Verilog Implementation Report

Computer Organization Project 8

Jaewon Chung, 2015-11832
2019/05/30

1. Introduction

- In this project, I will implement the DMA(Direct Memory Access) module, which interfaces between the external device and the memory so that the CPU stalls minimally while the external device accesses the memory.

2. Design

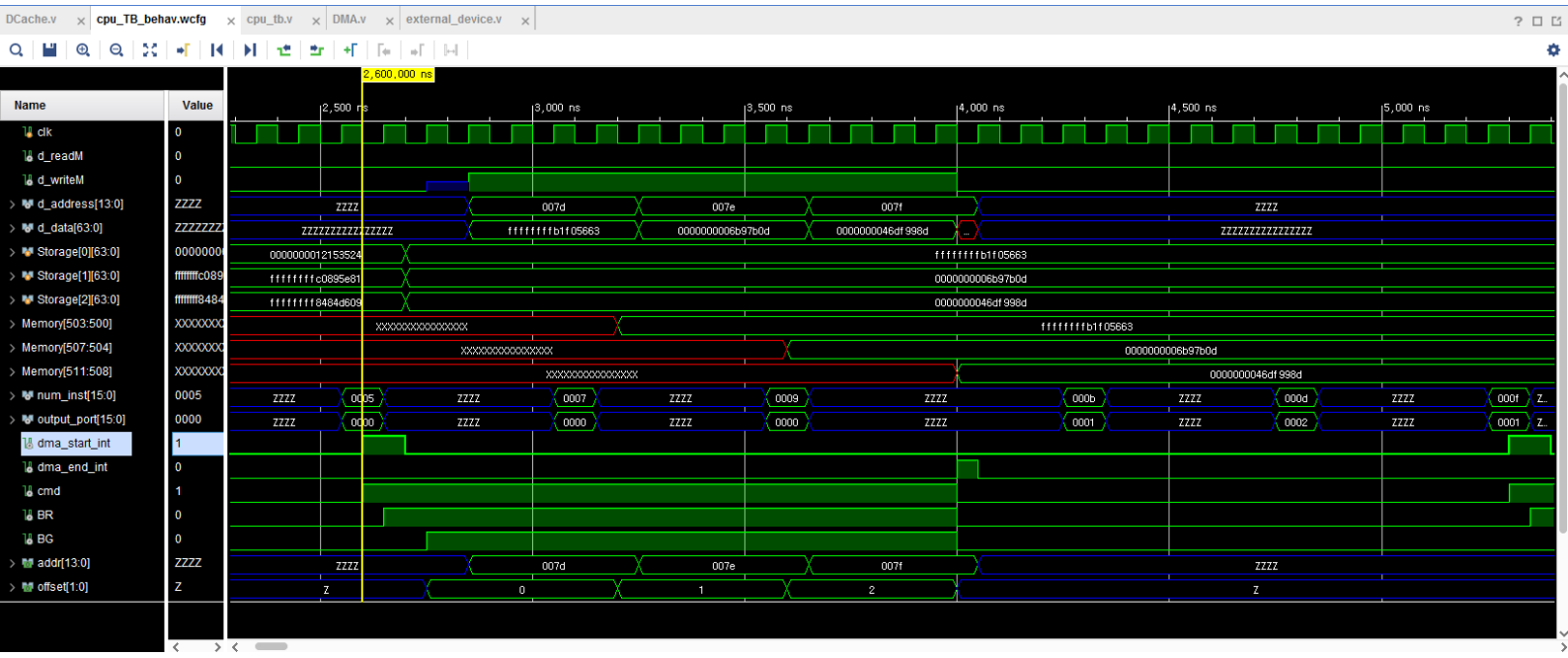
This part is replaced by the design document attached behind.

3. Implementation

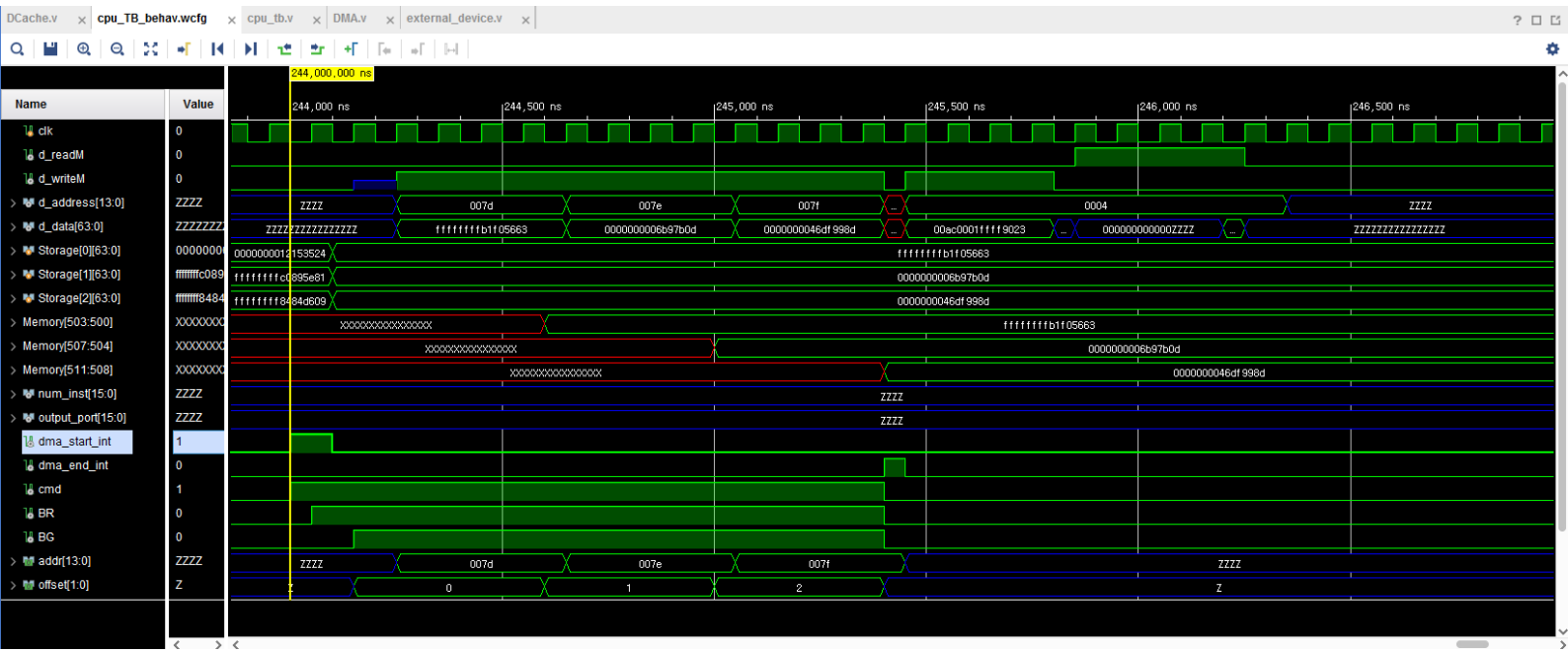
- The CPU handles the DMA request by asserting the DMA command signal. On receiving this command, the DMA module requests bus access. This signal is removed when the DMA module sends the DMA end interrupt.
- Data cache solely has access to the memory ports. On detecting a bus request, the data cache checks if the memory is currently being accessed. If not, it grants bus access by asserting BG and putting high-impedance to d_writeM, d_address, and d_data. If memory is being accessed, it does not grant bus access. This check is done every clock positive edge.
- The part of the DMA module that writes data to the memory is trivial, and its structure is identical to that of the data cache. It keeps a counter for latency checking and a block offeset register.
- The baseline model resets the memory latency counter and increments the block number on the clock negative edge. However it does not remove the WR and BR unless it was the third block that just completed the write.
- Cycle stealing is implemented with a tiny change to the baseline model. We always puts high-impedence to WR and deassert BR on the clock negative edge when a block write is done. Then on the very next clock positive edge BR is re-asserted. The data cache detects BR = 0 on that cycle. If the memory does not have to be accessed by the cache, bus will be granted immediately on the next cycle, letting the DMA module write the next block to the memory. If a memory access is required and waiting on the CPU side, that will begin first, making the DMA module wait. Due to the reset logic implementation, the DMA module must keep a register that indicates whether or not its cycle is stolen during a DMA operation.

4. Discussion

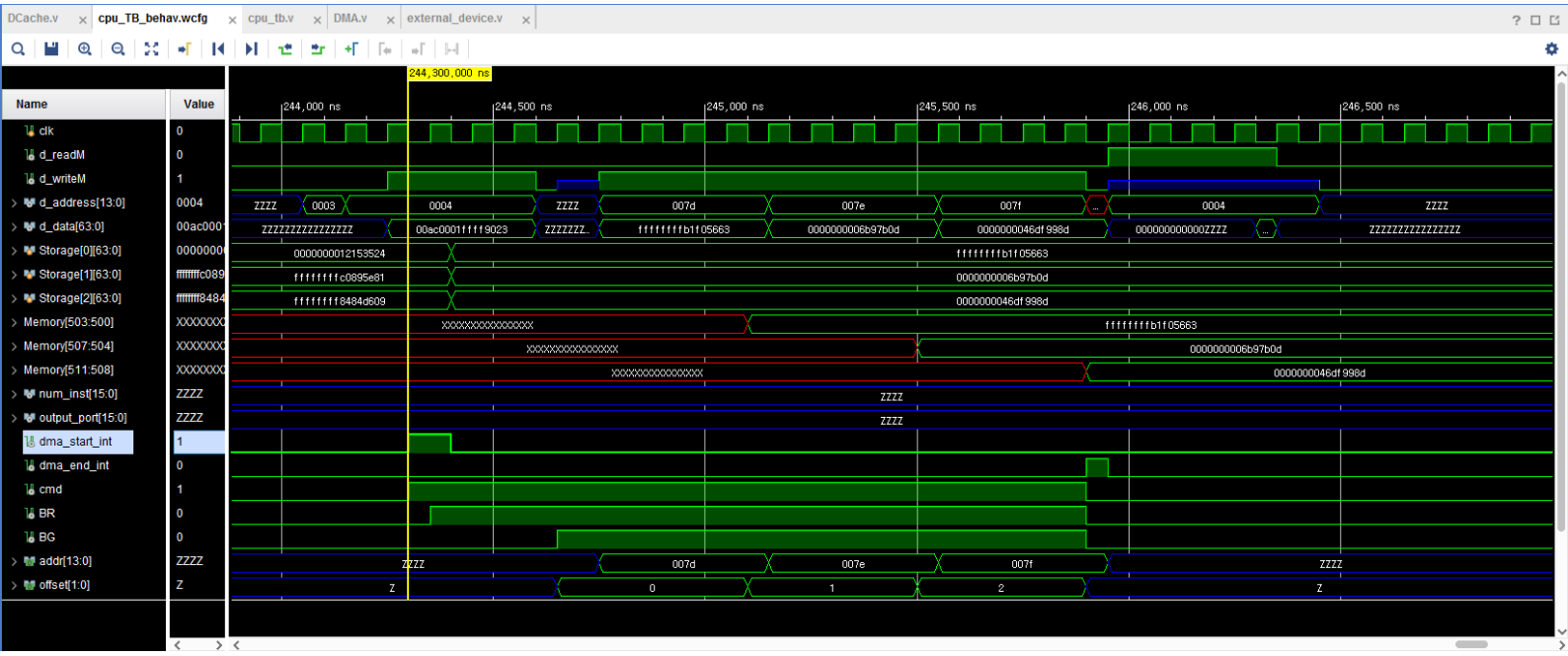
- Baseline model



FIRE_TIME is 2,600. Notice that num_inst and output_port changes; ordinary instructions that do not require memory access can proceed without any stalls. This is true for memory access instructions that hit on cache, because cache hits are serviced combinationally without changing any state registers in the data cache.

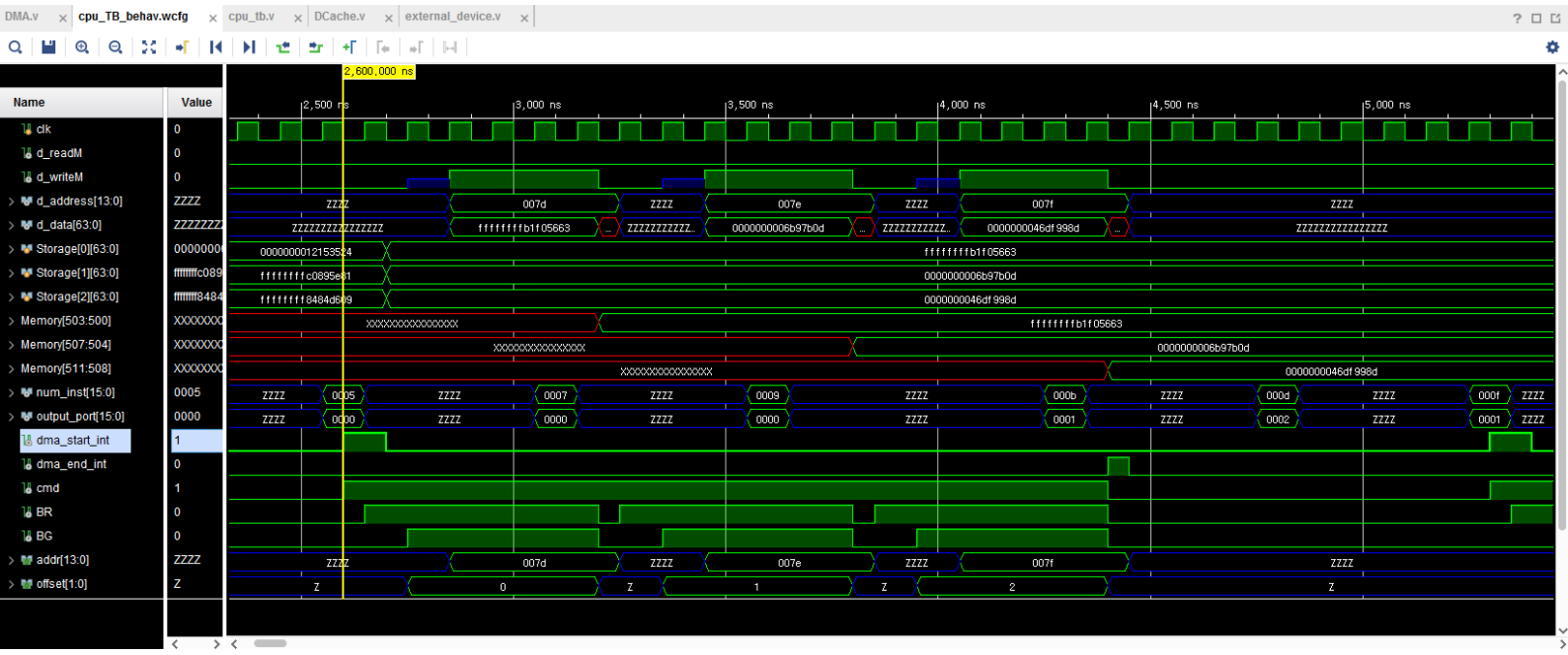


FIRE_TIME is 244,000. This is just before the execution of a LWD instruction that miss in the data cache. It first needs to evict the dirty cache line (since it has write-back scheme) and when fetch from memory. Since the DMA takes control of the bus, LWD waits and the pipeline stalls.

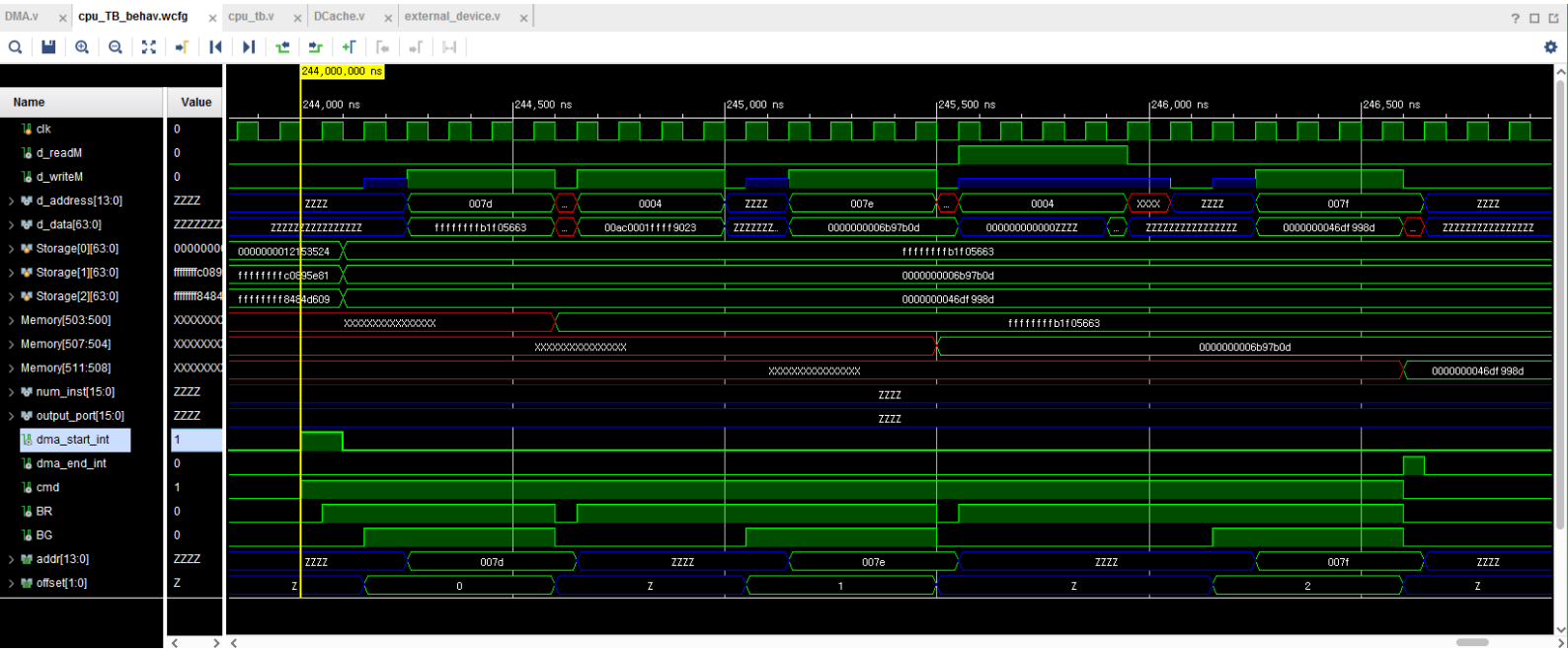


FIRE_TIME is 244,300. DMA interrupts the CPU during the execution of a LWD instruction. Notice that even if BR is asserted, bus is not granted until the eviction part of the LWD instruction finishes.

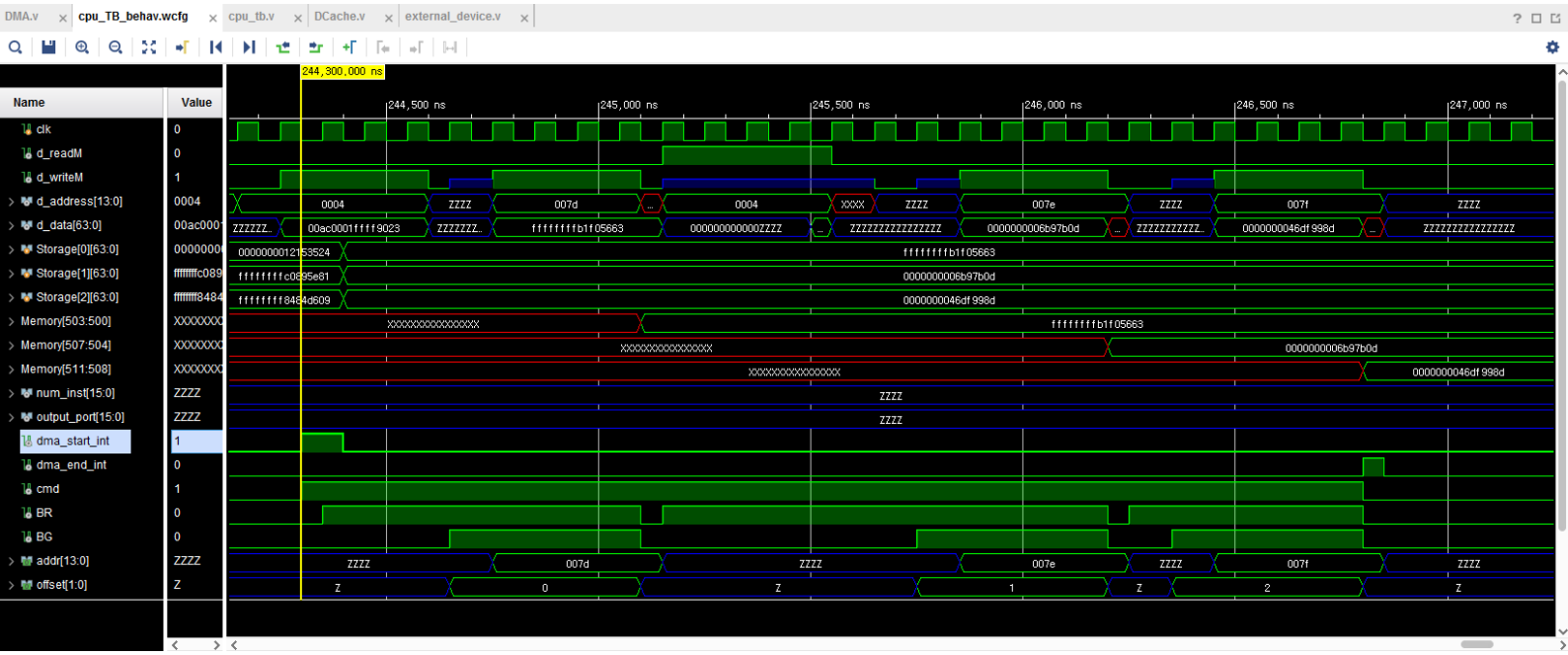
- Cycle-stealing model



FIRE_TIME is 2,600. After the write of one block completes, the DMA module deasserts the bus request signal (at the clock negative edge). This is re-asserted at the next cycle positive edge.



FIRE_TIME is 244,000. Memory accesses to address 0004 are from a LWD instruction. The DMA writes to block addresses 007d, 007e, and 007f. The execution sequence is: DMA block 1 -> LWD miss; eviction -> DMA block 2 -> LWD miss; fetch -> DMA block 3.



FIRE_TIME is 244,300. This time the execution sequence is: LWD miss; eviction -> DMA block 1 -> LWD miss; fetch -> DMA block 2 -> DMA block 3.

The total number of cycles for each model and experiment is shown on the right. The cycle-stealing scheme improves the total number of cycles when the DMA request overlaps with a memory access instruction, and does not when there are no overlaps. On the latter case, the total number of cycles is identical to the figure reported in lab 7.

FIRE_TIME	Baseline	Cycle Stealing
2600	2578	2578
244000	2590	2587
244300	2591	2583

5. Conclusion

I believe I have achieved every goal specified in the introduction.

DMA Controller

CPU :

If dma_start_int, DMACommand = 1.

If dma_end_int, DMACommand = 0

DataCache Busy signal asserted then.

DataCache : // During DMA (BG==1), memory access is blocked.

If BR and not d_readM and not d_writeM,

BG = 1. // set on posedge

d_readM = 1'bz.

d_writeM = 1'bz.

d_address_mem = 14'bz

d_data_mem = 64'bz

DMA :

DMA keeps block[1:0] register. It is the offset block number (from 0x1F4) to write to the memory. This can be sent directly to external_device with :

assign offset = BG ? block : 2'bz ;

If DMACommand, BR = 1. Else, BR = 0.

If BG, // data already arrived, since BG changed earlier.

addr = 14'h7D + block

data = edata

// counter logic same as data cache memory access.

When write is done :

If block == 2'b10 : block = 0 ; interrupt = 1.

↳ deasserted uncond-

Else : block ++ ;

reset counter. itionally next posedge