Report for Lab7: Direct Memory Access

20140843 Taekang Eom

20150384 Eunyoung Hyung

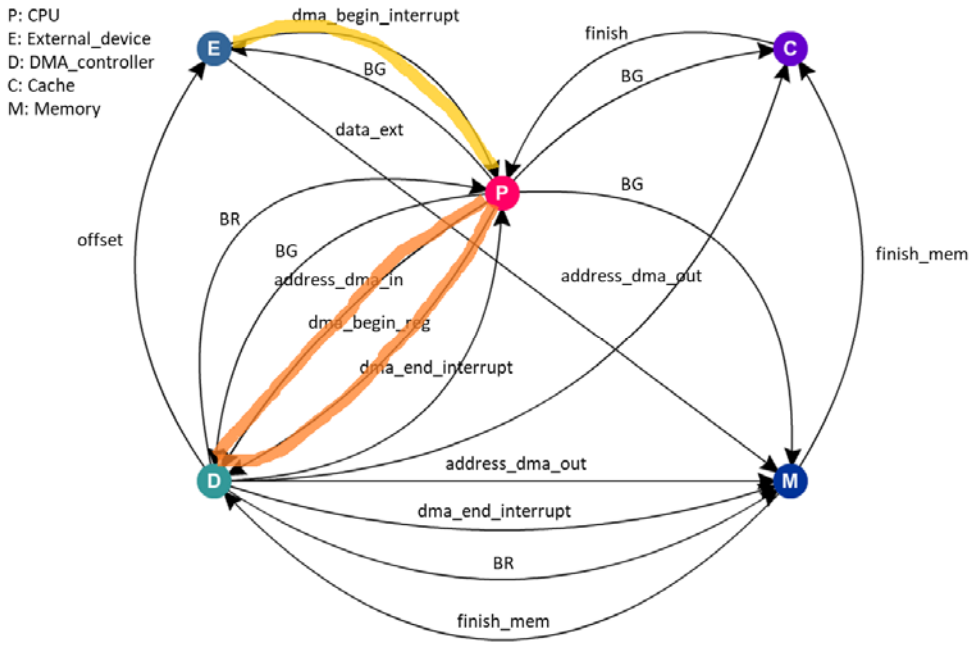
1. Introduction

In this lab, we need to implement Direct Memory Access(DMA) which makes the external I/O device directly access the memory, not through CPU. DMA controller which controls the bus to act on half of CPU can make DMA feasible. To do this, we need to implement a simple external device and DMA controller and then modify CPU to handle the interrupt caused by DMA controller.

2. Design

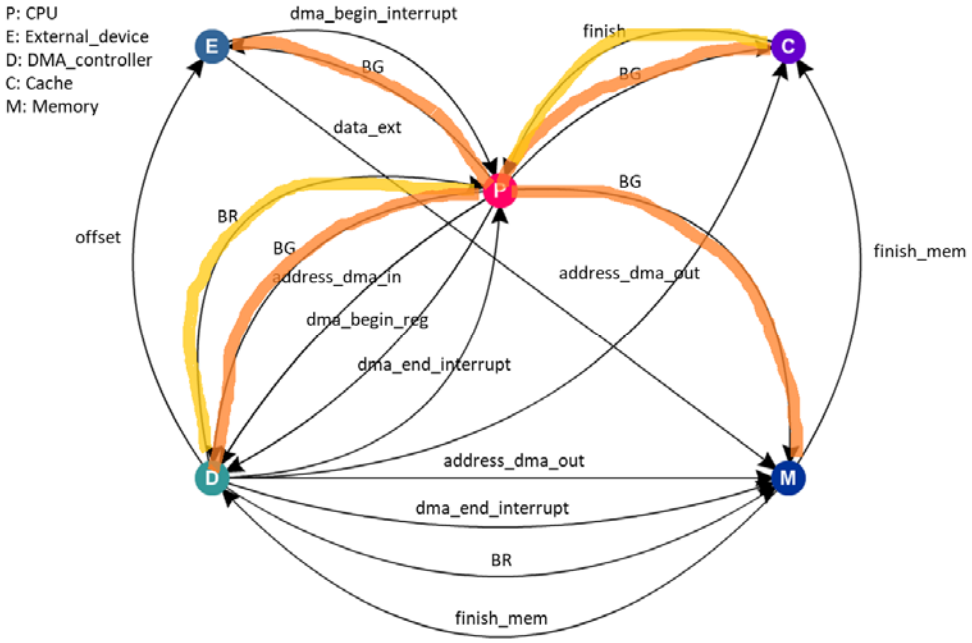
1) First, we explain how all modules work together in signal-wise.

① An external device sends an interrupt to a CPU then CPU sends the address and dma_begin_reg signal to DMA controller.

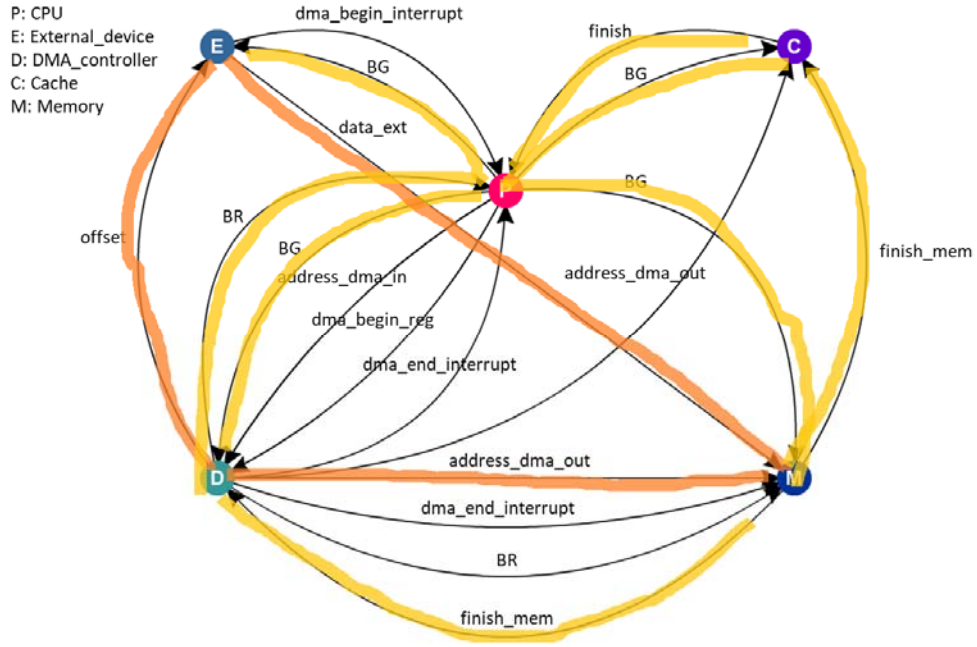


② DMA controller sends Bus Request(BR) to CPU. CPU blocks itself and in the case

that CPU can use the cache, then it sets Bus Granted(BG).

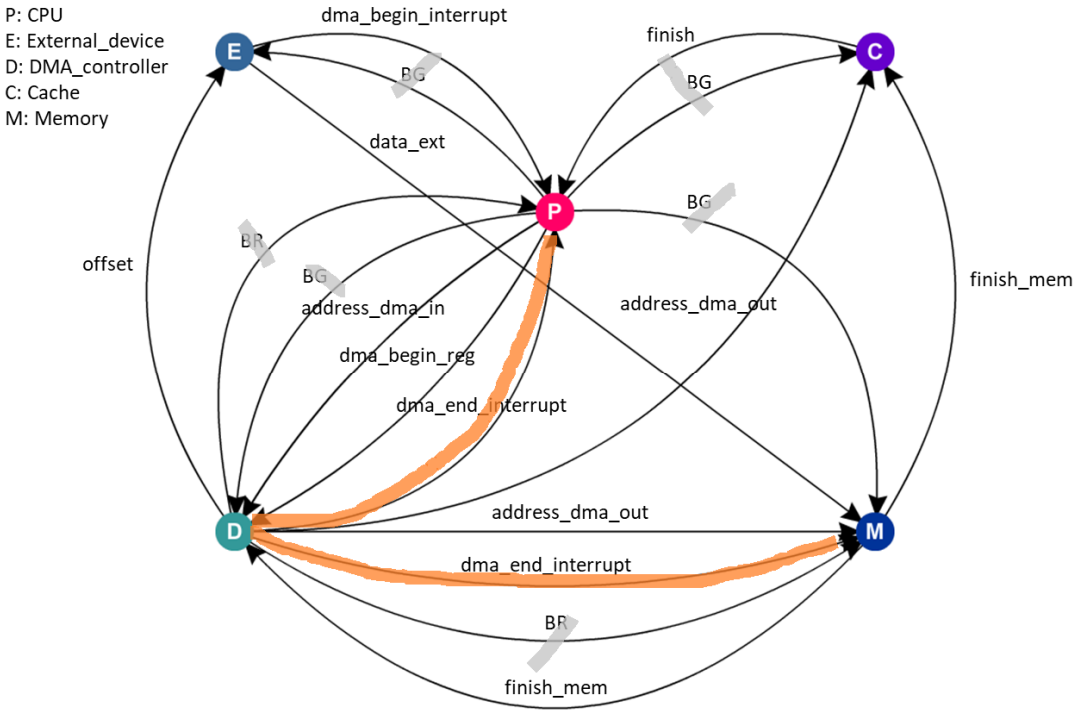


③ If DMA gets BG and memory accessing is available, DMA controller keeps setting the offset of data by 4 until it reaches the end and the address received from CPU(address_dma_in). At the same time, external devices can write the proper data on the bus to the memory while the BG is on.



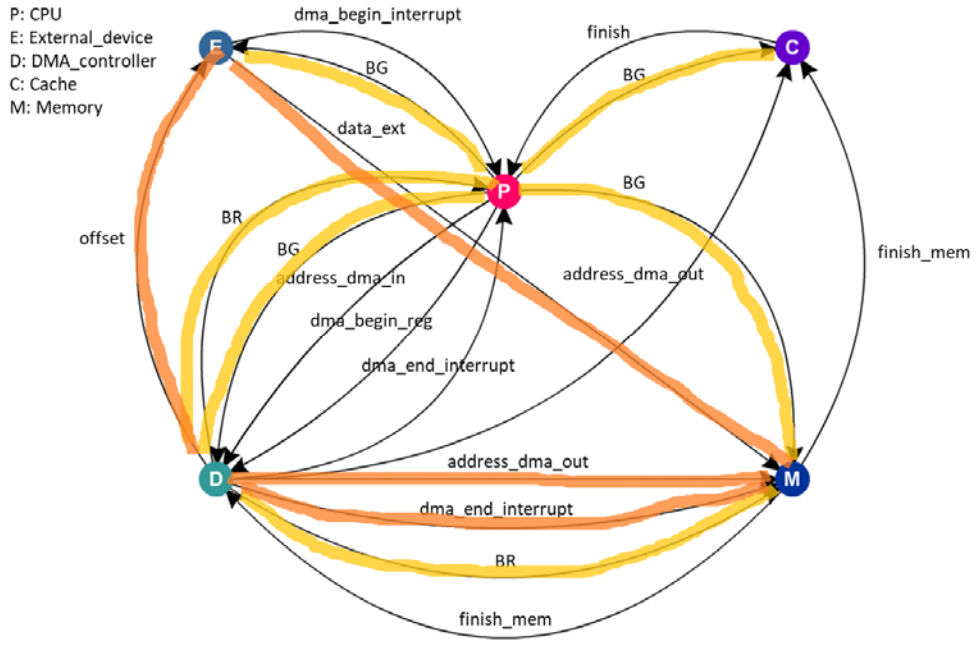
④ When the DMA controller finishes its work, it clears BR and sends the end interrupt to CPU and memory. Then CPU also clear the BG so the CPU can use memory

again. (Second case is explained in 2-2) part)



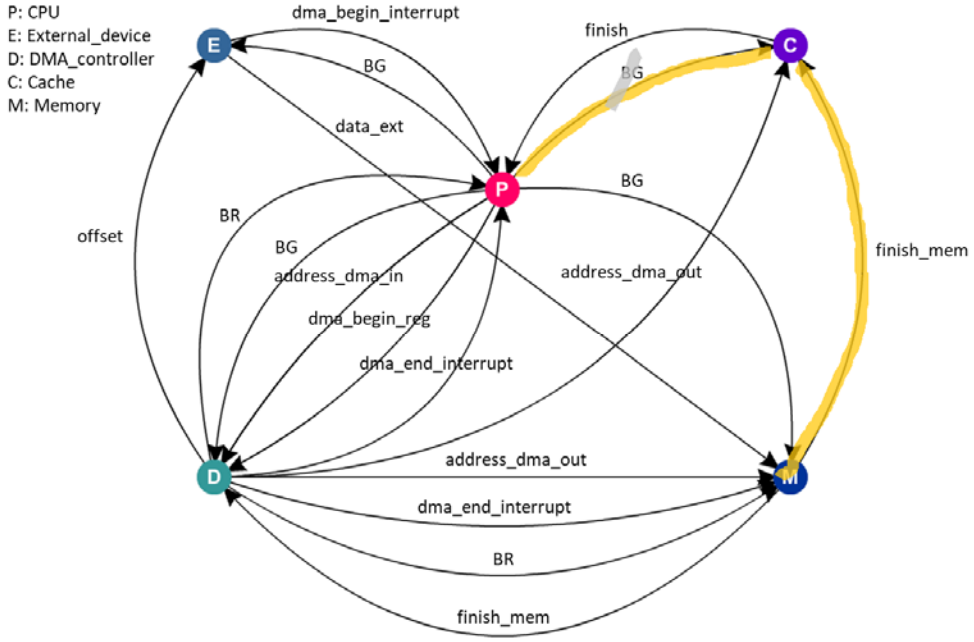
2) Memory view

If write from external device is needed(BR && BG), it uses (address_dma_out + offset) to write at proper memory location. And for grading, we use dma_end_interrupt to print values in the memory.



3) Cache view

Only when memory is accessible and BG is off, cache can read data from the memory.



4) Invalidating the cache line corresponding to the received address

The cache line may be out-of-sync just after the memory receives data from the external device so, we invalidate the cache line corresponding to the received address when BG is on.

3. Implementation

We only address the newly added or updated modules.

1) Opcode.v

```
// we set the address for the external device from 16'h10~16'h1b
`define DEVICE_ADDRESS 16'h10
`define DEVICE_LENGTH 16'd12
```

2) external_device.v

```
module external_device (data, BG, offset, dma_begin_interrupt);
    /* input */
    input BG;
```

```
    input [`WORD_SIZE-1:0] offset;

    /* output */
    output reg dma_begin_interrupt;
    output wire [`BLOCK_SIZE-1:0] data;

    /* external device storage */
    reg [`WORD_SIZE-1:0] stored_data [11:0];

    /* When the external device gets the offset of data from DMA Controller,
    write the proper data on the data bus to the memory */
    assign data = BG?
{stored_data[offset+16'b11],stored_data[offset+16'b10],stored_data[offset+16'b
01],stored_data[offset+16'b00]}:64'b0;

    /* Initialization */
    initial begin
        dma_begin_interrupt <= 0;
        stored_data[0] <= 16'h0000;
        stored_data[1] <= 16'h1111;
        stored_data[2] <= 16'h2222;
        stored_data[3] <= 16'h3333;
        stored_data[4] <= 16'h4444;
        stored_data[5] <= 16'h5555;
        stored_data[6] <= 16'h6666;
        stored_data[7] <= 16'h7777;
        stored_data[8] <= 16'h8888;
        stored_data[9] <= 16'h9999;
        stored_data[10] <= 16'haaaa;
        stored_data[11] <= 16'hbbbb;
    end

    /* Interrupt CPU at some time */
    initial begin
    /* it waits few hundreds of clock cycles, then raise an interrupt*/
        #50000;
        $display("LOG: Start DMA! #1");
        dma_begin_interrupt = 1;
        #60;
        dma_begin_interrupt = 0;
        #(150000-60-50000)
        $display("LOG: Start DMA! #2");
        dma_begin_interrupt = 1;
        #60;
        dma_begin_interrupt = 0;
    end
endmodule
```

3) DMA_controller.v

```
module DMA_controller (clk,start, address, address_dma, BG, BR, offset,
dma_end_interrupt, finish);
    /* input */
    input clk;
    wire clk;
    input start;
    wire start;
    input BG;
    wire BG;
    input [`WORD_SIZE-1:0] address;
    wire [`WORD_SIZE-1:0] address;
    input finish;
    wire finish;

    /* output */
    output BR;
    reg BR;
    output dma_end_interrupt;
    reg dma_end_interrupt;
    output [`WORD_SIZE-1:0] address_dma;
    reg [`WORD_SIZE-1:0] address_dma;
    output [`WORD_SIZE-1:0] offset;
    reg [`WORD_SIZE-1:0] offset;

    reg [`WORD_SIZE-1:0] remain;
    reg [`WORD_SIZE-1:0] dma_num;

    /* initialize */
    initial begin
        dma_end_interrupt = 1'b0;
        offset = 16'b0;
        BR = 1'b0;
        dma_num = 16'b0;
    end
    always @(posedge clk) begin
        if (start) begin // If external device triggers the interrupt
            BR <= 1'b1; // signal BR to CPU
            address_dma <= address; // then cpu set address <=
`DEVICE_ADDRESS;
            remain <= `DEVICE_LENGTH; // total length is 12
            dma_num <= dma_num + 1;
        end
        if(BG && finish) begin // If DMA gets BG from CPU and memory accessing
is finised
```

```
            if(remain != 16'd0) begin // keep setting the offset by 4 till it
ends
                offset <= `DEVICE_LENGTH - remain;
                remain <= remain - 16'd4;
            end
            else begin
                $display("LOG: End DMA! #%d",dma_num);
                dma_end_interrupt <=1'b1; // let CPU know DMA's finished
                BR <= 1'b0; // So clear the BR
            end
        end
        if(dma_end_interrupt) begin
            dma_end_interrupt <=1'b0; // and then clear the end interrupt
signal
        end
    end

endmodule
```

4) cpu.v

```
module cpu(Clk, Reset_N, finish, dma_begin_interrupt,dma_end_interrupt,
dma_begin_reg, address_dma, BR, BG, readM1, address1, data1, readM2, writeM2,
address2, data2, num_inst, output_port, is_halted);

       // Omitted
       // Omitted

       // DMA part
        /* if dma_begin_interrupt has been occurred,
        it sets dma_begin_reg as same as dma_begin_interrupt
        and send it to the DMA controller with the deviece address */
        if(!dma_begin_interrupt) begin
            dma_begin_reg <= dma_begin_interrupt;
        end
        else begin
            address_dma <= `DEVICE_ADDRESS;
            dma_begin_reg <= dma_begin_interrupt;
        end
        // If DMA signals BR and it can use cache, set BG
        if(BR && finish) begin
            BG <= 1'b1;
        end
        // If DMA signals end_interrupt, clear BG
        if(dma_end_interrupt) begin
            BG <= 1'b0;
```

```
        end
    end
endmodule
```

5) cache.v

```
module Cache(clk, reset_n, readM1, address1, data1, readM2, writeM2, address2,
data2,
 finish,access_num,hit_num, readM1_mem,readM2_mem,
writeM2_mem,data1_mem,write_data, data2_mem, evict1,evict2, finish_mem,
address_dma, BG);

       // Omitted
       // Omitted

   /* The cache line may be out-of-sync just after the memory receives data
from the external device
        so, invalidate the cache line corresponding to the received address */
           if(address_dma[15:5] == cache[address_dma[4:2]][76:66] && BG)
begin
               cache[address_dma[4:2]][`VALID] <= 1'b0;
           end

       // Omitted
       // Omitted

           else if(finish_mem && !BG) begin // when memory is accessible and
BG is off, it can read data from memory
               cache[address1[4:2]] <= {address1[15:5],1'b1,1'b0,data1_mem};
               if(address1[4:2] != address2[4:2]) begin
                   cache[address2[4:2]] <=
{address2[15:5],1'b1,1'b0,data2_mem};
               end

       // Omitted
       // Omitted
```

6) Memory.v

```
module Memory(clk, reset_n, readM1, address1, data1, readM2, writeM2,
address2,write_data, data2, evict1, evict2, finish, BG, BR, address_dma,
data_ext, dma_end_interrupt);

       // Omitted
```

```
       // Omitted
            memory[16'hc5] <= 16'hf819;
            memory[16'hc6] <= 16'hf01d;
        end
        else if((readM1 || readM2 || writeM2) && (!BG || !BR)) begin
// memory access caused by not external devices (same as before)

       // Omitted
       // Omitted

        else if (BG && BR) begin
// write memory from device, then use address from dma and data from external
            if(finish) begin
                memory[address_dma+16'b0] <= data_ext[15:0];
                memory[address_dma+16'b1] <= data_ext[31:16];
                memory[address_dma+16'b10] <= data_ext[47:32];
                memory[address_dma+16'b11] <= data_ext[63:48];
                finish <= 1'b0;
                timer <= 16'b100;
            end
            else if(timer != 16'b0) begin
                timer <= timer -1;
            end
            else begin
                finish <= 1'b1;
            end
        end
        if(dma_end_interrupt) begin // for showing that it works properly
            $display("memory[16'h10]:%h",memory[16'h10]);
            $display("memory[16'h11]:%h",memory[16'h11]);
            $display("memory[16'h12]:%h",memory[16'h12]);
            $display("memory[16'h13]:%h",memory[16'h13]);
            $display("memory[16'h14]:%h",memory[16'h14]);
            $display("memory[16'h15]:%h",memory[16'h15]);
            $display("memory[16'h16]:%h",memory[16'h16]);
            $display("memory[16'h17]:%h",memory[16'h17]);
            $display("memory[16'h18]:%h",memory[16'h18]);
            $display("memory[16'h19]:%h",memory[16'h19]);
            $display("memory[16'h1a]:%h",memory[16'h1a]);
            $display("memory[16'h1b]:%h",memory[16'h1b]);
        end
    end
endmodule
```

4. Discussion

As 3-1) says, we set the address for the external device from 16'h10~16'h1b. The values in the memory in that range after end interrupt are like below, so we can conclude our DMA controller is working well.

We use the same test bench for lab6 and we get "all pass" so we can say it is still functionally correct even with DMA

```
VSIM 9> run -all
# LOG: Start DMA! #1
# LOG: End DMA! #    1
# memory[16'h10]:0000
# memory[16'h11]:1111
# memory[16'h12]:2222
# memory[16'h13]:3333
# memory[16'h14]:4444
# memory[16'h15]:5555
# memory[16'h16]:6666
# memory[16'h17]:7777
# memory[16'h18]:8888
# memory[16'h19]:9999
# memory[16'h1a]:aaaa
# memory[16'h1b]:bbbb
# LOG: Start DMA! #2
# LOG: End DMA! #    2
# memory[16'h10]:0000
# memory[16'h11]:1111
# memory[16'h12]:2222
# memory[16'h13]:3333
# memory[16'h14]:4444
# memory[16'h15]:5555
# memory[16'h16]:6666
# memory[16'h17]:7777
# memory[16'h18]:8888
# memory[16'h19]:9999
# memory[16'h1a]:aaaa
# memory[16'h1b]:bbbb
# Clock # 1981
# Access # 1442, Hit # 1354
# The testbench is finished. Summarizing...
# All Pass!
# ** Note: $finish    : C:/Modeltech_pe_edu_10.4a/examples/lab7/cpu_TB.v(172)
#    Time: 198250 ns  Iteration: 2  Instance: /cpu_TB
```

5. Conclusion

We understand what is DMA and how DMA controller can make this possible. While implementing it, we can know about this concept more well. We know without DMA, CPU's performance could not be better when with I/O device.