Lab 7: Direct Memory Access

20190084 권민재, 20190335 양승원 `CSED311`

Introduction

이번 과제에서는 CPU에 장착된 외부 장치가 메모리에 직접 데이터를 쓸 수 있도록 DMA를 구현하는 것이 목표이다. 이것은 인터럽트와 데이터 버스를 이용한 데이터 전송을 포함하며, CPU가 버스를 arbitrating 할 수 있어야 한다.

Design

CPU

Interrupt

CPU가 인터럽트 명령어를 받았을 때, 만약 현재 처리 중인 메모리 접근이 있다면 그를 해결한 후에 interrupt에 진입하며, 아니라면 곧장 interrupt에 진입한다. Interrupt 중에는 파이프라인 레지스터의 포워딩을 막아서 CPU의 작동을 중단시키고, interrtupt가 해결되었다는 신호를 외부에 보내서 interrupt를 종료하도록 설계하였다.

Arbitrating

CPU가 외부 장치로부터 bus의 사용을 요청받으면, 현재 CPU가 버스를 사용 중인지 확인한 후, 사용 중이지 않다면 그 즉시 외부 장치에게 bus를 grant하며, 아니라면 CPU가 버스 사용을 종료할 때 까지 대기한 후에 bus를 grant 한다. Request가 종료되면 그 즉시 CPU가 버스를 사용하도록 설계하였다. 이것은 추가 기능 구현을 위해 필수적인 사항이다. 이때, CPU에 버스가 granted 되지 않으면 캐시의 데이터만을 이용하도록 설계하였다.

DMA Controller

Interrupt

DMA를 통한 writing이 모두 종료되었을 때, DMA end 인터럽트명령을 CPU에게 전송하여 interrupt를 일으키도록 디자인하였다.

DMA

DMA의 첫 단계에서 CPU로부터 전송되는 writing 할 주소와 데이터의 길이를 DMA 컨트롤러가 저장한 후에, DMA를 통해 메모리에 값을 쓸 때 address bus에 컨트롤러가 주소를 보내고, external device에게는 현재 적어야할 데이터 주소의 offset을 알려주는 방식으로 디자인하였다.

Bus

DMA start interrupt가 종료된 이후 바로 bus를 request 하며, DMA 작업이 종료되었을 때 bus request를 중지하는 방식으로 설계하였다.

External Device

Interrupt

External Device는 200 클럭까지 대기하고 CPU에 DMA start interrupt를 전송할 수 있도록 설계하였다.

DMA

DMA가 실행 중일 때에는, DMA controller로 부터 전송받은 offset에 따라 자신이 가지고 있는 데이터 중 4 words를 데이터 버스에 전송한다.

Implementation

CPU

Interrupt

CPU에게 interrupt가 주어지면 메모리의 read, write 신호를 확인해서 busy인지 그 여부를 결정한 후에, busy 하지 않다면 interrupt 여부와 그 명령어 종류를 CPU에 저장한다. 이후, interrupt 명령어가 DMA start 였다면 DMA controler에게 주소와 적어야 할 데이터의 길이를 전송하고 interrupt를 해결하게 하였다. Interrupt 명령어가 DMA end 였다면, 그 때 동시에 CPU의 arbiter 로직에서 DMAC의 bus grant를 종료시키도록 구현하였기 때문에 별도의 처리 없이 interrupt를 바로 종료하도록 구현하였다.

Arbitrating

CPU는 bus request가 들어왔을 경우에 버스의 ready 여부에 따라서 요청을 grant 해주는 방식으로 구현하였다. CPU에게 bus가 grant 되지 않았을 때에 cache는 ready 되어 있고 memory는 ready 되어 있지 않은 상태로 시그널을 준비하여 CPU가 캐시의 데이터에만 접근할 수 있도록 구현하였다.

DMA Controller

Interrupt

DMA controller는 현재 가리키고 있는 오프셋이 마지막 오프셋일 때 메모리의 기록이 끝난다면 DMA end interrupt를 발생시킨다.

DMA

DMA controller는 CPU로부터 전송된 주소와 길이에 따라서 DMA를 관리한다. DMA는 bus가 granted 된 상태일 때에만 메모리에 접근하여 값을 쓰도록 하였는데, 4 words를 전송한 이후에는 bus request를 끊는 방식으로 추가 구현을 하였다. DMA controller는 bus request가 끊긴 그 직후 클럭에서 bus를 다시 requests하며, 이것은 CPU의 arbitrating에 따라서 CPU가 메모리를 사용하고 있지 않을 때에 비로소 DMA controller에게 bus가 다시 grant 되기 때문에 이 과정을 통해 1번 추가 구현이 동작함을 알 수 있다. README.txt에서 보였듯이, 220 사이클에서 DMA Controller로부터 bus request가 다시 시작되었지만, CPU arbitrating으로 CPU의 메모리 접근이 먼저 수행된 뒤 BG 시그널이 가는 것을 볼 수 있다.

External Device

Interrupt

외부 디바이스는 num_clk이 일정 수준이 되었을 때에 DMA start interrupt 명령어를 CPU에 전송하며, CPU가 해결되었다는 신호를 보냈을 때에 interrupt 명령어의 전송을 중지하도록 구현하였다.

DMA

외부 디바이스는 DMA controller로부터 지금 적어야할 데이터의 offset을 전달받고, DMA controller에게 bus가 grant 된 상태일 때에만 전달받은 offset으로부터 4 words를 데이터 버스에 전송하도록 설계하였다.

Discussion

- 처음에 구현할 때에는 DMA controller가 CPU에게 bus를 요청하면 그 다음 클럭에 bus를 grant해주었는데, 이와 같은 방식은 매우 비효율적이었기 때문에 메모리가 busy하지 않다면 request 즉시 bus를 grant 할 수 있도록 구현하였다.
- CPU의 요청에 의한 메모리의 ack와 DMA의 요청에 의한 메모리의 ack 시그널을 구별하기 위해 고민하였고, CPU의 grant 신호를 활용하여 그를 구분할 수 있었다.
- DMA를 진행 하는 동안 CPU는 cache에만 접근할 수 있기 때문에, cache의 효율이 좋을 수록 DMA의 효율이 올라갈 것으로 생각된다.
- 1번 추가 구현 사항을 구현하였더니, DMA 중에도 CPU가 메모리 접근을 충분히 할 수 있었다. External Device가 상당히 큰 데이터를 전송하고자 할 때 등에는 1번 추가 구현 사항이 없다면 상당히 큰 CPU 성능 저하가 발생할 것 같다.

Conclusion

이번 과제를 통해 CPU의 관여 없이 외부 장치가 메모리에 직접 접근할 수 있도록 하는 것이 매우 효율적이라는 것을 알 수 있었다.