# TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – GJ (박현우)
uc820@naver.com

# 1. 시스템 프로그래밍 – 8 (Thread)

```c
#include<stdio.h>
#include<pthread.h>

void *task1(void *X){
    printf("Thread A Complete\n");
}
void *task2(void *X){
    printf("Thread B Complete\n");
}

int main(void){

    pthread_t ThreadA, ThreadB;

    pthread_create(&ThreadA, NULL, task1, NULL);  // task1은 ThreadA가 구동 형태
    pthread_create(&ThreadB, NULL, task2, NULL);  // task2은 ThreadB가 구동 형태
                                                  // 여기까지는 형태만 만듦.
    pthread_join(ThreadA,NULL); // join으로 구동
    pthread_join(ThreadB,NULL);

    return 0;
}
```

- Thread

Process는 독립적이고
Thread는 종속적이다.

Pthread_creat로 형태만 만들고

Join으로 thread를 구동시킨다.

# 1. 시스템 프로그래밍 – 8 (프로세스간 signal 통신)

```c
#include<stdio.h>
#include<signal.h>
#include<unistd.h>
#include<stdlib.h>
void gogogo(int voidv){

    printf("SIGINT Accur\n");
    exit(0);
}

int main(void){

    signal(SIGINT, gogogo);

    for(;;){
        printf("kill Test\n");
        sleep(2);
    }

    return 0;
}
```

```c
#include<stdio.h>
#include<signal.h>
#include<unistd.h>
#include<stdlib.h>

int main(int argc, char *argv[]){  // 프로세스 간 시그널 주고 받기.

    if(argc <2 )
        printf("Usage : ./exe pid\n");
    else
        kill(atoi(argv[1]), SIGINT);

    return 0;
}
```

- 프로세스간 signal 주고 받기

왼쪽 첫 번째 예제의 pid값을 찾아

두 번째 예제를 사용하여 kill함수로 첫 번째 예제에게 signal을 보낼 수 있다.

즉, 다른 프로세스에서 원할 때 무엇인가 제어가 가능해짐.

# 1. 시스템 프로그래밍 – 8

```c
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

struct sigaction act_new;
struct sigaction act_old;

void sigint_handler(int signo){

    printf("Ctrl + C\n");
    printf("If you push it one more time the exit\n");
    sigaction(SIGINT, &act_old, NULL);
}

int main(void){

    act_new.sa_handler = sigint_handler; // signal 2번째 인자에 넣는거랑 같다

    sigemptyset(&act_new.sa_mask);  // 아무것도 sig를 막지 않는다.

    sigaction(SIGINT, &act_new, &act_old);  // 이전 sig 정보를 act_old에 셋팅, act_new 실행

    // 남이 쓴 코드 빨리 파악하는 법,  함수에 &를 보고 &붙은 것은 값이 변경되겠구나.
    // 안에 있는 내용이 변경이 될 수 있겠다고 파악.
        while(1){

        printf("sigaction test\n");
        sleep(1);
    }

    return 0;
}
```

- sigaction()

위 함수는 원하는 signal을 취급할 수 있다.

예제에서처럼 ctrl +c를 두 번 동작해야 꺼지는 것과 같은 원하는 방식으로 설계할 수 있다.

# 1. CPU와 GPU의 차이

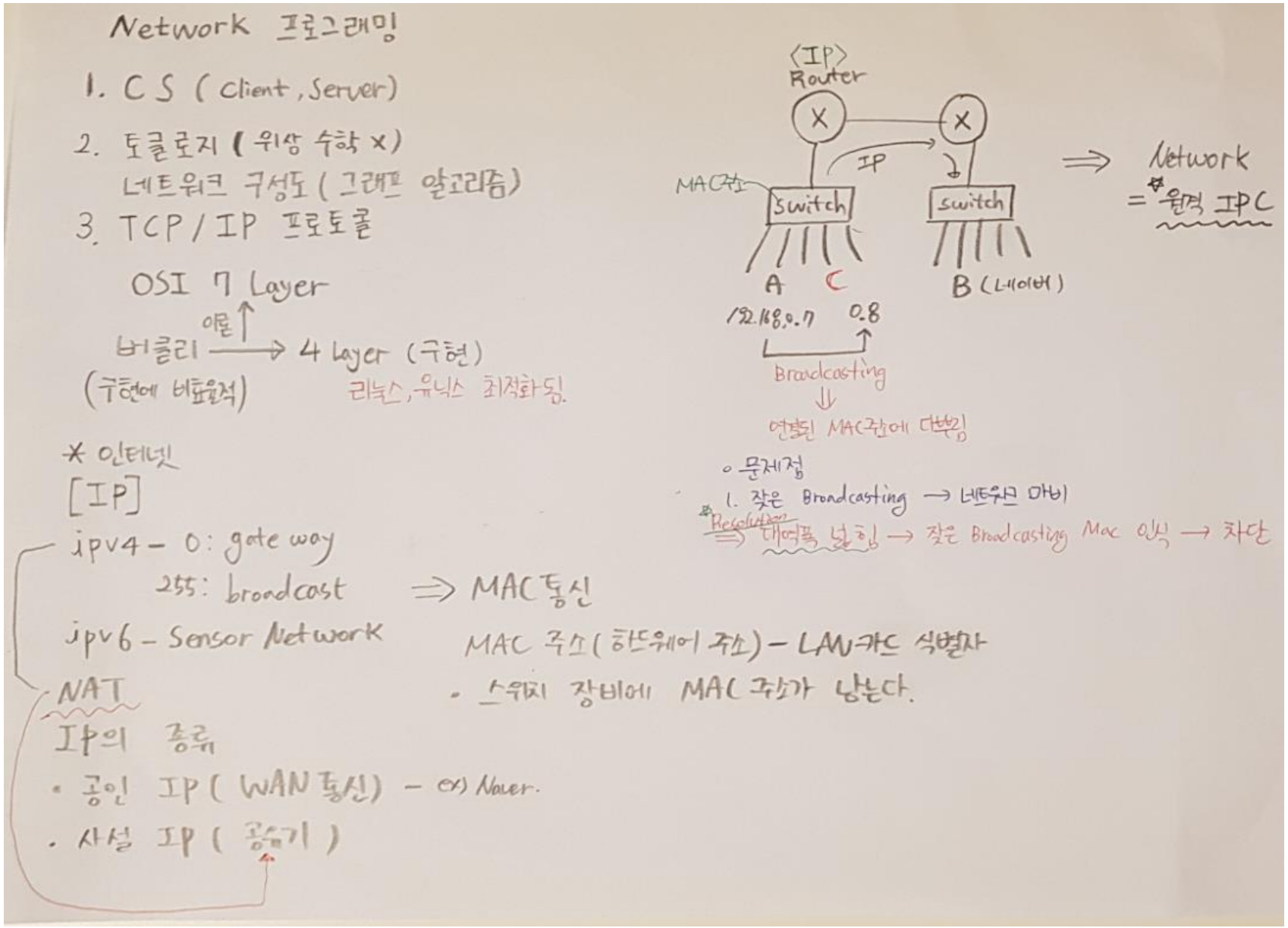
- CPU 고속 처리

cpu는 순차처리 특화로 클락 스피드가 굉장히 빠르기 때문에 빠른 연산에 유리하다.

- GPU 병렬 처리

Cpu 여러 개를 바탕으로 병렬처리하며 밴드 폭이 엄청 넓다. 클락 스피드는 느리지만, 픽셀과 같이 여러 개를 처리해야 할 때 굉장히 효율 적이다.

# 2. 네트워크 프로그래밍 개념




(추가내용)

- Port 번호 ( service number )

80 – web

20(upload), 21(download) – ftp

22 – ssh

# 2. 네트워크 프로그래밍 – Server

```c
#include<arpa/inet.h>
#include<sys/socket.h>

typedef struct sockaddr_in si;
typedef struct sockaddr* sap;

void err_handler(char *msg){

    fputs(msg, stderr);
    fputc('\n', stderr);
    exit(1);
}

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

    int serv_sock;
    int cint_sock;

    si serv_addr;
    si cint_addr;
    socklen_t cint_addr_size;

    char msg[] = "Hello Network Programming";

    if(argc !=2){
        printf("use : %s <port>\n", argv[0]);
        exit(1);
    }

    serv_sock = socket(PF_INET, SOCK_STREAM, 0);

    if(serv_sock == -1)
        err_handler("socket() error");

    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET; // tcp 형식 만들 때 쓰는 패턴
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);  // 127.0.0.1 = local host
    serv_addr.sin_port = htons(atoi(argv[1]));   // port 서비스 정보

    if(bind(serv_sock, (sap)&serv_addr, sizeof(serv_addr)) == -1)
        err_handler("bind() error");  // bind  server ip 주소 셋팅(127.0.0.1)

    if(listen(serv_sock, 5) == -1)   //client 5명 받겠다.
        err_handler("listen() error");  // 실제로 client를 기다림.

    cint_addr_size = sizeof(cint_addr);
    cint_sock = accept(serv_sock, (struct sockaddr *)&cint_addr, &cint_addr_size); //client의 접속 허용.

    if(cint_sock == -1)   // cint_sock의 정보fd 파일 디스크립터
        err_handler("accept() error");

    write(cint_sock, msg, sizeof(msg));  // socket->fd로 원격 동기화
                                         // 클라이언트한테 write한다.
    close(cint_sock);
    close(serv_sock);

    return 0;
}
```

- Server 동작방식

1. port 서비스 정보를 만든다.

2. bind로 server ip 주소를 셋팅한다

3. listen으로 client를 인자 수 만큼 기다린다.

4. accept으로는 client 별 접속을 허용한다.

5. write로는 클라이언트한테 data를 보낼 수 있다.

# 2. 네트워크 프로그래밍 – Client

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<sys/socket.h>

typedef struct sockaddr_in si;
typedef struct sockaddr* sap;

void err_handler(char *msg){

    fputs(msg, stderr);
    fputc('\n', stderr);
    exit(1);
}

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

    int sock;
    int str_len;
    si serv_addr;
    char msg[32];

    if(argc != 3){
        printf("use: %s <IP> <port>\n", argv[0]);
        exit(1);
    }

    sock = socket(PF_INET, SOCK_STREAM, 0); // 네트워크상 fd얻어옴
                                            // socket = open과 같다
    if(sock == -1)
        err_handler("socket() error");

    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(argv[1]);   // 192.168.0.x setting
    serv_addr.sin_port = htons(atoi(argv[2]));

    if(connect(sock, (sap)&serv_addr, sizeof(serv_addr)) == -1)
        err_handler("connect() error");   // connect하면server listen에서 받아서
                                          // accept함

    str_len = read(sock, msg, sizeof(msg) - 1); //server의 정보가 sock에 옮
                                          // 정보 받고 msg에 정보가 들어감.
    if(str_len == -1)
        err_handler("read() error!");

    printf("msg from serv: %s\n", msg);
    close(sock);

    return 0;
}
```

- client 동작방식

1. socket함수로 sock을 얻어온다.
2. connect함수로 server에 접속한다.
3. read함수를 통해서 서버로 부터 받은 data를 읽어온다.

# 2. 네트워크 프로그래밍 – read _ client

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<sys/socket.h>

typedef struct sockaddr_in si;
typedef struct sockaddr * sap;

void err_handler(char *msg){

    fputs(msg, stderr);
    fputc('\n', stderr);
    exit(1);
}

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

    int sock;
    int str_len = 0;
    si serv_addr;
    char msg[32] = {0};
    int idx =0, read_len = 0;

    if(argc !=3){
            printf("use: %s <IP> <port>\n",argv[0]);
            exit(1);
    }
    sock = socket(PF_INET, SOCK_STREAM, 0);

    if(sock == -1)
        err_handler("socket() error");

    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
    serv_addr.sin_port = htons(atoi(argv[2]));

    if(connect(sock, (sap)&serv_addr, sizeof(serv_addr)) == -1)
        err_handler("connect() error");

    while(read_len =read(sock, &msg[idx++], 1)){ //read에서 총 데이터 길이를 이미 알고 있음.
        if(read_len == -1)
            err_handler("read() error!");

        str_len += read_len;  // 도중에 읽히다가 끊어진 걸 고려해서 만듦
        // 총 14byte인데 중간에 끊겨서 12byte만오면 다시 read해서 14byte를
        //읽음
    }

    printf("msg from sev: %s\n", msg);
    printf("read count: %d\n", str_len);
    close(sock);

    return 0;
}
```

- Client에서 read를 하다가 도중에 데이터가 끊길 경우

1. read함수에서 총 데이터 길이를 가지고 있는다.
2. 총 16byte데이터가 12byte만 오고 연결이 끊길 경우 일단 str_len에 길이를 저장한다.
3. 이후 다시 read함수가 실행되서 남은 4byte를 가져와서 str_len에 덧붙인다.
4. 위 방식으로 데이터가 송신이 중간에 끊기더라도 끝까지 데이터를 받아온다.

# 2. 네트워크 프로그래밍 – socket fd

```c
#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
#include<sys/socket.h>

int main(void){

    int fd[3];
    int i;
    fd[0] = socket(PF_INET, SOCK_STREAM, 0);  //tcp --file ferm[0] s에 대한 정보
    fd[1] = socket(PF_INET, SOCK_DGRAM, 0); //udp
    fd[2] = open("test.txt", O_CREAT | O_WRONLY | O_TRUNC);

    for(i=0;i<3;i++){
        printf("fd[%d] = %d\n",i, fd[i]);
    }
    for(i=0;i<3;i++){
        close(fd[i]);
    }

    return 0;
}
```

- Socket의 리턴

Socket의 리턴값은 file discriptor이다.

즉, 결국 소켓도 파일이라는 뜻이다.

# 2. 네트워크 프로그래밍 – socket fd

```c
#include<stdio.h>
#include<arpa/inet.h>
int main(void){ //  변수 저장방식이 리틀인디안인지 빅 인디안인지 확인하는 코드

    unsigned short host_port = 0x5678;
    unsigned short net_port;
    unsigned long host_addr = 0x87654321;
    unsigned short net_addr;

    net_port = htons(host_port); // host to network  short 2byte
    net_addr = htonl(host_addr);  // host to network long 4byte

    printf("Host Ordered Port: %#x\n", host_port);
    printf("network ordered port: %#x\n", net_port);
    printf("Host ordered adrress: %lx\n", host_addr);
    printf("network drdered address: %#x\n", net_addr);

    // cpu 마다 인디안이 다르기 때문에 하나로 통일 시켜 줌.
    return 0;
}
```

```c
#include<stdio.h>
#include<arpa/inet.h>

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

    char *addr1 = "3.7.5.9";
    char *addr2 = "1.3.5.7";

    unsigned long conv_addr = inet_addr(addr1);
    if(conv_addr == INADDR_NONE)
        printf("Error!\n");
    else
        printf("Network Ordered integer Addr: %#lx\n", conv_addr);

    conv_addr = inet_addr(addr2);
    if(conv_addr == INADDR_NONE)
        printf("Error!\n");
    else
        printf("Network Ordered integer Addr: %#lx\n", conv_addr);

    return 0;
}
```

- 통신할 때 ip 주소의 전달 방식

cpu마다 인디안이 다르기 때문에 통신을 할 때

인디안 방식을 통일하고 각 컴퓨터 상황에 맞게

인디안을 변형시킨다.

즉, 이러한 방식으로 데이터 전달이나 ip주소가 꼬이는 일이 없게 만든다.