**CSED311 Lab Report #8**

20140658 최유정

20130221 이다훈

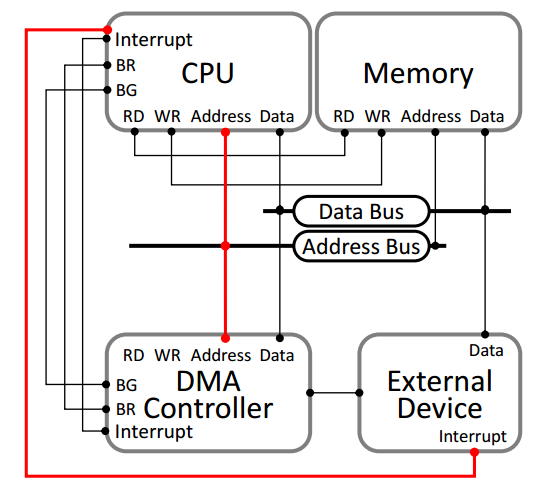
Introduction

DMA는 Direct Memory Access로 bus에 대해서 CPU의 grant를 받아서 external device가 DMA unit을 통해서 memory에 바로 access를 할 수 있도록 하는 것을 말한다. DMA를 하는 동안 CPU는 cache에 있는 data만으로 CPU 작업을 수행하도록 한다. DMA를 통해서 memory가 바로 external device와 data를 주고 받음으로써 성능을 더 높일 수 있다. 이번 lab의 목표는 이와 같은 역할을 해주는 DMA를 만드는 것인데, 그의 성능을 확인 하기 위해서 임의의 external device를 만들어서 그 data가 12word를 memory에 쓰려고 시도하는 것을 가정할 것이다.

Design

DMA를 포함하는 전체적인 그림은 [Figure 1]과 같다. CPU, Memory, DMA controller, External device가 있으며, 이 사이에 data bus, address bus가 존재한다.

현재 구현되어 있는 바로는 cpu\_TB에 모든 module이 존재하게 되고, 각 module로 들어가는 선들은 상황에 따라서 mux로 해당하는 address를 넣어줌으로써 data bus, address bus를 구현하였다.



[Figure 1] design of DMA system

Lab7인 Cache에서 변화된 부분은 DMA, external device가 추가되었다. External device에서 CPU로 가는 interrupt line을 연결했고, interrupt의 신호가 오는 event에 의해서 cpu에서 DMA에 address를 주는 dma address line에 의해서 external device가 memory에 저장할 수 있는 address를 받아 오게 된다. (interrupt 신호는 임의의 delay를 주었는데, 2000의 delay를 주었다.) 그리고 DMA와 CPU 사이에는 BR와 BG 선이 존재하여, CPU에 request를 날리고, cpu가 request를 수락하도록 되어 있다. 이후 BG가 high voltage 인 경우에 memory로 들어가는 address와 data를 mux에 의해서 음 data와 address로 주며, address는 DMA controller가, data는 매 cycle마다 external device에서 4words씩 전송이 되도록 되어 있다. 이때 external device에 매번 주소를 높이면서 data를 보내라는 신호를 음 controller가 주도록 되어있다. (현 lab에서는 길이가 지정되어 있기 때문에 fixed length, fixed time으로 설정했다.) 모든 data를 전송하고 난 후에는 BG, BR 모두 zero가 되도록 구현했다.

Implementation

총 6개의 파일로 구성되어있다.

cpu(Clk, Reset\_N, in\_readM, in\_writeM, i\_address, in\_data, da\_readM, da\_writeM, d\_address, da\_data, num\_inst, output\_port, is\_halted, in\_stall, da\_stall, IFID\_IR, read\_d\_data, BR, BG, dma\_begin, dma\_end, dma\_address, address\_set);

Data Path (Clk, next\_pc, i\_data, d\_data, is\_halted, ALUSrcB, ALUOp, MemtoReg, RegDst, RegWrite, ALUOut, PCOut, WWD, WriteData, sig\_d\_readM, sig\_d\_writeM, d\_readM, d\_writeM, num\_inst, flush, d\_addr);

ALU (A, B, Clk, OP, C, bcond, finish)

Forwarding (src, trg, EX\_RegWrite, EX\_MemtoReg, EX\_dst, MEM\_RegWrite, MEM\_MemtoReg, MEM\_dst, ALU1stMux, ALU2ndMux, Clk);

ID단계에서의 src, trg과 Ex, Mem stage의 dest Register를 비교하여 forwarding이 필요한지를 판별해 낸다.

Cache (clk, reset\_n, i\_readM, i\_writeM, i\_address, i\_data, d\_readM, d\_writeM, d\_address, d\_data, inst\_stall, data\_stall, i\_from\_M, d\_from\_M, i\_not\_yet, d\_not\_yet, RorW, read\_d\_data)

Cache 는 CPU (data-path)와 Memory사이에 존재하기 때문에 각 유닛과 주고받는 signal들의 중간 조율자 역할을 한다. 그 때문에 읽고 뱉어야 할 signal이 많았다.

external\_device(clk, interrupt, from\_DMA, e\_data, last\_data, e\_writeM);

외부 device이며, DMA로부터 신호를 받으면 data를 memory로 보낼 수 있도록 되어 있다.

DMA (clk, BG, BR, given\_addr, address\_set, dma\_addr, addr\_change, dma\_end, not\_yet, dma\_writeM);

DMA는 state가 있기 때문에 clk를 쓴다. BR, BG는 Bus request, Bus grant이고, given\_addr은 CPU로부터 받은 address이다. Address\_set은 BG와 동기화 되어있다. DMA\_addr은 memory unit으로 보낼 addr, add\_change는 external\_device에게 addr을 증가시키라는 신호이다. DMA\_end는 끝났다는 신호, not\_yet은 memory write가 진행중이라는 signal이다. DMA\_writeM은 memory에게 쓰라고 보내주는 signal이다.

Discussion

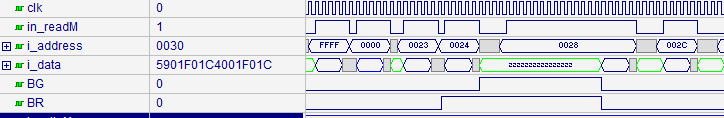
이번 lab에서는 12word 밖에 보내지 않기 때문에 cpu가 기다리는 시간이 많지 않았지만 만약 더 많은 data의 전송이 일어나야 하는 경우에 cpu가 지나치게 많이 기다려야 하는 부분이 있을 수 있을 것이다. 따라서 적정한 길이 이후 cpu의 grant가 다시 회수되는 모델인 extra 구현의 모델을 하면 성능이 더 좋아 질 것이라고 생각이 된다. 그리고 현 test bench의 경우에 loop와 같은 경우가 없어서 case가 없어서 cache의 내용이 아주 빈번하게 활용되는 경우는 없었는데 이런 경우에 또한 cpu가 memory 요청을 해야 하는데 이를 수행하지 못하기 때문에 이 또한 성능을 좋지 못하게 만드는 요인이 될 것이다.

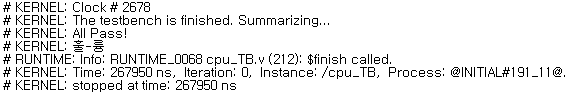
External device의 경우에는 DMA로부터 data를 보내야 한다는 신호가 오면 적당히 address를 늘려가면서 data를 전송하는 device이므로 많은 구현이 필요 없었으며, cpu\_TB에서 적당히 bus를 잘 연결해주었기 때문에 처리가 잘 되었다. 만약 external device가 여럿이었다면 각각의 external device에 해당하는 data와 address를 주어야 하기 때문에 조금 더 복잡한 구현이 필요했을 지도 모른다.

실제적인 DMA를 생각한다면 첫 번째 extra implementation은 오히려 기본적인 구현이어야 한다고 생각한다. 물론 시간이 없어서 구현하지 않은 것이다. 내년에는 이 점이 반영되어 default 가 수정되었으면 한다.

Conclusion

1. ALL PASS
2. NO EXTRA CREDIT





(BG가 있는 동안에는 instruction을 읽으려고 시도해도 읽을 수가 없도록 되어 있다.)