Computer Organization Report

Lab08 DMA

2019-12172 이재원

1 Introduction

Understand and implement simplified DMA that only writes to data memory at fixed address and length

2 Design

2.1 DMA module

2.1.1 Baseline

2.1.1.1 BR (Bus Request)

If command is fetched from cpu, BR should be 1. And if memory write is finished BR should be 0.

As end of interrupt is same as BR changing from 1 to 0, interrupt end signal will not be used.

2.1.1.2 Counter

When BG (Bus Granted), DMA should write on memory. However, as there is no write done signal from memory, DMA should have a counter to specify the time when DMA_write ends.

The counter must increase by 1 every cycle when memory write (i.e. BG), and stop after the length recieved from cpu command, which is fixed to 12.

2.1.2 Cycle stealing

Only BR is changed if cycle stealing is implemented. Other logic is the same. BR must be 0 every 4-word memory write (i.e. 4cycles), and changed to 1 next cycle.

2.2 Cpu interrupt handler

2.2.1 cmd

If cpu receives DMA begin signal, cpu must transfer cmd signal at next cycle.

2.2.2 BG (Bus Granted)

If BR is 1, cpu should handle BG. After data memory read stall or data memory write stall it becomes 1.

2.3 Cache stall

When BG, cache should stop (stall) memory access. For read miss, memory access is needed, therefore must stall.

However, for write it must stall at BG whether cache miss or hit. This is because cache write is implemented as write-through. In case of write-through, memory should be written in both cache miss and hit, where stall is need when BG.

3 Implementation

3.1 *cpu_TB.v*

mux is added because data and address from DMA and cpu both goes in / out to memory.

```
    // data bus mux
    // choose d_data or dma_data to d_data_bus(memory)
    // implemented mux with high impedance because d_data bus is inout port
wire [`WORD_SIZE * 4 - 1 : 0] d_data_bus;
    assign d_data_bus = BG && dma_WRITE ? dma_data : 64'hz;
    assign d_data_bus = !BG && d_writeM ? d_data : 64'hz;
    assign d_data = !BG && d_readM ? d_data_bus : 64'hz;

    wire [`WORD_SIZE - 1 : 0] d_address_bus;
    assign d_address_bus = BG ? dma_addr : d_address;
```

Figure 1. mux from *cpu_TB.v*

3.2 *interrupt_handler.v*

Declared a module in cpu module. It handles interrupt.

3.3 *cache.v*

cache stall can be easily implemented by making memory read or write signals to 0 when BG

```
if (!hit) begin
   // Read data from lower memory into the cache block

    readM <= BG ? 0 : 1; // if BG don't readM
   writeM <= 0;
end
```

Figure 2. BG added to cache.v

3.4 *DMA.v*

Counter changes from 0 to fixed length (i.e. 12), and transfers signal based on counter.

```
// counter to count num of cycles of memory access
reg [`WORD_SIZE - 1 : 0] counter;
always @ (posedge CLK) begin
    if (BG && counter == `LENGTH) counter <= 0;
    else if (BG) counter <= counter + 1;
end
```

Figure 3. counter form *DMA.v*

3.5 *DMA_cycle_stealing.v*

The only difference from *DMA.v* is BR logic. Here, BR signal is split by using counter.

```
// BR logic;
// BR is 1 after cmd and turns to 0 when write ends (when counter is LENGTH)
always @ (posedge CLK) begin
    if (cmd) BR <= 1;
    else if (BR == 0 && (counter == `LENGTH - 8 || counter == `LENGTH -4)) BR <= 1;   // cycle stealing: set BR to 1 every 1 word
    // because of cycle stealing.set BR to 0 after 1 word dma write
    else if (counter == `LENGTH - 1 || counter == `LENGTH - 5 || counter == `LENGTH - 9) BR <= 0;
end
```

Figure 4. BR logic from *DMA_cycle_stealing.v*

3.6 Waveform

In the given skeleton code, FIRE_TIME is 2600, where DMA writes when no data memory access. Therefore, if changed to 42900, write hit and miss occurs when DMA is writing to data memory.

In case for write miss, if there is no DMA interrupt, it takes “Cache(1cycle) + memory_read(4cycle) + cache(1cycle) + memory_write(4cycle) + cache(1cylcle) = 11 cycles”. Note that in case of write miss, data memory is read first. Figure 5 and 6 shows this case when DMA interrupted.

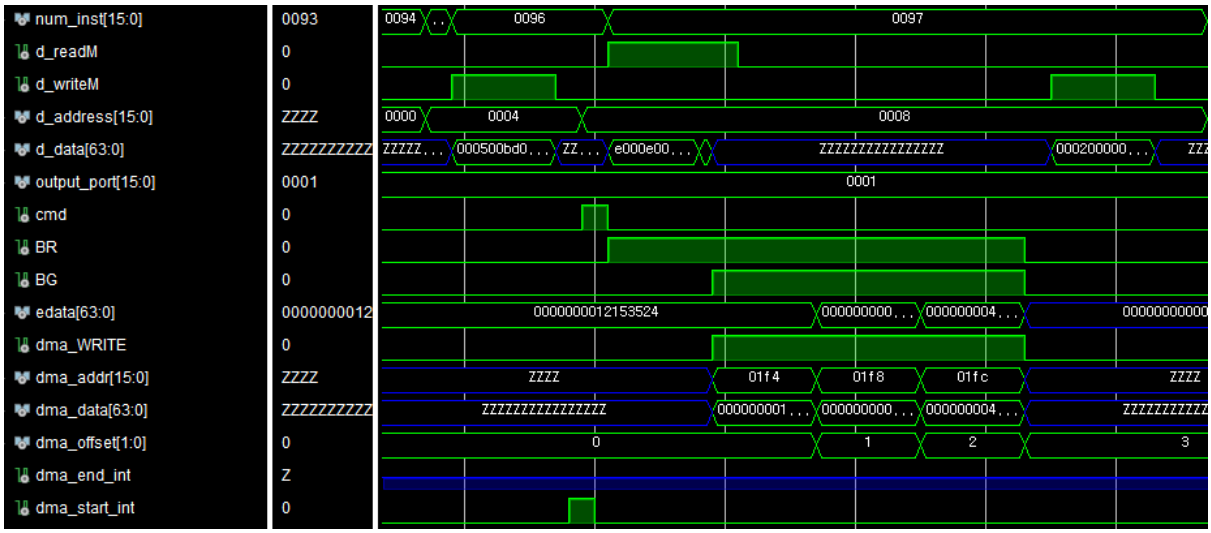

Figure 5. DMA interrupt when cache write miss. DMA is implemented without cycle stealing. FIRE_TIME is set to 42900. Write is stalled in cache, and starts writing after DMA write.

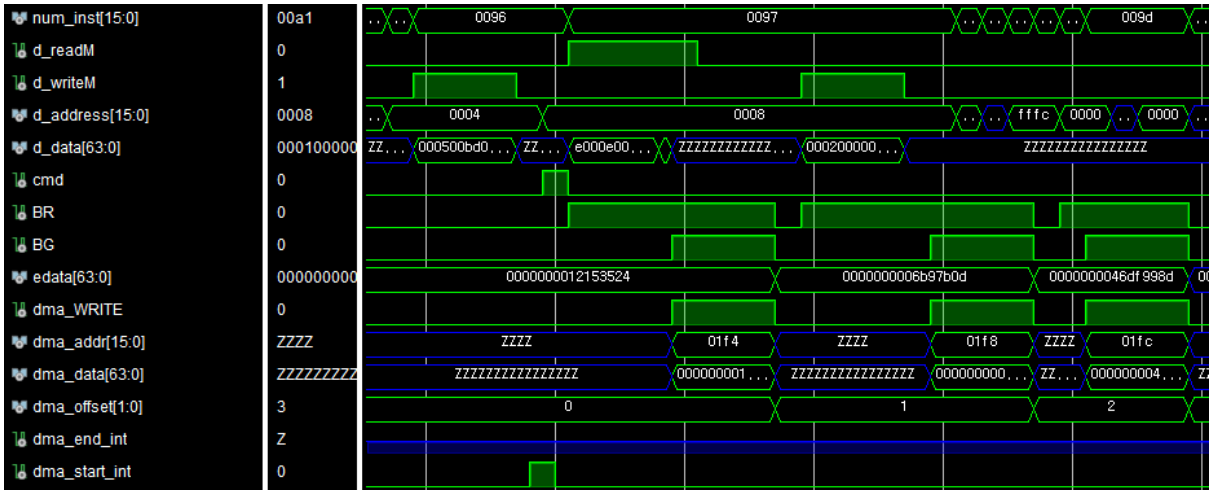

Figure 6. DMA interrupt when cache write miss. It is the same situation form Figure 5, but DMA is implemented with cycle stealing. FIRE_TIME is set to 42900. Write is stalled in cache, and starts writing in middle of first 4-word DMA write and second 4-word DMA write.

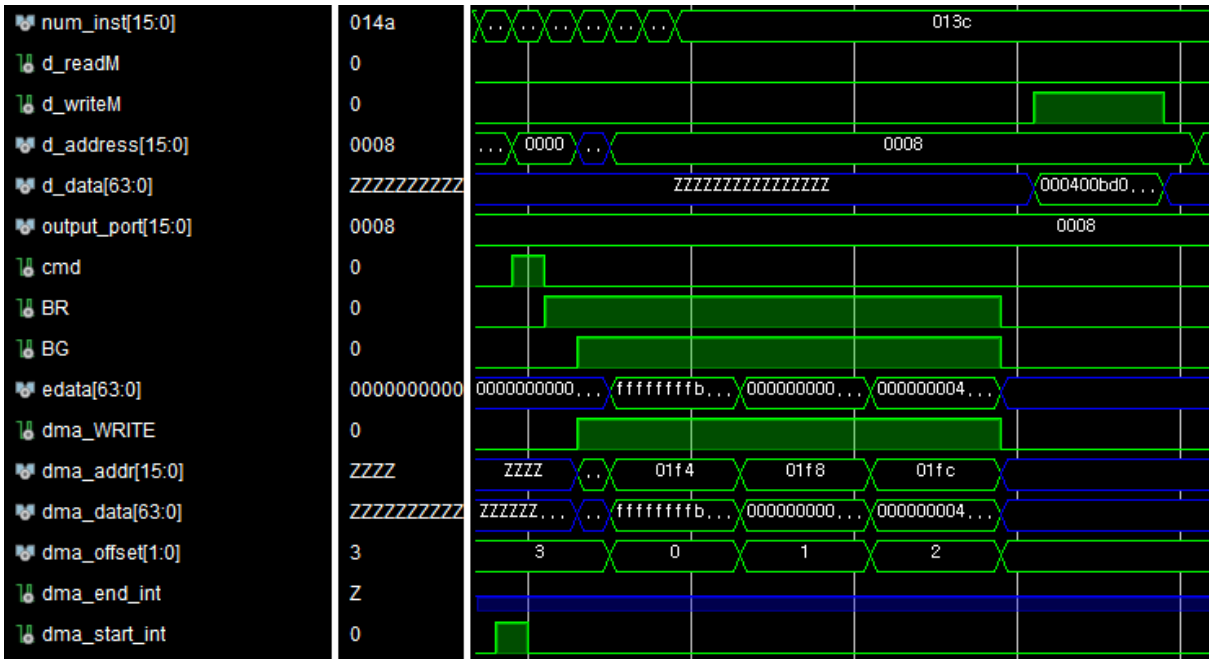

Figure 7. DMA interrupt when cache write hit. Note that, cache must stall at DMA interrupt if write hit because cache is implemented write-through. FIRE_TIME is set to 42900, but at different cycles from Figure 5.

4 Discussion

		Total instructions	Num of cycles	IPC
FIRE_TIME 2600	**DMA w/o cycle stealing**	3036	5110	0.594129
	DMA w/ cycle stealing			
FIRE_TIME 42900	**DMA w/o cycle stealing**		5133	0.591467
	DMA w/ cycle stealing		5114	0.593664

Just like from Figure 6, cycle stealing has better performance in case where FIRE_TIME is set to 42900. Total number of cycles reduced by 19 cycles when FIRE_TIME 42900, cycle stealing. This is because 8cycle loss (first DMA interrupt) + 12 cycle loss (second DMA interrupt).

5 Conclusion

Successfully implemented DMA, interrupts and checked performance enhance of cycle stealing.