

4190.301A Hardware System Design

Spring 2020

Hardware System Design Lab9 Report

Kim Bochang

2014-16757

1. <lab9> Introduce

Lab 9의 목적은 주어진 zedboard의 SD 카드 부팅 모드를 이용해서, 보드의 자체 OS를 통해 보드를 실행 하는법을 익히는 것이다.

2. Implementation

2.1 main.c

튜토리얼과 관련된 내용은 그대로 따라하면 되므로 생략하였다.

이번 랩에서는 implementation 할만한 것은 없고, hsd20_lab09_practice 프로젝트의 비트스트림과 main.c 코드를 실행해보고 어떤 동작이 일어나는지 예측하는 것이다.

main.c의 원본 코드는 아래와 같다.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>

#define SIZE 4

int main(int argc, char** argv)
{
    int i;

    int foo = open("/dev/mem", O_RDWR);
    int *fpga_bram = mmap(NULL, SIZE * sizeof(int), PROT_READ|PROT_WRITE, MAP_SHARED, foo, 0x40000000);
    int *fpga_ip = mmap(NULL, sizeof(int), PROT_READ|PROT_WRITE, MAP_SHARED, foo, 0x43C00000);

    // initialize memory
    for (i = 0; i < SIZE; i++)
        *(fpga_bram + i) = i * 2;
    for (i = SIZE; i < SIZE * 2; i++)
        *(fpga_bram + i) = 0;

    printf("%-10s%-10s\n", "addr", "FPGA(hex)");
    for (i = 0; i < SIZE * 2; i++)
        printf("%-10d%-10X\n", i, *(fpga_bram + i));

    // run ip
    *(fpga_ip) = 0x5555;
    while (*(fpga_ip) == 0x5555);

    printf("%-10s%-10s\n", "addr", "FPGA(hex)");
    for (i = 0; i < SIZE * 2; i++)
        printf("%-10d%-10X\n", i, *(fpga_bram + i));

    return 0;
}
```

동작을 분석해보면, 먼저 메모리의 bram영역과 ip영역을 mmap함수를 이용하여, 각각 size * 4바이트, 4바이트 만큼을 각각 fpga_bram과 fpga_ip 포인터를 통해 참조할 수 있도록 가상 메모리 영역을 지정해준다.

이때, 읽기와 쓰기가 모두 가능한상태로, shared 메모리로 지정해주어 모든 프로세스에서 사용할 수 있도록 한다.

그리고 나서 bram 영역의 SIZE 만큼의 영역을 순서대로 0, 2, 4, 6과 같이 초기화 하고, 그 다음 SIZE만큼의 영역을 0으로 초기화 한다. (int *형 포인터라, 0.0f와 같이 float형 데이터를 대입하려고 하면 자동으로 int형으로 형변환이 일어나게 된다. 이때, 0.0은 0으로 형변환이 일어난다. 1.5f와 같은 값은 0.5를 버리고 1로 변환된다.)

그 후, BRAM의 SIZE * 2만큼의 영역에 무엇이 저장되어 있었는지 프린트 해주고,

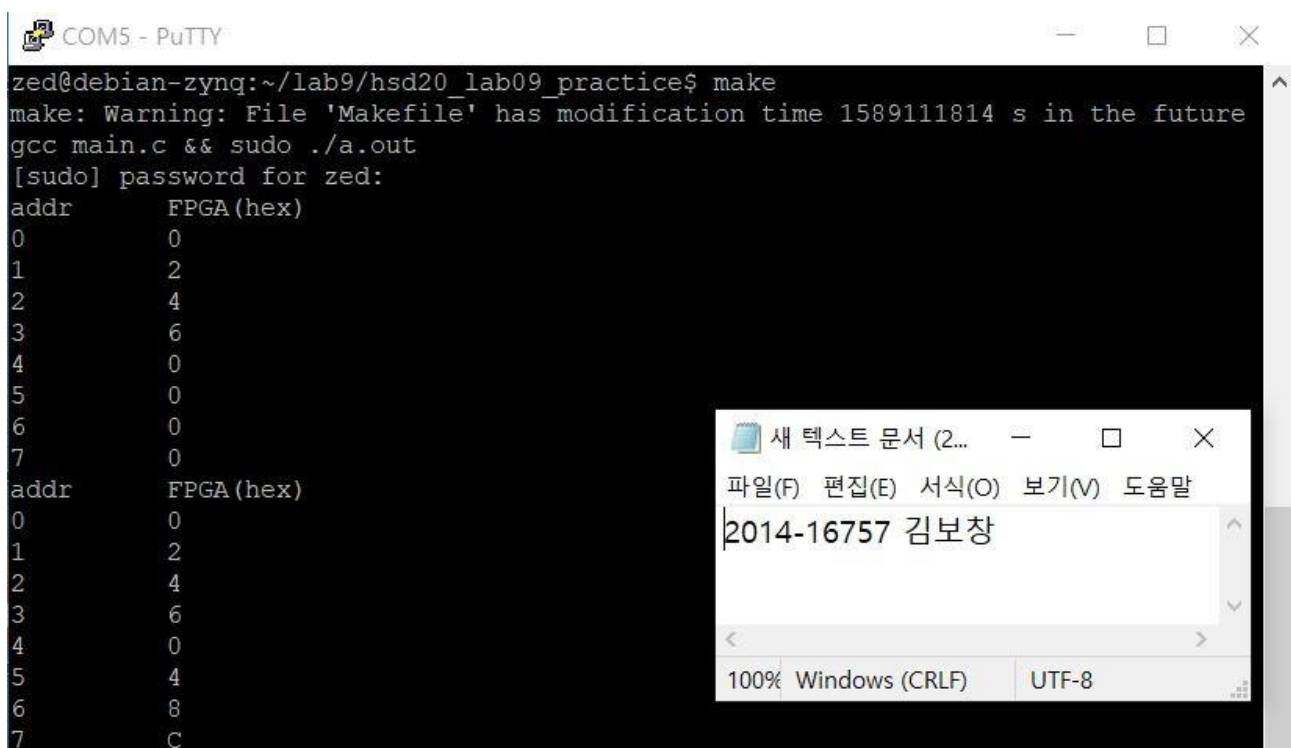
ip영역에 0x5555의 값을 대입 한 후, FPGA의 실행이 끝날 때까지 기다린 후에

BRAM의 SIZE * 2만큼의 영역에 무엇이 저장되어 있는지 프린트해준다.

3. Result & Discussion

먼저, tutorial에서 이용한 zync.bit 파일은 같은 폴더에 첨부되어 있다.

위 코드를 실행한 결과는 다음과 같다.

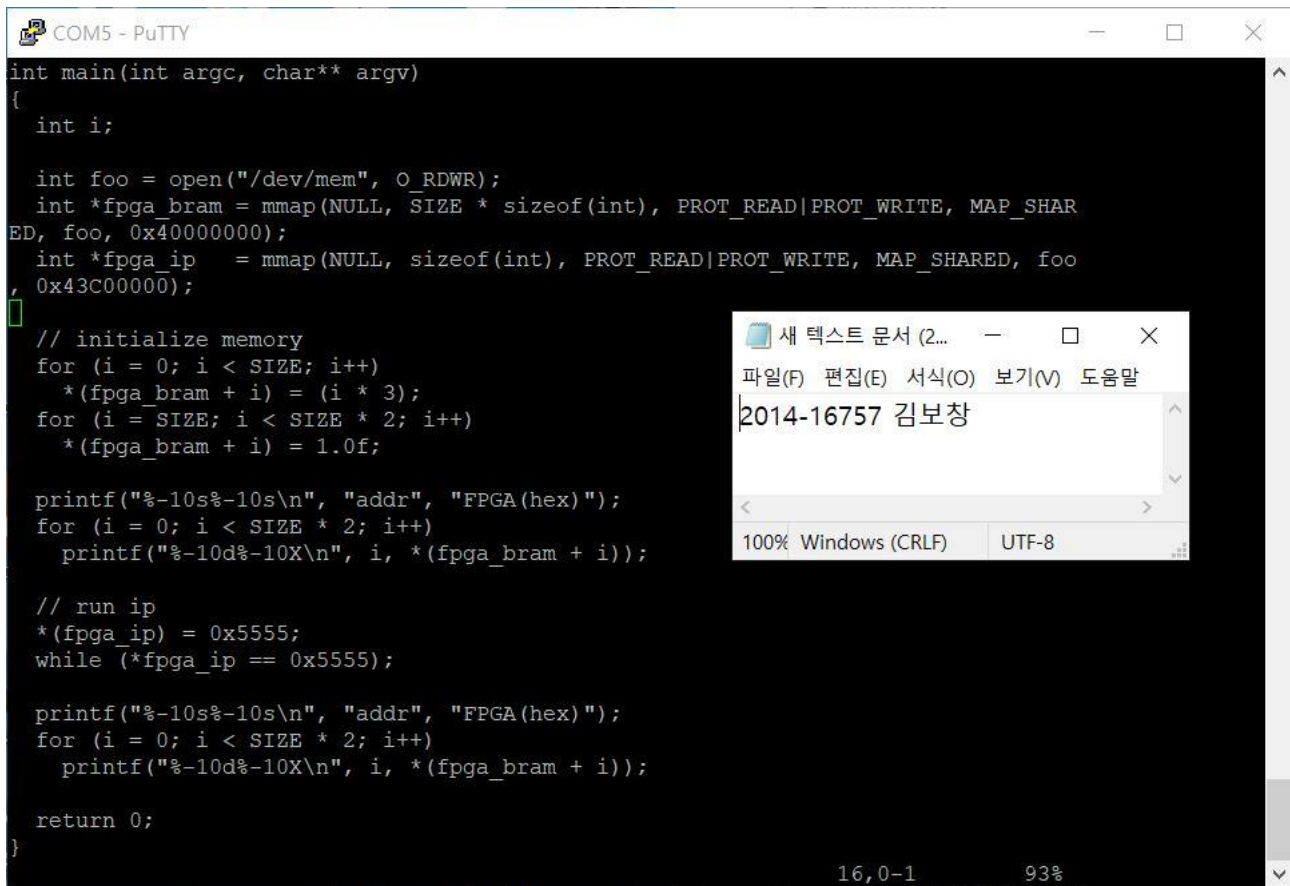


```
zed@debian-zynq:~/lab9/hsd20_lab09_practice$ make
make: Warning: File 'Makefile' has modification time 1589111814 s in the future
gcc main.c && sudo ./a.out
[sudo] password for zed:
addr      FPGA (hex)
0          0
1          2
2          4
3          6
4          0
5          0
6          0
7          0
addr      FPGA (hex)
0          0
1          2
2          4
3          6
4          0
5          4
6          8
7          C
```

새 텍스트 문서 (2...
파일(F) 편집(E) 서식(O) 보기(V) 도움말
2014-16757 김보창
100% Windows (CRLF) UTF-8

FPGA의 실행이 끝난 이후, 0~3번지에 저장된 값이 4~7번지에 2배씩 되어 저장된것을 알 수 있다.

이러한 작동을 정확히 알아보기위해, 다음과 같이 main.c의 코드를 고쳐보았다.



```
COM5 - PuTTY
int main(int argc, char** argv)
{
    int i;

    int foo = open("/dev/mem", O_RDWR);
    int *fpga_bram = mmap(NULL, SIZE * sizeof(int), PROT_READ|PROT_WRITE, MAP_SHARED, foo, 0x40000000);
    int *fpga_ip = mmap(NULL, sizeof(int), PROT_READ|PROT_WRITE, MAP_SHARED, foo, 0x43C00000);

    // initialize memory
    for (i = 0; i < SIZE; i++)
        *(fpga_bram + i) = (i * 3);
    for (i = SIZE; i < SIZE * 2; i++)
        *(fpga_bram + i) = 1.0f;

    printf("%-10s%-10s\n", "addr", "FPGA(hex)");
    for (i = 0; i < SIZE * 2; i++)
        printf("%-10d%-10X\n", i, *(fpga_bram + i));

    // run ip
    *(fpga_ip) = 0x5555;
    while (*(fpga_ip) == 0x5555);

    printf("%-10s%-10s\n", "addr", "FPGA(hex)");
    for (i = 0; i < SIZE * 2; i++)
        printf("%-10d%-10X\n", i, *(fpga_bram + i));

    return 0;
}
```

새 텍스트 문서 (2...
파일(F) 편집(E) 서식(O) 보기(V) 도움말
2014-16757 김보창
100% Windows (CRLF) UTF-8

16, 0-1 93%

먼저, SIZE는 그대로 유지하고, 초기에 0~3번지에 저장될 값을 $i * 3$ 이 되도록 해보았다.

실행 결과는 아래와 같다.

```
COM5 - PuTTY
zed@debian-zynq:~/lab9/hsd20_lab09_practice$ ls
Makefile a.out main.c zynq.bit
zed@debian-zynq:~/lab9/hsd20_lab09_practice$ make
gcc main.c && sudo ./a.out
[sudo] password for zed:
addr      FPGA(hex)
0          0
1          3
2          6
3          9
4          1
5          1
6          1
7          1
addr      FPGA(hex)
0          0
1          3
2          6
3          9
4          0
5          6
6          C
7          12
zed@debian-zynq:~/lab9/hsd20_lab09_practice$
```

마찬가지로, 0~3번지에 저장된 값이 2배가 되어 4~7번지에 저장되었음을 알 수 있다.

여기서 한발 더 나아가, SIZE를 6으로 조정하고, 다른 값들도 바꾸어 보았다.

```
COM5 - PuTTY
#define SIZE 6

int main(int argc, char** argv)
{
    int i;

    int foo = open("/dev/mem", O_RDWR);
    int *fpga_bram = mmap(NULL, SIZE * sizeof(int), PROT_READ|PROT_WRITE, MAP_SHARED, foo, 0x40000000);
    int *fpga_ip   = mmap(NULL, sizeof(int), PROT_READ|PROT_WRITE, MAP_SHARED, foo, 0x43C00000);

    // initialize memory
    for (i = 0; i < SIZE; i++)
        *(fpga_bram + i) = (i * 4);
    for (i = SIZE; i < SIZE * 2; i++)
        *(fpga_bram + i) = 2.0f;

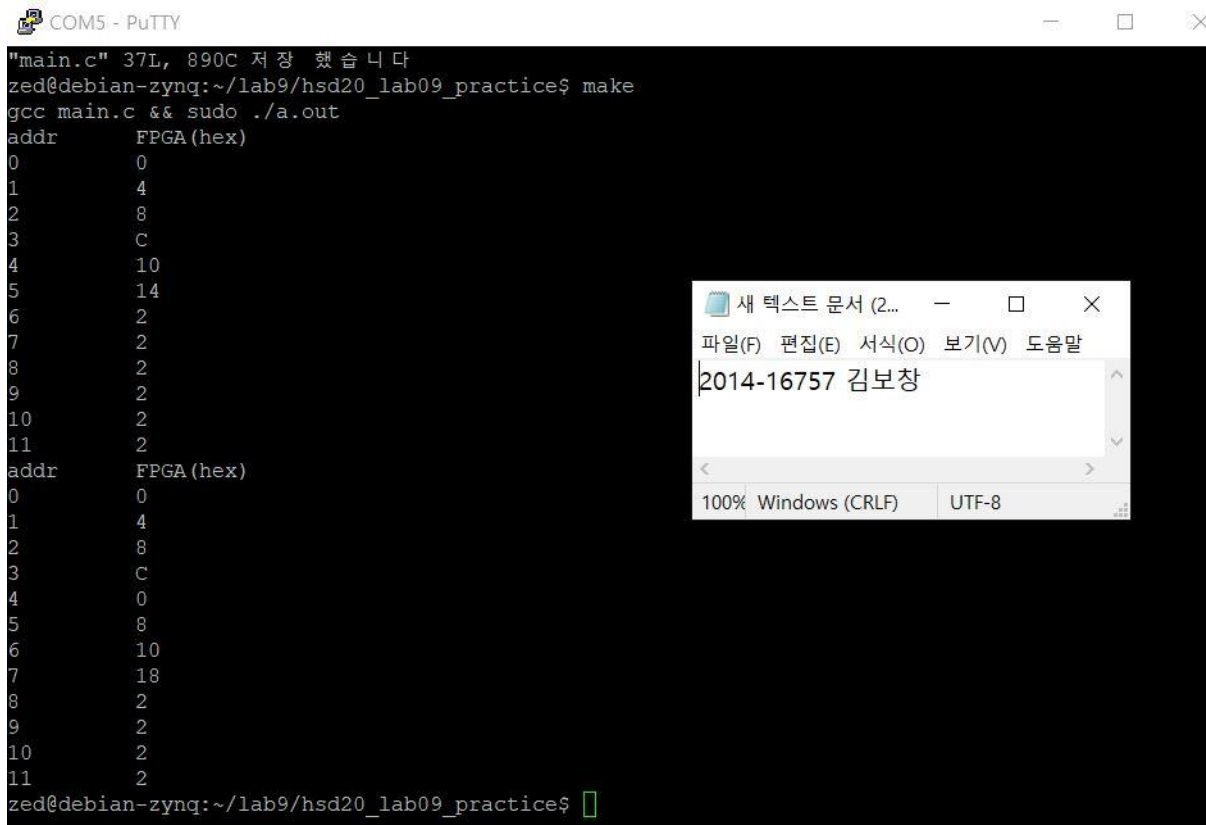
    printf("%-10s%-10s\n", "addr", "FPGA(hex)");
    for (i = 0; i < SIZE * 2; i++)
        printf("%-10d%-10X\n", i, *(fpga_bram + i));

    // run ip
}
```

0~5번지에는 $i * 4$ 의 값을, 6~11번지에는 2의 값이 (2.0f는 자동으로 int로 형변환되어 2가 됨)

저장되도록 하였다.

실행 결과는 아래와 같다.



```
COM5 - PuTTY
"main.c" 37L, 890C 저장 했습니다
zed@debian-zynq:~/lab9/hsd20_lab09_practice$ make
gcc main.c && sudo ./a.out
addr      FPGA (hex)
0          0
1          4
2          8
3          C
4          10
5          14
6          2
7          2
8          2
9          2
10         2
11         2
addr      FPGA (hex)
0          0
1          4
2          8
3          C
4          0
5          8
6          10
7          18
8          2
9          2
10         2
11         2
zed@debian-zynq:~/lab9/hsd20_lab09_practice$
```

새 텍스트 문서 (2...
파일(F) 편집(E) 서식(O) 보기(V) 도움말
2014-16757 김보창
100% Windows (CRLF) UTF-8

전과 같이, 0~3번지에 저장된 값이 2배가 되어 4~7번지에 저장되고, 나머지 값들은 변하지 않음을 알 수 있다.

마지막으로, 0~5번지에 저장되는 값들을 1, 6~11번지에 저장되는 값들을 $i*2$ 로 하여 나의 가설이 맞는지를 체크해 보았다.

```

#define SIZE 6

int main(int argc, char** argv)
{
    int i;

    int foo = open("/dev/mem", O_RDWR);
    int *fpga_bram = mmap(NULL, SIZE * sizeof(int), PROT_READ|PROT_WRITE, MAP_SHARED,
    ED, foo, 0x40000000);
    int *fpga_ip = mmap(NULL, sizeof(int), PROT_READ|PROT_WRITE, MAP_SHARED, foo,
    , 0x43C00000);

    // initialize memory
    for (i = 0; i < SIZE; i++)
        *(fpga_bram + i) = (int) 1.5f;
    for (i = SIZE; i < SIZE * 2; i++)
        *(fpga_bram + i) = i * 2;

    printf("%-10s%-10s\n", "addr", "FPGA(hex)");
    for (i = 0; i < SIZE * 2; i++)
        printf("%-10d%-10X\n", i, *(fpga_bram + i));

    // run ip

```



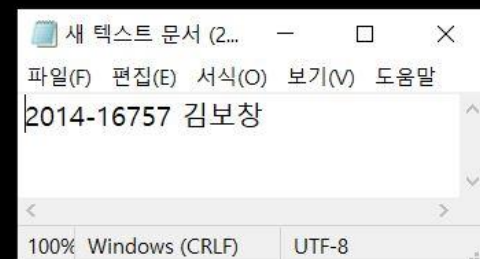
1.5f는 int형으로 형변환되어 1이 된다.

실행 결과는 아래와 같다.

```

zed@debian-zynq:~/lab9/hsd20_lab09_practice$ make
gcc main.c && sudo ./a.out
addr      FPGA (hex)
0          1
1          1
2          1
3          1
4          1
5          1
6          C
7          E
8         10
9         12
10        14
11        16
addr      FPGA (hex)
0          1
1          1
2          1
3          1
4          2
5          2
6          2
7          2
8         10
9         12
10        14
11        16
zed@debian-zynq:~/lab9/hsd20_lab09_practice$

```



정확히 4~7번지에 2배가 된 값이 들어감을 알 수 있다.

결론적으로, FPGA에 0x5555를 전달했을 때, 이는 BRAM의 0~3번지에 저장된 integer 값들을 정확히 두배 해서 BRAM의 4~7번지에 저장함을 알 수 있었다.

4. Conclusion

보드 내부에 설치된 OS를 이용하여 보드를 독립된 컴퓨터 처럼 이용하는 기능을 처음으로 사용해 본 실습이었다. C 코드를 보드를 이용하여 컴파일하고 실행까지 하면서, 제드보드 자체가 하나의 컴퓨터와 같다는 것을 느낄 수 있었다. 또한, 제드보드에 구운 회로를 mmap을 이용한 가상메모리 매핑으로 간편하게 이용할 수 있다는 것이 인상 깊었다.

앞으로의 프로젝트가 어떻게 진행될지 기대되게 만들어주는 실습이었다.