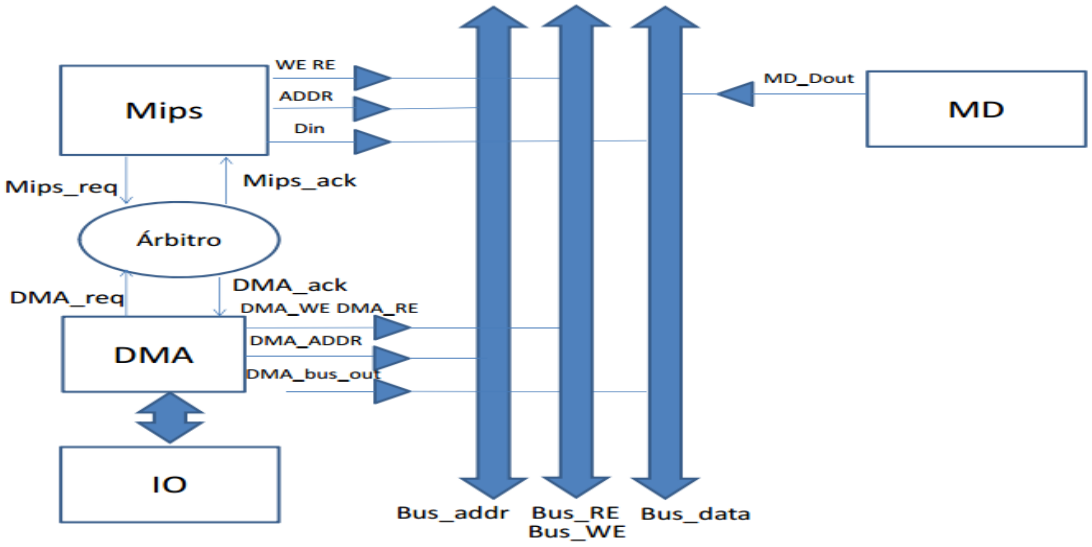
Cuando el Mips necesite usar el bus (WE o RE a 1) indicaremos que lo necesita, con Mips_req.

```
arbitro: process (WE, RE)
begin
    -- PETICION MIPS -> 1 cuando se quiera leer o escribir
    if (RE = '1' OR WE = '1') then
        Mips_req <= '1';
    else
        Mips_req <= '0';
    end if;
end process;

 Mips_ack <= '1' when Mips_req='1' else '0';--si el mips quiere usar la memoria siempre la encuentra disponible
 DMA_Ack <= '1' when (Mips_req='0' and DMA_req = '1') else '0';-- el DMA sólo puede usar la memoria si el Mips no la está usando
```

Cada vez que el Mips solicite el bus se le concederá, aunque lo esté usando el DMA.

Buses:



- **Bus ADDR:** Sera el encargado de las direcciones.

```
Bus_addr <=  ADDR when Mips_ack='1' else -- hay dos fuentes para las direcciones: Mips o DMA
             DMA_addr when DMA_Ack ='1' else
             x"00000000";
```

- **Bus data:** Sera el encargado de los datos; en el caso del Mips si tiene el bus entrara Din, si tiene el bus el DMA habrá que mirar si es lectura o escritura. Si es lectura -> entrara lo que salga de memoria; Si es escritura -> lo que venga del DMA (IO).

```
Bus_data <= Din when Mips_ack='1' else
            MD_Dout when (DMA_Ack ='1' and Bus_RE='1') else -- En lectura el DMA lee de MD
            DMA_bus_out when (DMA_Ack ='1' and Bus_WE='1') else -- En escritura escribe en MD
            x"00000000";
```

- **Bus RE/WE:** Son las señales de lectura y escritura del Mips y de la DMA, sacadas de comprobar el ack.

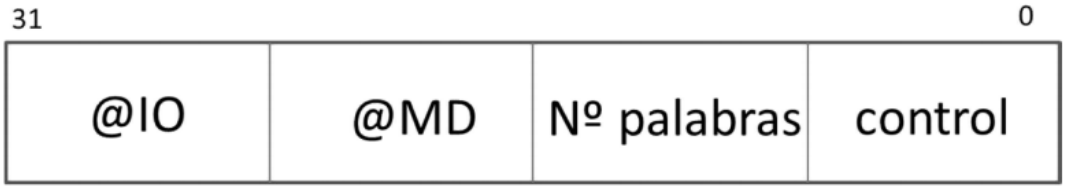
```
Bus_RE <=  RE when Mips_ack = '1' else
           DMA_MD_RE when DMA_Ack = '1' else
           '0'; -- hay dos fuentes para RE: Mips o DMA. por defecto a 0 para evitar operaciones no deseadas

Bus_WE <=  WE when Mips_ack = '1' else
           DMA_MD_WE when DMA_Ack = '1' else
           '0'; -- hay dos fuentes para WE: Mips o DMA. por defecto a 0 para evitar operaciones no deseadas
```

UC del DMA:

El DMA funcionara de tal forma que, el Mips le dará la orden de leer/escribir X palabras, una vez recibida esa orden a través del registro de control, hará las operaciones solicitadas (lectura de IO/MD, petición del bus y escritura en MD/IO), actualizando cada vez el contador para saber si quedan palabras por transferir (fin).

Registro de control:



- **@IO:** Dirección de la IO a partir de la cual se leerán/escribirán las palabras.
- **@MD:** Dirección de la MD a partir de la cual se leerán/escribirán las palabras.
- **Nº de palabras:** Es el numero de palabras de 32 bits que se leerán/escribirán.
- **Control:** 8 bits de los cuales solo se usaran 3:
  - 7: Lo actualiza el DMA al terminar una transferencia.
  - 1: Indica si es lectura o escritura (0 MD-IO, 1 IO-MD).
  - 0: Indica que hay que realizar la transferencia.

Para el desarrollo de la unidad de control de la UC DMA hemos obtenido **8 estados**:

1. **INI:** se comprobara si empezar esta a 1 y si fin esta a 0, es decir, que se puede empezar. Comprobaremos la operación a realizar, MD-IO o IO-MD.

```
-- ESTADO INICIAL: reseteamos contador y miramos empezar y fin; comprobamos L_E
if (state = INI) then
    if(empezar= '1' AND fin= '0') then
        if (L_E= '0') then
            --MD-IO
            next_state <= ESC_BUS;
        else
            --IO-MD
            next_state <= LEC_SYNC;
        end if;
    else
        next_state <= INI;
    end if;
```

2. **LEC_SYNC:** Estado para la sincronización del DMA con el IO, para ello se pondrá DMA_sync a 1, es decir que el DMA quiere hacer algo, activaremos el registro de datos de DMA con load_data, pondremos las señales para leer y comprobaremos si IO a terminado. Si ha terminado pasaremos al siguiente estado, si no ha terminado se volverá a ejecutar este estado.

```
--SYNC LECTURA
elsif (state = LEC_SYNC) then
    -- indicamos que queremos hacer algo
    DMA_sync <= '1';
    -- activamos el reg
    load_data <= '1';
    -- Leemos de IO
    DMA_IO_WE <= '0';
    DMA_IO_RE <= '1';
    -- Si IO ha terminado
    if IO_sync= '1' then
        next_state <= LEC_SYNC_ESPERA;
    else
        next_state <= LEC_SYNC;
    end if;
```

3. **LEC_SYNC_ESPERA:** Bajaremos DMA_sync (hay que esperar un ciclo para asegurarnos que el dato se ha cargado en el registro). Comprobaremos si IO a terminado, una vez haya terminado pasaremos al siguiente estado, si no ha terminado volveremos a ejecutar este estado.

```
--ESTADO DE ESPERA A QUE IO BAJE
elsif (state = LEC_SYNC_ESPERA) then
    --Bajamos cuando ya tenemos cargado el dato en el registro
    DMA_sync <= '0';
    --IO ha terminado
    if(IO_sync = '0') then
        next_state <= LEC_BUS;
    else
        next_state <= LEC_SYNC_ESPERA;
    end if;
```

4. **LEC_BUS:** pediremos el bus y esperaremos a que nos llegue, una vez lo tenemos (DMA_ack = 1) activaremos las señales para escribir en memoria, contaremos y pasaremos al estado para comprobar si hemos terminado o hay que seguir escribiendo. Mientras no recibamos el ack se ejecutara este estado.

```
--PEDIR BUS LECTURA
elsif (state = LEC_BUS) then
    --Pedimos el bus
    DMA_req <= '1';
    --Miramos cuando nos lo conceden
    if(DMA_ack= '1') then
        -- Escribimos en MD
        DMA_MD_RE <= '0';
        DMA_MD_WE <= '1';
        count_enable <= '1';
        next_state <= FIN_LECTURA;
    else
        next_state <= LEC_BUS;
    end if;
```

5. **FIN_LECTURA:** Se comprobara fin, es decir, si hemos terminado con el numero de palabras, si hemos terminado, resetearemos el contador, activaremos el bit de que se ha terminado la transferencia y volveremos al estado inicial. Si no hemos terminado, se volverá al estado de sincronización (LEC_SYNC) para seguir operando con mas palabras.

```
--COMPROBACION FIN LECTURA
elsif (state = FIN_LECTURA) then
    if(fin = '1') then
        -- Total de palabras completado
        reset_count <= '1';
        update_done <= '1';
        next_state <= INI;
    else
        -- Volvemos a leer de IO
        next_state <= LEC_SYNC;
    end if;
```

6. **ESC_BUS:** pediremos el bus y esperaremos a que nos llegue (DMA_ack = 1), una vez lo tenemos activaremos el registro, las señales para leer de memoria y pasaremos al estado de sincronización. Hasta que no recibamos el ack seguiremos ejecutando este estado.

```
--PEDIR BUS ESCRITURA
elsif (state = ESC_BUS) then
    --Pedimos el bus
    DMA_req <= '1';
    --Miramos cuando nos lo conceden
    if(DMA_ack= '1') then
        -- activamos el registro
        load_data <= '1';
        -- Leemos de MD
        DMA_MD_RE <= '1';
        DMA_MD_WE <= '0';
        next_state <= ESC_SYNC;
    else
        next_state <= ESC_BUS;
    end if;
```

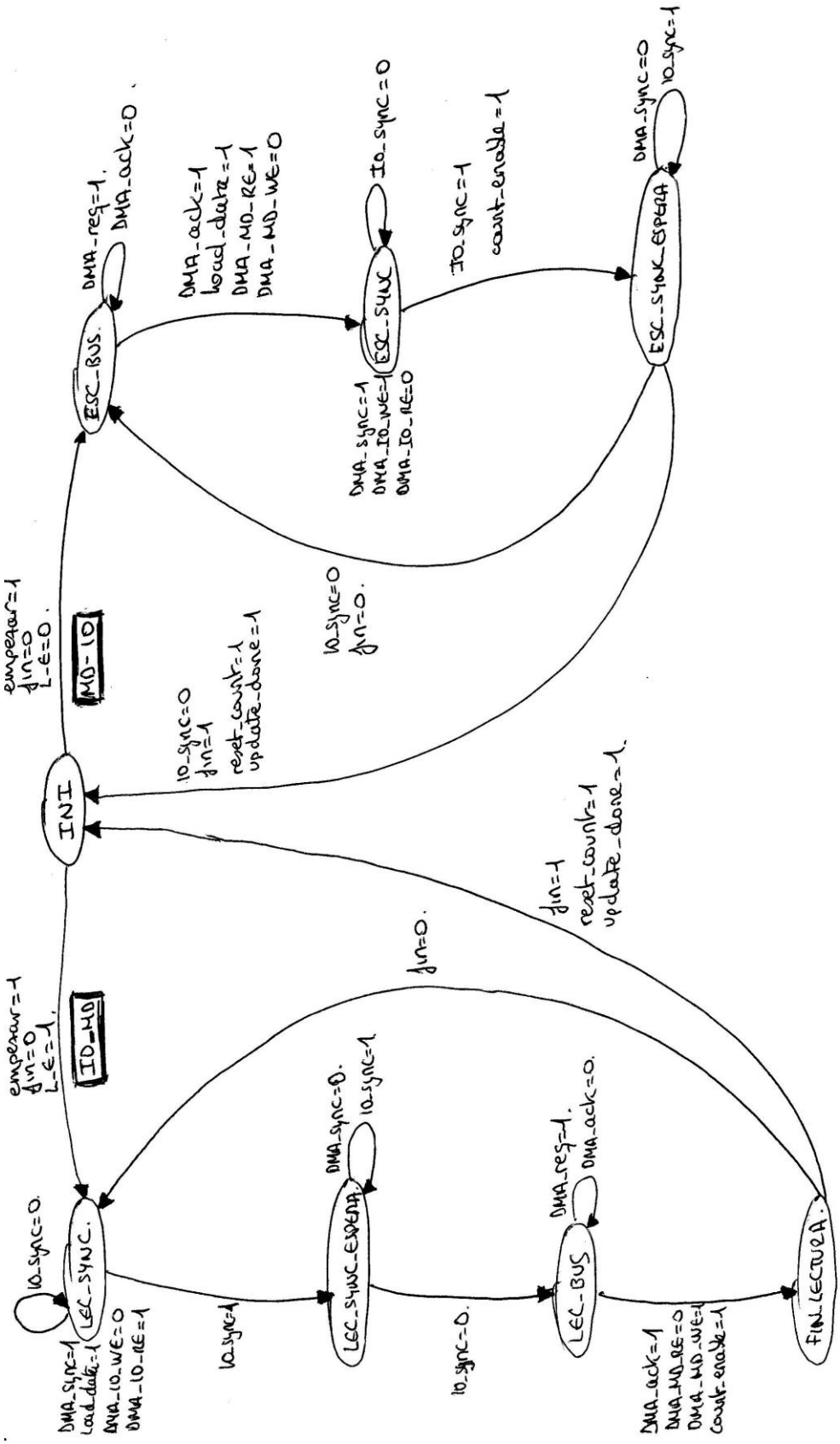
7. **ESC_SYNC:** Estado para la sincronización del DMA con el IO, para ello se pondrá DMA_sync a 1, es decir que el DMA quiere hacer algo, activaremos el registro de datos de DMA con load_data, pondremos las señales para escribir en IO y comprobaremos si IO a terminado. Si ha terminado contaremos y pasaremos al estado de espera a que IO termine. Hasta que IO no termine seguiremos en este estado.

```
--SYNC ESCRITURA
elsif (state = ESC_SYNC) then
    -- indicamos que queremos hacer algo
    DMA_sync <= '1';
    -- Escribimos en IO
    DMA_IO_WE <= '1';
    DMA_IO_RE <= '0';
    -- Si IO ha terminado
    if IO_sync= '1' then
        count_enable <= '1';
        next_state <= ESC_SYNC_ESPERA;
    else
        next_state <= ESC_SYNC;
    end if;
```

8. **ESC_SYNC_ESPERA:** Bajaremos DMA_sync (hay que esperar un ciclo para asegurarnos que el dato se ha cargado en el registro). Comprobaremos si IO a terminado, una vez haya terminado miraremos fin, para saber si tenemos que pasar al estado inicial o seguir escribiendo palabras (volver al estado ESC_BUS a pedir de nuevo el bus). Si fin resetearemos el contador, activaremos el bit de que se ha terminado la transferencia y pasaremos al estado inicial. Hasta que IO no termine seguiremos en este estado.

```
--ESTADO DE ESPERA A QUE IO BAJE ESCRITURA
elsif (state = ESC_SYNC_ESPERA) then
    DMA_sync <= '0';
    --IO ha terminado
    if(IO_sync = '0') then
        if(fin = '1') then
            -- Total de palabras completado
            reset_count <= '1';
            update_done <= '1';
            next_state <= INI;
        else
            -- Si no fin vuelve a leer de MD
            next_state <= ESC_BUS;
        end if;
    else
        next_state <= ESC_SYNC_ESPERA;
    end if;
end if;
```

Diagrama de estados:



Bancos de pruebas (señales de control):

Para comprobar el correcto funcionamiento del DMA y sus señales se han realizado unos bancos de pruebas para comprobar que en los estados definidos se obtienen correctamente todas las señales de control así como se realiza de una manera correcta las sincronizaciones entre IO y DMA, así como entre el Mips y el DMA.

Para realizar el banco de pruebas de la UC del DMA, **lo hemos dividido en 2**, que son básicamente las 2 operaciones que realiza el DMA:

IO-MD

En la que vamos recorriendo todos los casos del diagrama de estados que componen la operación de leer (**INI, LEC_SYNC, LEC_SYNC_ESPERA, LEC_BUS y FIN_ESCRITURA**).

1. **INI:** Pondremos las señales para empezar.

2. pondremos las señales con las que pasaremos al siguiente estado:

- empezar = 1
- fin = 0
- L_E = 1 -> IO-MD

3. **LEC_SYNC:** Aquí se activaran DMA_sync, load_data y lectura de IO, pondremos **IO_sync** a 1 para pasar al siguiente estado (IO ha terminado).

4. **LEC_SYNC_ESPERA:** Se bajara DMA_sync, pondremos **IO_sync** a 0 (IO terminado).

5. **LEC_BUS:** El DMA nos pedira el bus, pondremos **DMA_ack** a 1 (Se le ha concedido el Bus de datos). Cuando se le concede el bus se activaran las señales de escritura de MD y count_enable.

6. **FIN_LECTURA:** Pondremos **fin = 0**, para simular que aún queda otra palabra con la que trabajar.

7. Se volverá al estado LEC_SYNC, por lo que se repetirán los pasos 3, 4, y 5.

8. **FIN_LECTURA:** Pondremos **fin = 1**, por lo que volveremos al estado INI, activandose reset_count y update_done.

Como comprobaremos al ejecutar el banco de pruebas (**UC_DMA_test_IO-MD**), en cada estado muestra las señales correctas para cada estado.

MD-IO

En la que vamos recorriendo todos los casos del diagrama de estados que componen la operación de leer (**INI, ESC_BUS, ESC_SYNC, ESC_SYNC_ESPERA**).

1. **INI:** Pondremos las señales para empezar.

2. pondremos las señales con las que pasaremos al siguiente estado:

- empezar = 1
- fin = 0
- L_E = 0 -> MD-IO

3. **ESC_BUS:** El DMA nos pedirá el bus, pondremos **DMA_ack** a 1 (Se le ha concedido el Bus de datos). Cuando se le concede el bus se activaran las señales de load_data y lectura de MD.

4. **ESC_SYNC:** Aquí se activaran DMA_sync, load_data y escritura de IO, pondremos **IO_sync** a 1 para pasar al siguiente estado (IO ha terminado) y count_enable.

5. **ESC_SYNC_ESPERA:** Se bajara DMA_sync, pondremos **IO_sync** a 0 (IO terminado).

6. Para simular que hay mas palabras con las que trabajar pondremos **fin = 0**, por lo que se volverá al estado ESC_BUS y se repetirán los pasos 3, 4 y 5.

7. En este caso pondremos **fin = 1**, con lo que habremos terminado la transferencia, se activaran las señales de reset_count y update_done y se volvera al estado INI.

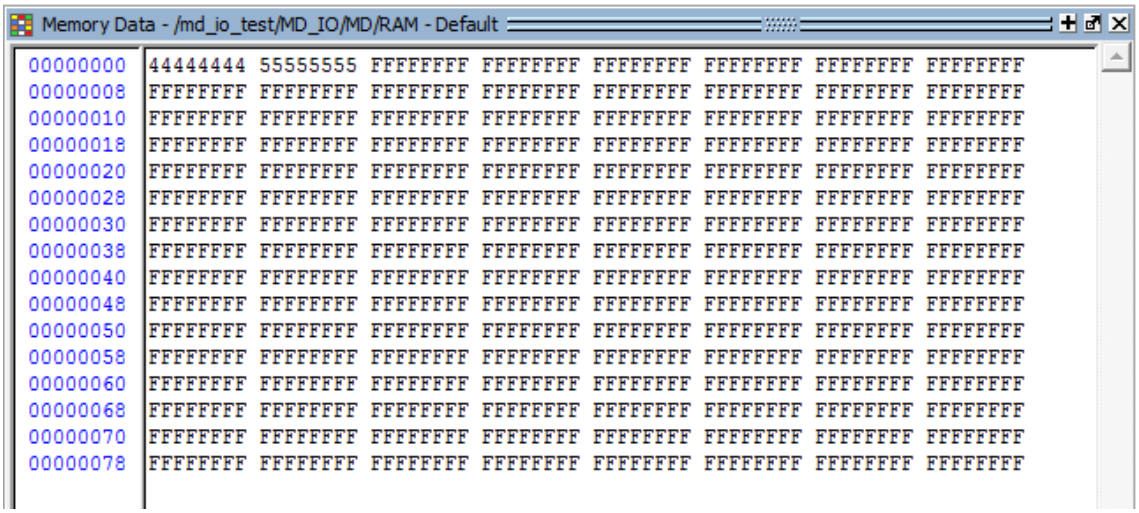
Igual que en el anterior, al ejecutar el banco de pruebas(**UC_DMA_test_MD-IO**), en cada estado muestra las señales correctas para cada estado.

**** No hemos puesto los bucles en cada estado porque consideramos que con las pruebas realizadas van a funcionar sin ningún tipo de problema.**

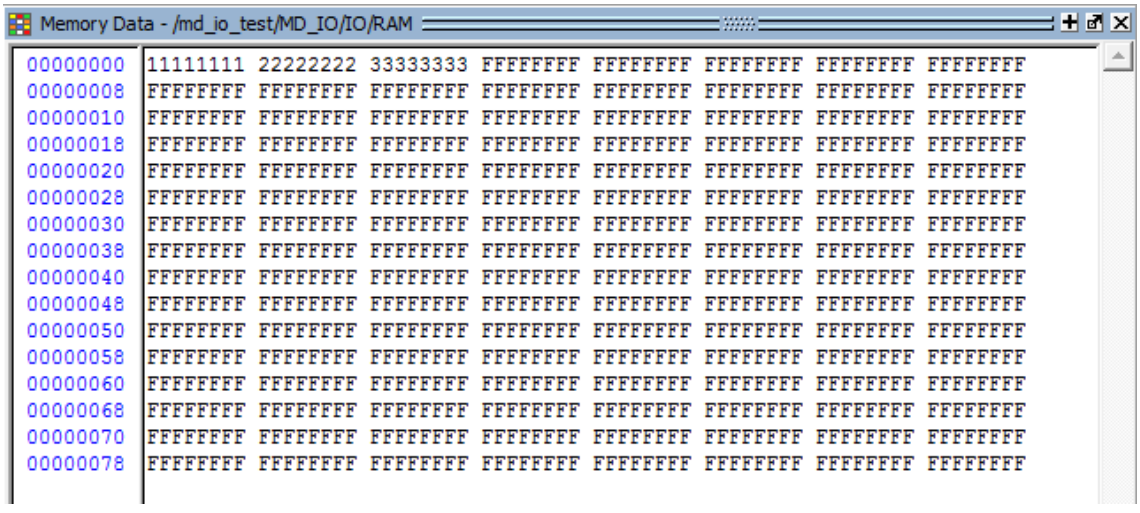
Bancos de pruebas (IO-DMA-MD):

Para verificar que el DMA actúa como debe, realizando las transferencias de datos de IO a MD y viceversa, se han realizado pruebas con el test **MD_IO_test.**

Contenido de la Memoria de MD:



Contenido de la Memoria de IO:



En este ejemplo habrá **2 instrucciones que** pasaremos a explicar a continuación:

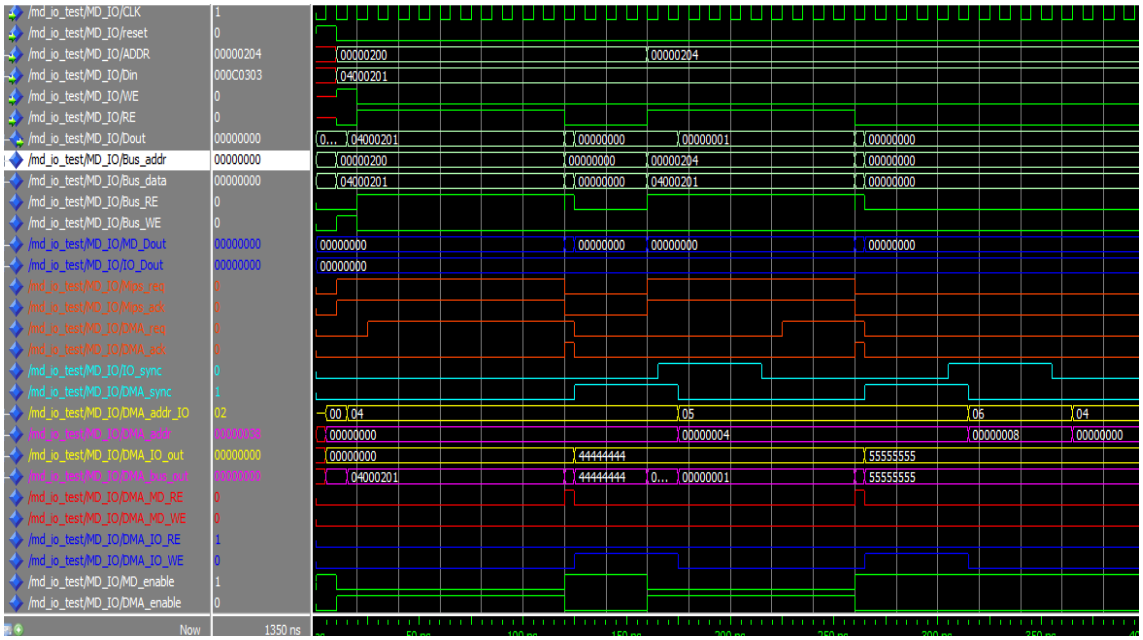
04000201

Siguiendo el esquema del registro de control, tenemos que:

- @IO: 4
- @MD: 0
- Nº palabras: 2

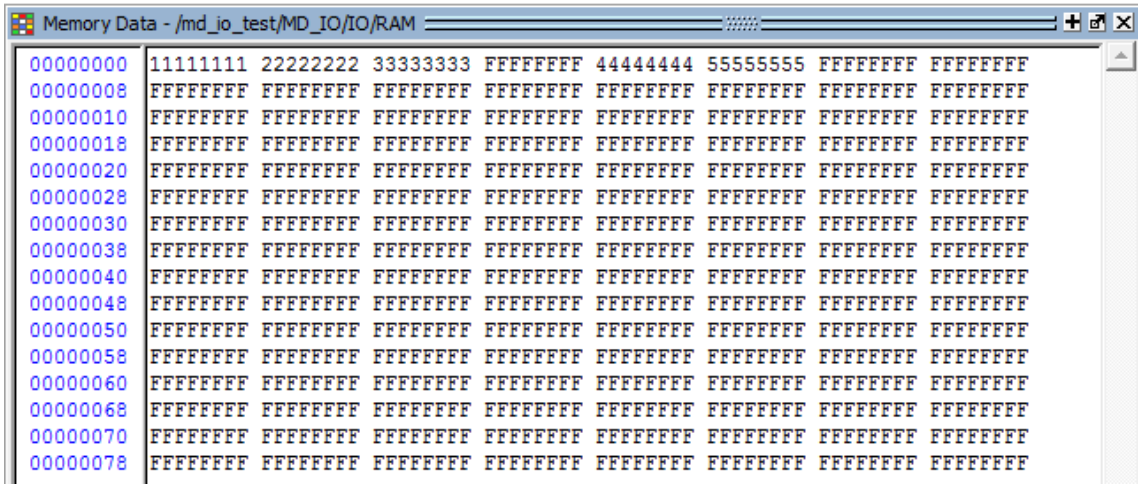
Es decir, se leerán 2 palabras de la @0 de MD y se escribirán en la @4 de IO.

Vista de la instrucción en el cronograma:



Como podemos ver, el DMA solicita el bus, cuando el Mips deja de tenerlo le llega el ack. Inmediatamente el DMA tiene el bus realiza la lectura de la primera palabra, que será 44444444 (sale de memoria, MD_Dout), acto seguido deja libre el bus e inicia la sincronización con el IO, escribiéndose en IO la palabra. Hasta que IO_sync no baje no se podrá realizar otra operación. Cuando ha terminado esta primera transmisión se inicia la segunda, de la misma forma que antes. Como vemos se respetan tanto las sincronizaciones como el bus, ya que cuando lo tiene el Mips, el DMA espera, y nada más se le concede el bus al DMA, este actúa con la operación solicitada.

Como podemos ver el contenido de la memoria del IO ha sido actualizado con la operación realizada, tiene a partir de la @4 las 2 palabras que estaban a partir de la @0 de MD:



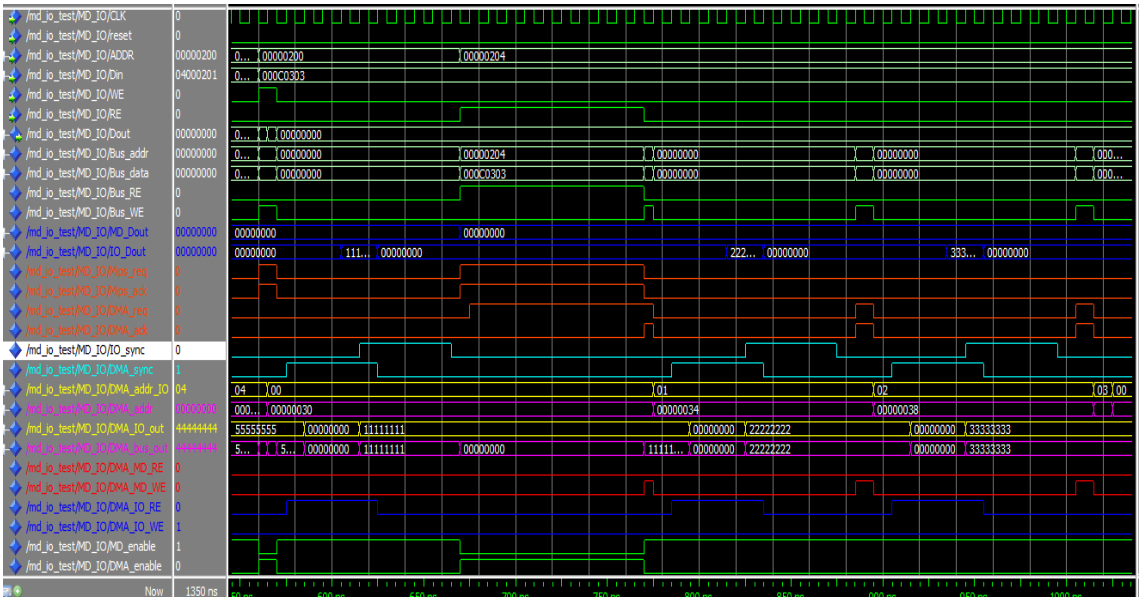
000C0303

Siguiendo el esquema del registro de control, tenemos que:

- @IO: 0
- @MD: 12
- Nº palabras: 3

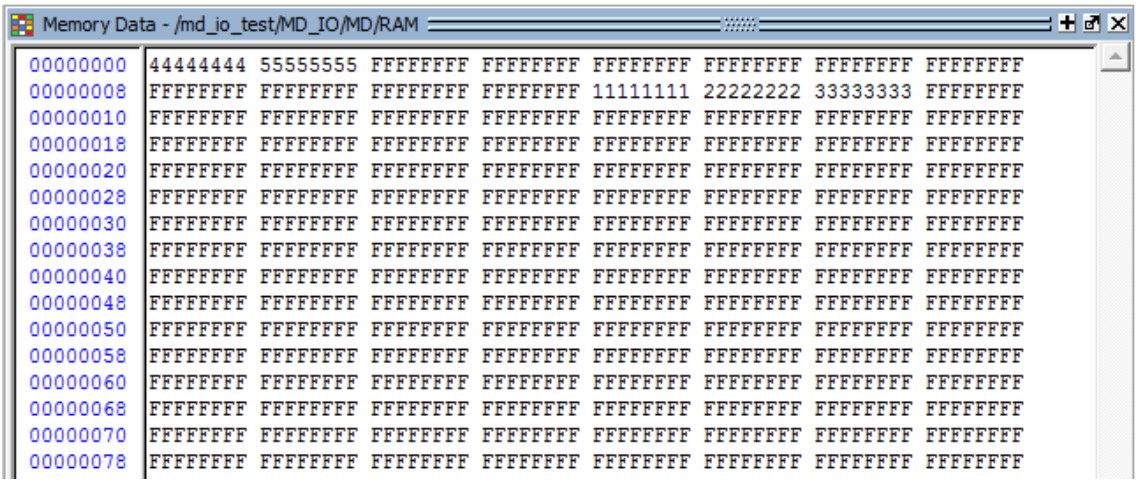
Es decir, se leerán 3 palabras de la @0 de IO y se escribirán en la @12 de MD.

Vista de la instrucción en el cronograma:



Esta vez en el cronograma podemos ver que primero se realiza la operacion de lectura de IO en @0 (DMA_addr_IO), sacando la primera palabra 11111111 (IO_Dout). Esperamos a que baje IO, una vez termina se espera a que el Mips deje libre el bus mientras DMA lo pide (DMA_req activado). Cuando el Mips deja libre el bus, inmediatamente el DMA lo coge y realiza la escritura en MD de la primera palabra. Lo mismo ocurrirá con las otras tres palabras restantes. Aquí también se puede ver que se respetan todas las sincronizaciones y el uso del bus de una manera correcta.

Como podemos ver el contenido de la memoria de la MD ha sido actualizado con la operación realizada, tiene a partir de la @12 las 3 palabras que estaban a partir de la @0 de IO:


Memory Data - /md_io_test/MD_IO/MD/RAM
00000000 44444444 55555555 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
00000008 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 11111111 22222222 33333333 FFFFFFFF
00000010 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
00000018 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
00000020 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
00000028 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
00000030 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
00000038 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
00000040 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
00000048 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
00000050 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
00000058 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
00000060 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
00000068 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
00000070 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
00000078 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF