Hardware System Design 4190.309A
Seoul National University

Practice #11. DMA (Direct Memory Access)
Jiwon Lee, Sangjun Son

Goal

- Compare data transfer using CDMA vs transferring data through CPU.
- Compare performance between CDMA, conventional data transferring through CPU using `gettimeofday()`.

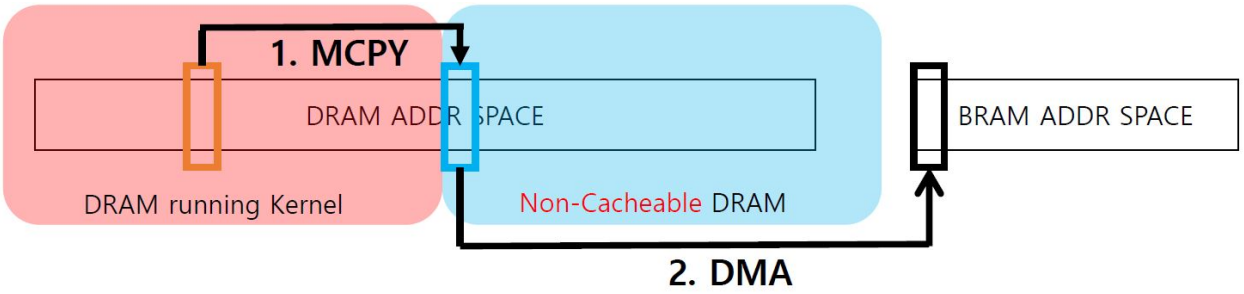

Figure 1: Memcopy and DMA Process with BRAM [1].

1 Implementation

이번 랩에서는 Xilinx AXI CDMA(Central Direct Memory Accesss)를 이용하여 DMA를 수행해보고, 기존의 CPU를 통해 데이터를 쓰는 과정과 성능 면에서의 차이를 확인해본다. CDMA는 DRAM에서 BRAM(SRAM)으로 데이터를 쓰고, memcopy는 DRAM에서 DRAM으로 데이터를 쓴다. 하지만, CDMA는 CPU를 거치지 않기 때문에, memcopy보다 수행시간 측면에서 더 효율적이라는 것을 예상해 볼 수 있다. 또한, memcoppy와 memory load의 수행시간을 측정하여, 총 3가지 wrdata과정을 비교한다.

결론적으로, 이번 실습에서 비교해야 하는 Operation은 다음의 세 가지이다.

- CDMA
- `memcpy()` in C
- Memory load

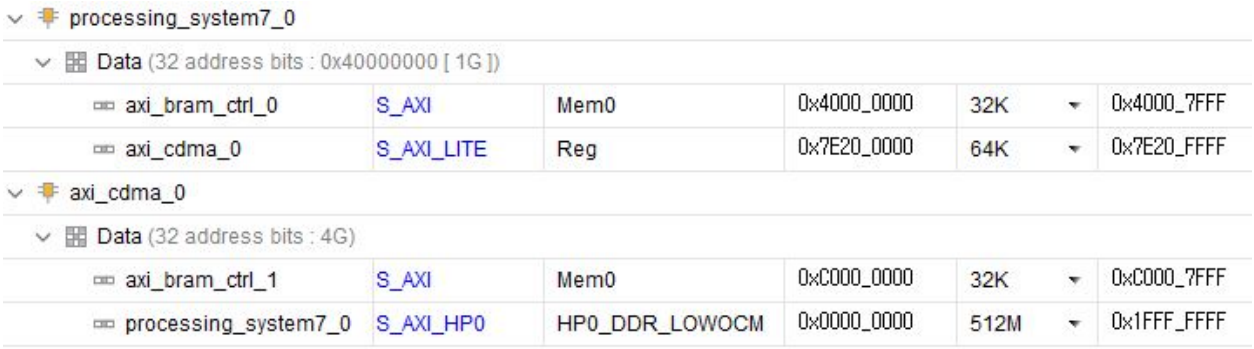

Figure 2: Address of block elements

Practice #11. DMA (Direct Memory Access)

Jiwon Lee, Sangjun Son

아래는 보드에 매핑된 block element들의 주소와 이번 실습에서 performance 비교를 위한 C코드이다.

CDMA

```
1  struct timeval st[4];
2
3  // DMA : DRAM -> BRAM
4  unsigned int *fpga_dma = mmap(NULL, 16*sizeof(unsigned int), PROT_READ|PROT_WRITE, MAP_SHARED, foo, 0x7E200000);
5  gettimeofday (&st[2], NULL);
6  *(fpga_dma+6) = 0x10000000; // SA
7  *(fpga_dma+8) = 0xC0000000; // DA
8  *(fpga_dma+10) = SIZE * (SIZE + 1) * sizeof(float); // BTT
9  while ((*(fpga_dma+1) & 0x00000002) == 0); // CDMASR.IDLE
10 gettimeofday (&st[3], NULL);
```

- `mmap()` 함수를 통해, bitstream에서 미리 할당된 CDMA address(0x7E200000 - 0x7E20FFFF)를 메모리(fpga_dma)에 대응시킨다(4번째 줄).
- CDMA IP에 올바른 input값들 (source addr, dest addr, bytes to trnasfer)을 할당하고, while문을 통해 DMA operation이 종료될 때까지 기다린다(6-8번째 줄).
- `gettimeofday()`함수를 통해 수행시간을 측정한다(5, 10번째 줄).

About Register Address Map of CDMA

아래는 AXI CDMA register mapping table [2]을 나타낸다. 이 표를 통해, CDMA를 수행하는 c코드에서 source address, destination address, bit to transfer, AXI CDMA state의 address offset을 확인할 수 있다.

Address Space Offset[(1)]	Name	Description
00h	CDMACR	CDMA Control
04h	CDMASR	CDMA Status
08h	CURDESC_PNTR	Current Descriptor Pointer
0Ch[(2)]	CURDESC_PNTR_MSB	Current Descriptor Pointer. MSB 32 bits. Applicable only when the address space is greater than 32.
10h	TAILDESC_PNTR	Tail Descriptor Pointer
14h[(2)]	TAILDESC_PNTR_MSB	Tail Descriptor Pointer. MSB 32 bits. Applicable only when the address space is greater than 32.
18h	SA	Source Address
1Ch[(2)]	SA_MSB	Source Address. MSB 32 bits. Applicable only when the address space is greater than 32.
20h	DA	Destination Address
24h[(2)]	DA_MSB	Destination Address. MSB 32 bits. Applicable only when the address space is greater than 32.

Figure 3: AXI CDMA Register Address map

Address Space Offset[1]	Name	Description
28h	BTT	Bytes to Transfer

Figure 4: AXI CDMA Register Address map (cont.)

위의 CDMA subsection 코드의 4-8번째 줄에서, *fpga_dma는 integer pointer이므로, 4byte의 크기를 갖는다. 따라서, fpga_dma+6은 fpga_dma로부터 24(=0x18)의 Address Space Offset을 가지고, 이는 위의 표에서 확인할 수 있듯, SA register를 가리킨다. 마찬가지로, fpga_dma+8은 32(=0x20)의 Address Space Offset을 가지고, 이는 DA register를 가리킨다. fpga_dma+10은 40(=0x28)의 Address Space Offset을 가지고, 이는 BTT(Bytes To Transfer) register를 가리킴을 확인할 수 있다.

Memcopy & Memory load, write

```
1  struct timeval st[4];
2
3  // memory load
4  int foo;
5  foo = open("/dev/mem", O_RDWR);
6  float *ps_dram = mmap(NULL, (SIZE * (SIZE + 1))* sizeof(float), PROT_READ|PROT_WRITE, MAP_SHARED, foo, 0x10000000);
7
8  // MCPY: DRAM -> DRAM
9  gettimeofday (&st[0], NULL);
10 for (i = 0; i < SIZE * (SIZE + 1); i++)
11     *(ps_dram + i) = flat[i];
12 gettimeofday (&st[1], NULL);
13 memcpy( ps_dram, flat, SIZE * (SIZE + 1) * sizeof(float) );
14 gettimeofday (&st[2], NULL);
```

- `mmap()` 함수를 통해, bitstream에서 미리 할당된 DRAM address(0x00000000 - 0x0000FFFF)를 메모리(ps_dram)에 대응시킨다 (6번째 줄).
- for문을 통해 memory load를 수행한다 (10-11번째 줄).
- `memcpy()` 함수를 통해 memory copy를 수행한다 (13번째 줄).
- `gettimeofday()` 함수를 통해 수행시간을 측정한다 (9, 12, 14번째 줄).

memcpy와 for문을 이용한 memory load에서 memcpy가 더 좋은 수행시간을 가질 것이라 예상해 볼 수 있는데, 그 이유는 memcpy는 byte포인터가 아닌 word pointer를 이용하여 메모리 복사를 수행하고, SIMD (single instruction, multiple data)를 이용하여 assembly instruction을 수행하기 때문이다.

2 Result

```
The number of mismatch (DMA dram <-> bram) : 0
The number of mismatch (CPU dram <-> bram) : 0
MCPY1: 539
MCPY2: 152
DMA  : 70
CPU  : 470
```

Figure 5: `for-loop`(*MCPY1*)/`memcpy`(*MCPY2*) 비교, DMA/CPU memory load의 수행시간 (micro second) 비교: DMA를 사용하여 non-cacheable memory에서 BRAM으로 데이터를 전달하는 시간이 CPU의 내장함수 `memcpy` 보다 훨씬 (약 7배) 적은 시간 이 걸렸다.

주어진 main.c코드를 zedboard위에서 수행하였을 때, 수행속도는 DMA, memcpy (MCPY2), memory load (MCPY1)순으로 빠르게 나타난 것을 확인할 수 있었다. 실험을 진행하기 전 예상과 일치하였는데, 이는 DMA는 CPU를 거치지 않고 바로 memory에 접근하여 copy하여, c코드를 통해 구현한 memcpy, memory load보다 속도 측면에서 더 효율적이었다. memcpy, memory load는 위에서 언급했듯, memcpy의 구현이 for loop를 통한 memory load보다 더 효율적이기 때문에 속도 측면에서 차이가 난 것이라 생각하였다.

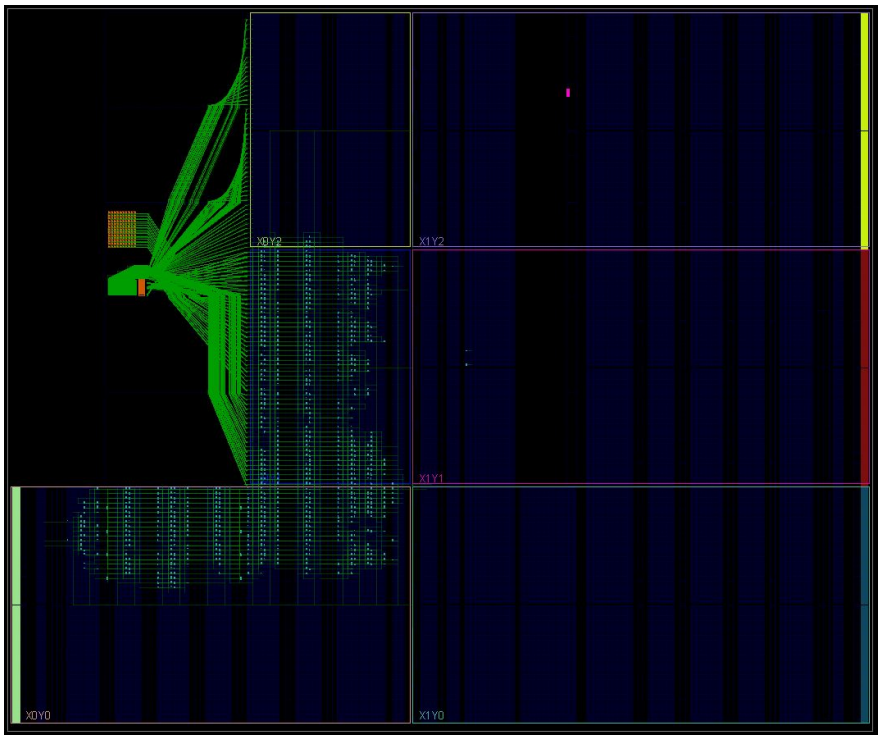

Figure 6: DMA 모듈을 함께 넣은 하드웨어 Implementation 모습이다.

3 Conclusion

이번 실습에서는 DMA를 사용해 수행시간을 일반적으로 사용하는 memory copy, load와 비교해 보았다. 우리는 최종적으로 CNN을 위한 Matrix-Matrix multiplication을 구현한다. 여기서, DMA를 사용한 memory load의 속도 개선은 결과적으로 training, inference time의 속도 개선을 의미한다. 따라서, 이번 실습을 토대로 최종 프로젝트의 optimization의 힌트를 얻을 수 있었다.

References

[1] Computing Memory Architecture Lab. *Practice 11: DMA (Direct Memory Access)*. Hardware System Design, May 2021.

[2] Xilinx. *AXI Central Direct Memory Access v4.1*. Xilinx, 2018.