

Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – 장성환

redmk1025@gmail.com

*남이 짠 코드를 빠르게 파악하는 방법

&가 들어있는 것, 즉 포인터를 넣어준것은 함수내에서 값이 변경되거나 한다.

함수는 오직 output 이 하나지만, 포인터를 넣어주면 값 변경을 여러개를 해줄 수 있다. 따라서 &가 입력된 것을 확인 해야한다.

ctrl + y 위에것 따라쓰기

ctrl + e 아래 것 따라쓰기

vim 상태에서 u 되돌리기

gg=G 다 정렬

*sigaction.c

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

```
#include <signal.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 then exit\n");
```

```
    sigaction(SIGINT, &act_old, NULL);
```

```
}
```

```
int main(void){
```

```
    act_new.sa_handler = sigint_handler; //시그널의 두번째 변수
```

sigemptyset(&act_new.sa_mask); // 우리가 특정한 시그널이 들어오지 못하게 막을 수 있다. sigemptyset 은 아무것도 막지 않는다는 뜻이다.

```
    sigaction(SIGINT, &act_new, &act_old);
```

=====

sigaction() 시그널 처리를 다양하게 해주는 함수

```
int signum
```

시그널 번호

```
struct sigaction *act
```

설정할 행동 즉, 새롭게 지정할 처리행동

```
struct sigaction *oldact
```

이전행동, 이 함수를 호출하기 전에 지정된 행동 정보가 입력된다.

sa_flags 옵션

//act_new 를 동작시키고, 이전 핸들러를 act_old 에 넣는다.

```
while(1){
    printf("sigaction test\n");
    sleep(1);
}
return 0;
}
```

/*

sigaction 구조체 변수 act_new 와 act_old 를 선언

act_new 멤버인 sa_handler 에 sigint_handler 함수를 넣어 act_new 작동시 해당 함수가 작동되도록 함.

sigemptyset()함수를 통해 sigset_t 변수의 주소를 넘겨주어 해당 시그널 집합 내용을 모두 삭제.

sigaction()함수를 통하여
ctrl+c 입력과 설정행동인 act_new 그리고 이 함수를 호출전 지정된 행동인 old_act 를 전달한다.

old_act 는 널이므로 act_new.sa_handler 인 sigint_handler 가 실행되면 sigaction 이 다시 실행되어 act_old 가 들어간다.

따라서 ctrl+c 가 다시 입력되면 프로세스가 종료된다.

*/

옵션	의미
SA_NOCLDSTOP	sigum이 SIGCHLD일 경우, 자식 프로세스가 멈추었을 때, 부모 프로세스에 SIGCHLD가 전달되지 않는다.
SA_ONESHOT 또는 SA_RESETHAND	시그널을 받으면 설정된 행동을 취하고 시스템 기본 설정인 SIG_DFL 로 재설정된다.
SA_RESTART	시그널 처리에 의해 방해 받은 시스템 호출은 시그널 처리가 끝나면 재시작한다.
SA_NOMASK 또는 SA_NODEFER	시그널을 처리하는 동안에 전달되는 시그널은 블록되지 않는다.
SA_SIGINFO	이 옵션이 사용되면 sa_handler대신에 sa_sigaction이 동작되며, sa_handler 보다 더 다양한 인수를 받을 수 있습니다. sa_sigaction이 받는 인수에는 시그널 번호, 시그널이 만들어진 이유, 시그널을 받는 프로세스의 정보입니다.

sigemptyset() 시그널 집합 내용을 모두 삭제 (시그널을 하나씩 처리할 때 말고, 여러 개를 하나로 묶어서 한꺼번에 처리하는 것이 편할때 사용)

*시그널에 관하여

시그널 이름	내용
SIGABRT	abord() 함수를 호출하면 발생하며, 이 시그널을 받으면 코어 덤프하고 프로그램은 종료됩니다.
SIGALRM	alarm() 함수를 호출하면 발생하며, 이 시그널을 받으면 프로세스는 종료됩니다.
SIGBUS	하드웨어 결함이 탐지되면 발생하며, 이 시그널을 받으면 프로세스는 종료됩니다.
SIGCHLD	자식 프로세스가 종료될 때 부모 프로세스에 전달됩니다.
SIGCONT	중단된 프로세스가 이 시그널을 받으면 다시 활성화되어 실행이되며, 이미 실행 중이라면 무시됩니다.
SIGFPE	0으로 나누거나 부동소수점 오류 등이 생기면 발생하며, 프로세스는 코어 덤프 후에 종료됩니다.
SIGHUP	터미널 연결이 끊어지면 이 터미널과 연결된 세션 리더 또는 세션에 속한 모든 프로세스에게 전달되면, 전달 받은 프로세스는 종료됩니다.
SIGILL	불법 명령어를 실행할 때 발생하며, 이 시그널을 받으면 코어 덤프 후에 종료합니다.
SIGINT	터미널에서 인터럽트 키 중의 하나인 Ctrl-C 를 눌렀을 때 발생하며, 프로세스는 종료됩니다.
SIGKILL	프로세스를 종료하기 위해 전송되며, 시그널을 받은 프로세스는 반드시 종료합니다.
SIGPIPE	이미 닫힌 파이프에 쓰기를 할 때 발생하며, 이 시그널을 받으면 종료합니다.
SIGQUIT	터미널에서 종료키인 Ctrl-\를 눌렀을 때 발생하며, 프로세스는 코어 덤프 후에 종료합니다.
SIGSEGV	잘못된 메모리 주소를 접근할 때 발생하며, 이 시그널을 받은 프로세스는 코어 덤프 후에 종료합니다.
SIGSTOP	프로세스를 멈추기 위해 보내지며, 이 시그널을 받은 프로세스는 반드시 멈춥니다.
SIGSYS	잘못된 시스템 호출을 했을 때 보내지며, 이 시그널을 받은 프로세스는 코어 덤프 후에 종료합니다.
SIGTERM	프로세스가 종료 전에 처리해야될 작업을 처리할 수 있도록 종료 전에 미리 보내 집니다.
SIGSTP	터미널에서 프로세스를 잠시 멈추게 하기 위해 Ctrl-Z키를 눌렀을 때 전송되며, 이 시그널을 받은 프로세스는 멈춤이 됩니다.
SIGTTIN	백그라운드에서 작업 중인 프로세스가 표준 입력을 사용하려 할 때 현재 실행 중인 프로세스에 전송되며, 이 시그널을 받은면 멈춤상태가 됩니다.
SIGTTOU	백그라운드에서 작업 중인 프로세스가 표준 출력을 사용하려 할 때 현재 실행 중인 프로세스에 전송되며, 이 시그널을 받은면 멈춤상태가 됩니다.
SIGURS1	사용자 정의로 사용할 수 있는 시그널로, 이 시그널을 받으면 프로세스는 종료합니다.
SIGURS2	

*kill.c	=====
	kill() 시그널 전송하는 함수

```
#include <stdio.h>
#include <signal.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;
}
// gcc -o kill kill.c
// ./test & (&의 의미는 백그라운드에서 동작 시킨다는 것이다.)
```

셸에서 프로세스를 죽이는 kill 명령어와 달리 프로세스에 시그널을 전송해 주는 함수이다.

/*

프로그램 실행시 추가한 프로세스 id 는 문자열로 해당 문자열이 atoi 함수에 의하여 정수형 변환이 되어 인자로 들어간다. (id 전달)

시그널 번호는 SIGINT 를 전달하여 ctrl+c 를 전달한다.

따라서 백그라운드로 동작하던 kill 프로세서가 kill2 가 kill 프로세스의 id 로 ctrl+c 를 보낸 것이다.

따라서 kill 프로세서에 등록된 signal 의 함수 gogogo 는 해당 문자를 출력하고 종료하게 된다.

*/

* kill2.c

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

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

    return 0;
}

// gcc -o kill2 kill2.c
// ./kill2 pid(숫자)
```

* thread.c

스레드는 종속적이라고 했다. 따라서 메모리를 공유한다.
이 때문에 가장 큰 문제가 임계영역이다.

```

#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);
//스레드를 생성한다. 1 번째는 입력한 스레드의 값을 뭔가 채워준다는 것이고 3 번째
인자는 구동시킬 함수 리턴은 보이드라서 어떤것이든지 바꿀 수 있다.
// 스레드를 이렇게 구동시키겠다고 등록만 해 놓은 형태이다. 실제 메모리에 올리는
시점은 ? 조인이다.
    pthread_create(&ThreadB, NULL, task2, NULL);

    pthread_join(ThreadA, NULL);

    pthread_join(ThreadB, NULL);

    return 0;
}

// gcc thread.c -lpthread 로 구동한다.

/*

```

따라서 락을 걸어주고 사용한다.

병렬처리에서 스레드를 이용하여 연산을 극대화! 우선은 개념을 잡자.

OPENCL 이 붙으면 스레드는 여러개

gpu 는 병렬처리

아주빠른 고속처리 - dsp 이용 즉, cpu 는 고속의 순차처리

많은 병렬연산처리 - fpga? 즉, gpu 병렬 연산

gpu 는 처리 bandwidth 가 넓다!

인터페이스는 동일한데 다른 함수를 쓰고싶은 경우 함수포인터를 이용한다.

task_struct 에서 다 사용을 한다. (open 의 경우)

pthread_create() 스레드 생성 함수

pthread_t * thread - 스레드 성공적 생성시 넘겨주는 스레드 식별자

void * (*start_routine)(void*) - 스레드가 수행할 함수로 함수 포인터 넘겨줌

성공적 수행시 0 리턴, 실패시 1 을 리턴한다.

pthread_join() 스레드 정리 함수

(pthread_create()로 생성시킨 스레드는 pthread_join()을 통해서 기다리면 된다)

pthread_t - pthread_create 에 의해 생성된, 식별번호 th 를 가진 스레드를 기다림

void **thread_return

- 식별번호 th 인 스레드의 종료시 리턴값

각각 thread_a, b 를 선언하고 식별자를 받기위하여 pthread()함수에 넣는다.
수행할 함수는 각각 task1, task2 를 입력하였다.

pthread_join()함수로 쓰레드 식별자를 기다린다.

*/

* 네트워크 프로그래밍

Network 에서 가장 중요한 것은 ?

1. CS (Client, Server)
2. 토클로지(위상수학은 아니고 네트워크구성도(그래프 알고리즘))
3. TCP/IP 프로토콜(OSI 7 layer 버클리에서 7 계층이 너무 많다고 하여 4 계층을 만들었는데 이를 TCP/IP 프로토콜 이라고 한다.)
- 이 프로토콜은 라우터, 스위치, os 에 들어간다 리눅스 태생이라 리눅스, 유닉스에 최적화 되어있다.

도대체 어떤 식으로 네트워킹을 할 것인가 ?

인터넷을 하기위하여 IP(internet protocol)이 필요하다. (ip 확인은 ifconfig 여기서 inet addr 가 ipaddr 이다.)

ip 정보에서 ipv4, ipv6, NAT 가 중요하다.

ip 정보에서 ip 는 0(게이트웨이)하고 255(브로드캐스트)는 예약된다.

IP 에는 종류가 2 가지 있다.

1.공인 아이피 2.사설 아이피

공인 아이피가 있어야 외부로 나갈 수 있다. (WAN 통신을 하려면 필수적)

사설 아이피는 공유기 아이피이다.

어떻게 사설아이피들이 많은데 공인아이피가 할 수 있는 인터넷을 할까?

NAT 가 처리를 해준다.

ipv6 는 센서 네트워크이다 (ipv4 의 수가 적다)

MAC 통신 이란?

HWaddr 에 적힌 것이 mac 주소이다. 이 하드웨어 장치의 고유한 번호(LAN 카드의 식별자)를 나타낸다. 해킹시 mac 주소가 흔적으로 남는다.

어디에 흔적이 남는가?

스위치 장비에 남게 된다. 스위치 장비에 남게 되는 이유는 뭘까 ?

mac 주소를 보고 어디로 보내줘야 할지 정하기 때문이다.

ip 주소를 보고 보내주는것은 라우터이다.

<pre> * basic_server.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 serv_sock; int clnt_sock; si serv_addr; si clnt_addr; socklen_t clnt_addr_size; //32byte char msg[] = "Hello Network Programming"; if(argc != 2){ printf("use: %s <port>\n", argv[0]); exit(1); } // 포트번호 입력하라는 것 포트번호는 통로이다 </pre>	<p>실행방법 ./serv 7777 ./clnt 127.0.0.1 7777</p> <p>=====</p> <p>socket() 소켓을 생성하여 반환합니다.</p> <p>int domain 인터넷을 통해 통신할 지, 같은 시스템 내에서 프로세스끼리 통신할 지의 여부를 설정합니다.</p> <p>PF_INET, AF_INET IPv4 인터넷 프로토콜을 사용 PF_INET6 IPv6 인터넷 프로토콜을 사용 PF_LOCAL, AF_UNIX 같은 시스템 내에서 프로세스 끼리 통신 PF_PACKET Lowlevel socket 을 인터페이스로 이용 PF_IPX IPX 노벨 프로토콜을 사용.</p> <p>int type 데이터의 전송 형태를 지정하며 아래와 같은 값을 사용할 수 있습니다.</p> <p>SOCK_STREAM TCP/IP 프로토콜을 이용 SOCK_DGRAM UDP/IP 프로토콜을 이용</p> <p>int protocol 통신에 있어서 특정 프로토콜을 사용 지정을 위한 변수 보통 0 을 사용.</p>
---	--

```

// 80 은 www 22 ssh 20 ftp 해당 번호가 특정역할을 한다(서비스)
// 웹브라우저 들어가면 무조건 80 번
// 7777 은 우리가 만든 전용 커스텀 포트 (이것을 전용 서비스 번호라고 한다)

serv_sock = socket(PF_INET, SOCK_STREAM, 0);
//네트워크도 파일이다! 파일 디스크립터 반환!
if(serv_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 = htonl(INADDR_ANY);
serv_addr.sin_port = htons(atoi(argv[1])); //자기자신이 되겠다.
//127.0.0.1 은 로컬 호스트 자기 자신을 가리킨다.

if(bind(serv_sock, (sap)&serv_addr, sizeof(serv_addr)) == -1)
    err_handler("bind() error");
//bind 는 서버에 ip 주소 세팅
if(listen(serv_sock, 5) == -1)//5 명까지 받겠다는 것
    err_handler("listen() error");

clnt_addr_size = sizeof(clnt_addr); //32
clnt_sock = accept(serv_sock, (struct sockaddr*)&clnt_addr,
&clnt_addr_size); //서버 소켓이 클라이언트를 기다림 (실제 클라이언트 기다리는
구간은 listen 이다) sockaddr 찍어보면 접근한 ip 주소 나온다. 원격으로 세마포어
를만드는 것. 따라서 read write 등 다 할 수 있다.

if(clnt_sock == -1)
    err_handler("accept() error");

write(clnt_sock, msg, sizeof(msg)); //클라이언트 소켓으로 메시지 날아감

```

bind() 소켓에 IP 주소와 포트번호 지정

프로토타입은 **int bind(int sockfd, struct sockaddr * myaddr, socklen_t addlen)**

소켓 디스크립터, 주소정보, 주소정보의 크기를 인자로 전달하여 리턴값은 성공시 0 실패시 -1 을 리턴.

int sockfd

소켓디스크립터, 소켓을 생성하는 socket()의 리턴값인 소켓 식별자

struct sockaddr *myaddr

시스템 내부 통신인 AF_UNIX 인 경우에는 struct sockaddr 을 사용.

socklen_t addrlen

myadd 구조체의 크기

listen() 클라이언트 접속 요청을 받을 수 있도록 설정

int s

소켓 디스크립터, socket()의 리턴인 소켓식별자

int backlog

대기 메시지 큐의 갯수

<pre> close(clnt_sock); close(serv_sock); return 0; } </pre>	<div> <div>-----</div> <div> accept() 클라이언트의 접속 요청을 받고, 클라이언트와 통신하는 전용 소켓을 생성 </div> <div> 소켓디스크립터, 클라이언트 주소 정보의 포인터, 포인터가 가리키는 구조체 크기를 전달하여 리턴값은 성공시 -1 이외의 새로운 소켓디스크립터 식별자 실패시 -1 를 리턴한다. </div> <div> int s 소켓 디스크립터 </div> <div> struct sockaddr* addr 클라이언트 주소 정보를 가지고 있는 포인터 </div> <div> socklen_t *addrlen *addr 포인터가 가리키는 구조체의 크기 </div> <div>-----</div> <div> connect() 서버로 접속 요청 </div> <div> 소켓 디스크립터, 서버 주소정보에 대한 포인터, 포인터가 가리키는 구조체의 크기를 인자로 전달하여 리턴값은 성공시 0 실패시 -1 을 리턴한다. </div> <div> int sockfd 소켓 디스크립터 </div> <div> struct sockaddr* serv_addr 서버 주소 정보에 대한 포인터 </div> <div> socklen_t addrlen serv_addr 포인터가 가리키는 구조체의 크기 </div> </div>
---	---

<pre> * basic_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; //서버 어드레스 void err_handler(char *msg){ fputs(msg, stderr); fputc('\n', stderr); exit(1); } //192.168.0.X - 192 168 0 은 무조건 사설 ip 주소 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); } </pre>	<pre> ===== </pre>

sock = socket(PF_INET, SOCK_STREAM, 0); //내가 통신을 할 수 있는 파일 디스크립터를 받아옴. 네트워크에서는 socket 이 fd 와 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]); //입력한 ip 주소 세팅
serv_addr.sin_port = htons(atoi(argv[2]));
```

// 패턴이니 그냥 이대로 쓰면 된다.

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

str_len = read(sock, msg, sizeof(msg) - 1); //블록킹함수 들어올때까지 움직이지 않는다.

```
if(str_len == -1)
    err_handler("read() error!");
```

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

```
return 0;
```

```
}
```

//소켓은 이미 세마포어 처리가 되어있다.

```
*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);
```

```
    fd[1] = socket(PF_INET, SOCK_DGRAM, 0);
```

```
    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;
```

```
}
```

소켓이 파일과 같음을 증명하는 예제이다.

* 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);
```

중간에 데이터가 끊기더라도 손실 방지하는 방법이다.

```
while(read_len = read(sock, &msg[idx++], 1)){
    if(read_len == -1)
        err_handler("read() error!");

    str_len += read_len;
} // read 함수로 read_len 을 체크하고 있다.
```

처음에 connect 하고 read 할때 총 길이 정보를 알게 된다.
따라서 그만큼의 정보가 오지 않으면 오류로 판단한다.

```
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)) == 0)
    err_handler("connect() error");

while(read_len = read(sock, &msg[idx++],1)){
    if(read_len == -1)
        err_handler("read() error!");

    str_len += read_len;
} // read 함수로 read_len 을 체크하고 있다.

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

return 0;
}
```

```
* convert_endian.c
```

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

```
int main(void){
```

```
    unsigned short host_port = 0x5678;
```

```
    unsigned short net_port;
```

```
    unsigned long host_addr = 0x87654321;
```

```
    unsigned long net_addr;
```

```
    net_port = htons(host_port);
```

```
    net_addr = htonl(host_addr);
```

```
    printf("Host Ordered Port : %#x\n", host_port);
```

```
    printf("Network Ordered Port : %#x\n", net_port);
```

```
    printf("Host Ordered Address : %#x\n", host_addr);
```

```
    printf("Network Ordered Port : %#x\n", net_port);
```

```
    return 0;
```

```
}
```

빅엔디안과 리틀엔디안의 차이점을 알아보는 코드!

빅엔디안은 순차적으로 받고

리틀엔디안은 반대로 꼬아 받는다.

빅엔디안에서 리틀엔디안의 정보를 주고받을 수 있고

리틀엔디안에서 빅엔디안에서 받을 수 있다.

따라서 한가지 기준점을 가지고 꼬이지 말라고 변환해주는 것이다.

* inet_addr.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;
```

```
}
```

빅엔디안 리틀엔디안 저장 차이에 따라서

네트워크 통하여 가는 것에

애초에 메모리에 들어가는 형식으로 변환 후 전달하는 개념

*보드 구하는 방법

TI 보드와 Xilinx 보드

이니프로 쇼핑몰

zybo kit academic 직구 필요 없음 FPGA

TI 보드는 직구해야함

Texas instruments 회원가입

mcu 쪽 hercules safety mcu tms570LC43x cortex r5

국제표준 차량용

rm5720 tool and software

development kit

아마존 직구하듯이 한다.jtag 내장

dsp 는

sitara process

A15 am572x evaluation mode 차량의 중앙제어

카메라 안섬

맨 마지막에 체크 하는거 잘못하면 큰일 남. 군용옵션 체크하지 말것

기압센서 / BLDC / 서보 / 라이다 isquere c