CLI 기반의 하드웨어

한밭대학교 반도체설계실

Lee, Jae Heung

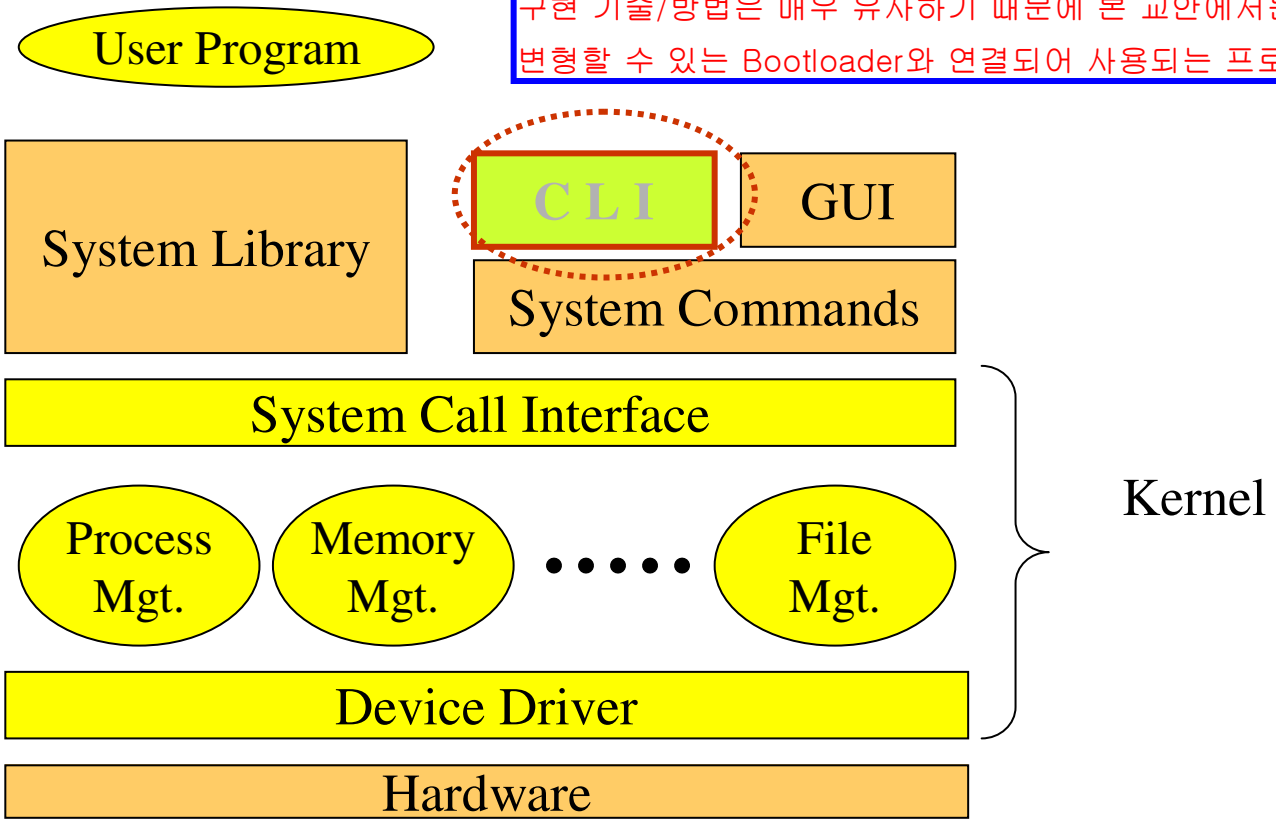
명령어 라인 인터페이스 개요

명령어 라인 인터페이스(command Line Interface)

- CLI 주요 기능
  - 명령어 / 인수 입력 처리
  - 사용자로부터의 각종 요구를 분석하고 처리 결과를 출력
  - 시스템 초기화/디버깅
  - 시스템의 상태/동작을 모니터링
  - 시스템 환경 설정
- 그래픽 인터페이스를 갖지 않는 장비에서 필수
- Web server와 연동하도록 하는 구조로 변화
- 본 강좌에서는 부트로더의 기능 일부를 이용해서 실습 진행

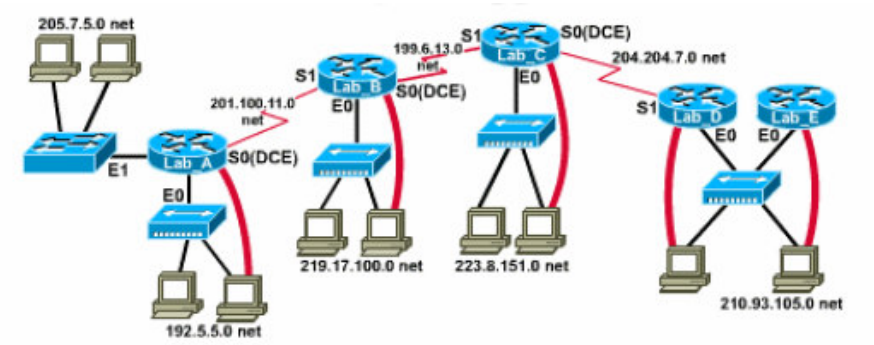
명령어 라인 인터페이스

CLI는 운영체제가 있고 없음에 따라
또한 운영체제에 따라서도 구현하는 위치/구조가 매우 다양할 수 있으나,
구현 기술/방법은 매우 유사하기 때문에 본 교안에서는 학생들이 쉽게
변형할 수 있는 Bootloader와 연결되어 사용되는 프로그램을 통해 학습



명령어 라인 인터페이스(2)

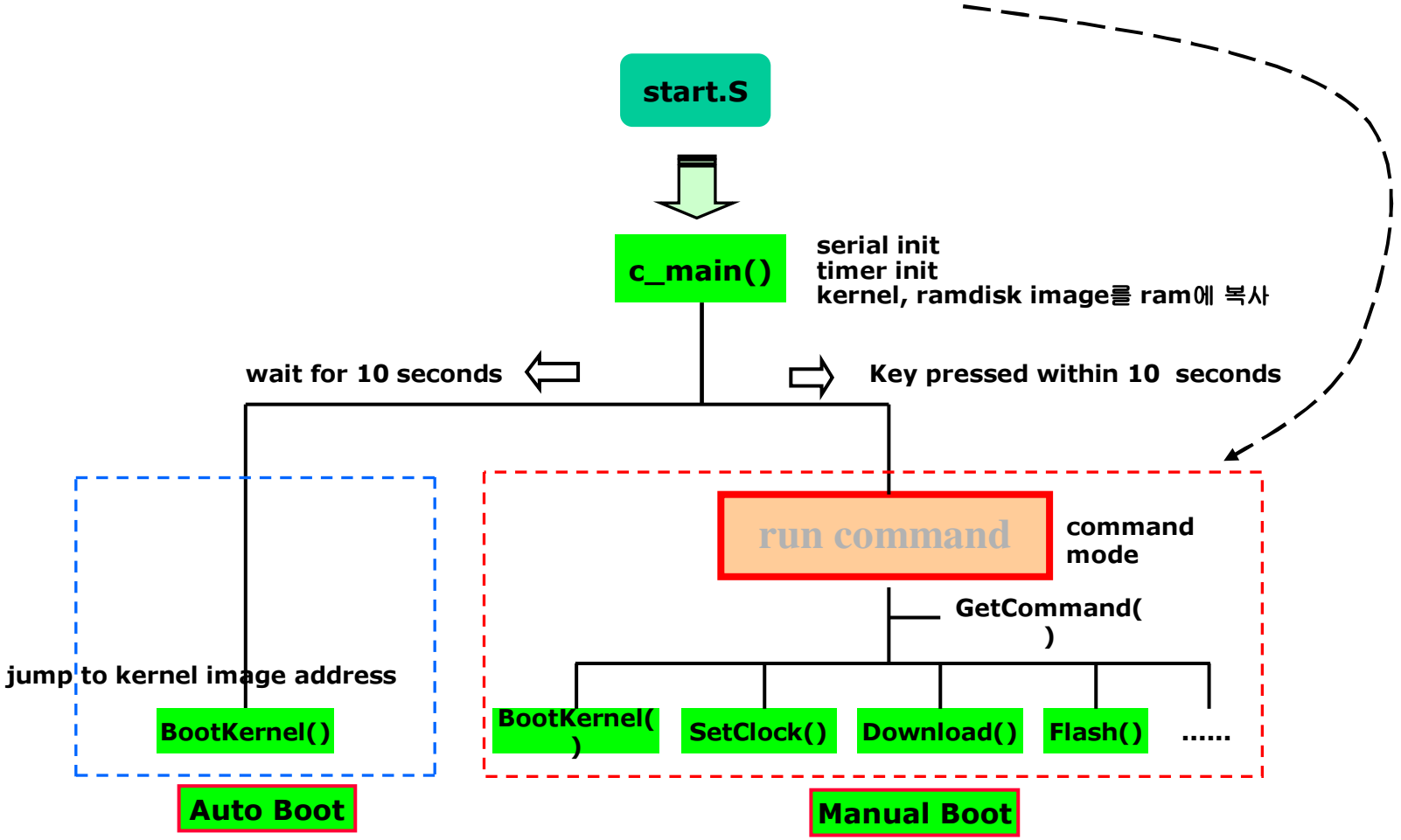
시스코 라우터에서의 **CLI** 예




```
rommon 16 > IP_ADDRESS=192.168.106.105
   라우터의 IP 주소를 설정한다.
rommon 17 > IP_SUBNET_MASK=255.255.255.0
   라우터의 서브넷매스크를 설정한다.
rommon 18 > DEFAULT_GATEWAY=192.168.106.1
   라우터의 디폴트게이트웨이를 설정한다.(큰 의미는 없다)
rommon 19 > TFTP_SERVER=192.168.106.5
   다운로딩할 IOS 소프트웨어가 있는 TFTP 서버의 IP 주소를 적는다.
rommon 20 > TFTP_FILE=c1700-y-mz[1].123-1a.bine
   라우터로 다운로딩할 IOS 파일 이름을 적는다.
rommon 25 > tftpdnld
        IP_ADDRESS: 192.168.106.105
    IP_SUBNET_MASK: 255.255.255.0
```

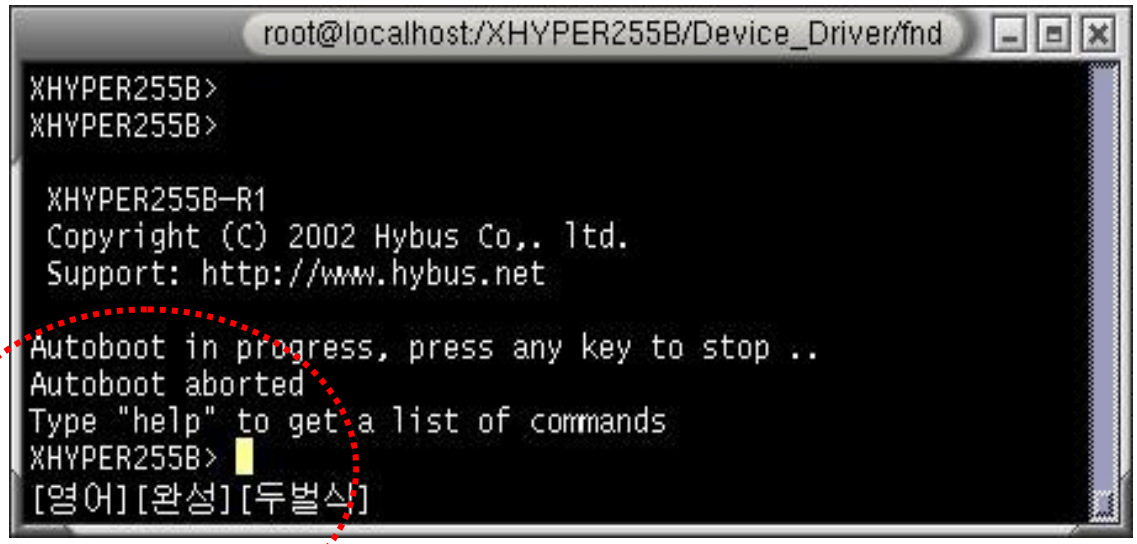
Command Line Interface 구현(예)

bootloader에서의 명령어 인터페이스(CLI)



Command Line Interface 사용(1)

- 전원을 인가한 후(**Power On**) 수 초 내에 키를 입력하면 명령모드에 진입

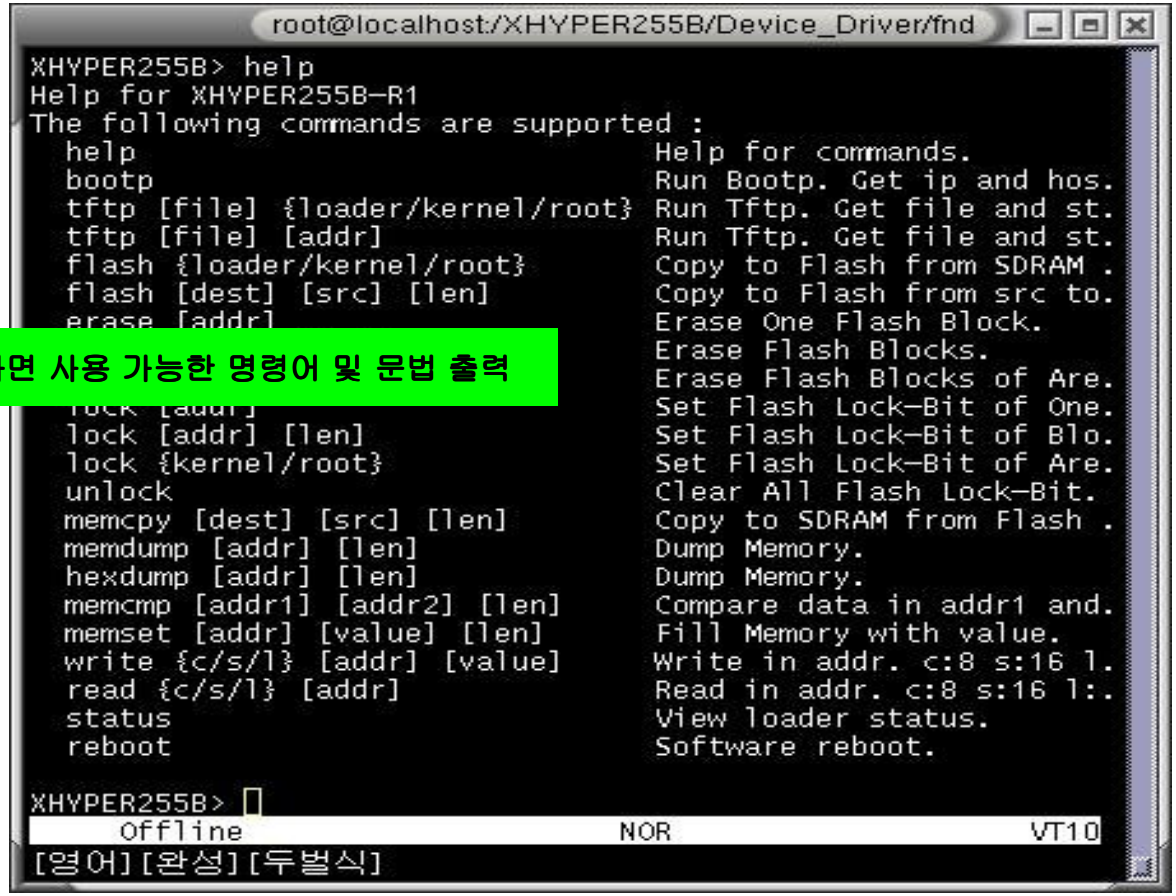


명령 입력을 기다리는 **prompt** 출력

Command Line Interface 사용(2)

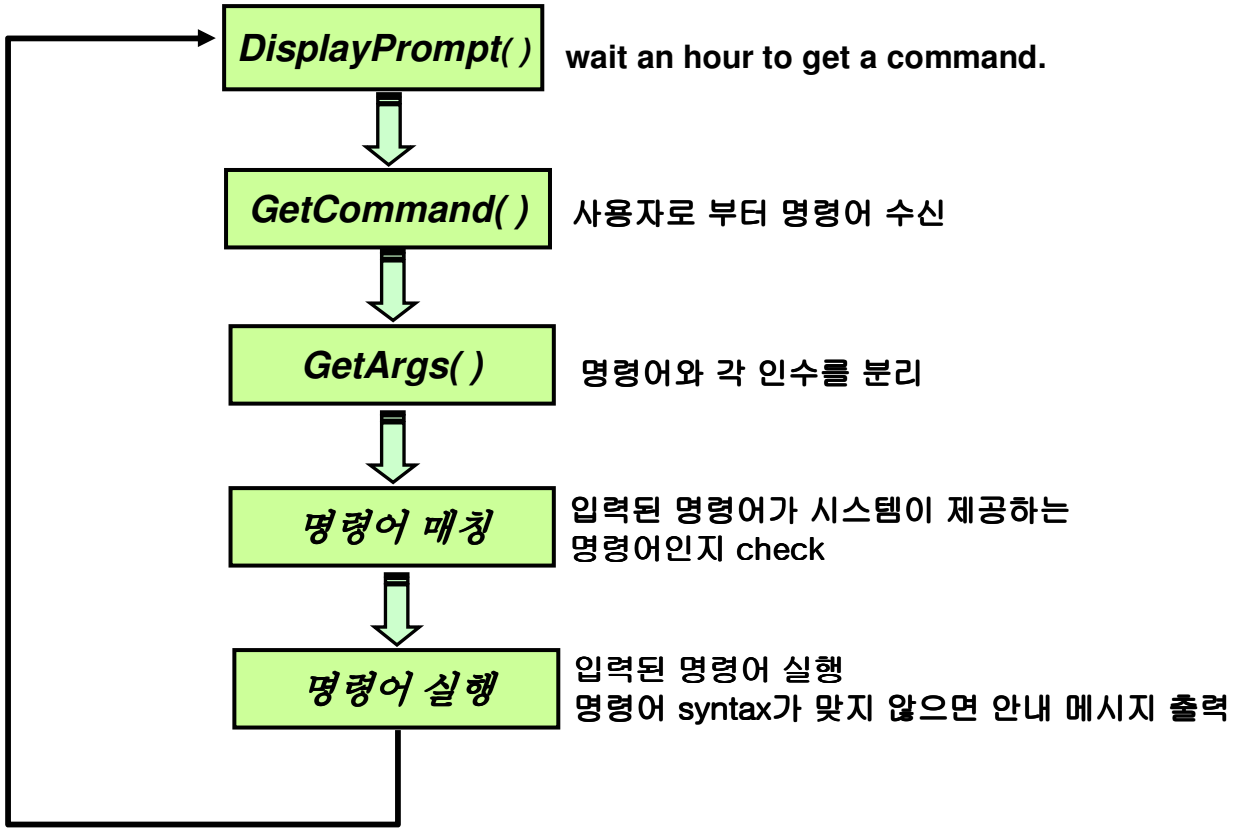
- 대부분의 시스템에서 명령 모드에서 사용 가능한 명령어는 **'help'** 혹은 **'?'**에 의해 알 수 있음

Help를 입력하면 사용 가능한 명령어 및 문법 출력



Command Line Interface 구현(2)

▣ 명령어 처리 흐름도



Command Line Interface 구현(3)

run command		
명령어	함수	기능
boot	BootKernel()	Kernel이 있는 Ram영역으로 Jump
reset	ResetTerminal()	Serial init
clock	SetClock()	Clock 설정
reload	Reload()	Flash의 kernel, ramdisk를 ram으로 copy
speed	SetDownloadSpeed()	Speed 설정
download	Download()	uudecode를 이용하여 Ram에 Download
flash	Flash()	Ram에서 Flash로 writing
status	PrintStatus()	현재 설정되어 있는 상태를 출력
serial	PrintSerialSpeed()	현재 serial의 speed 출력
help	PringHelp()	Help 메시지 출력
write	DoWriteToReg()	LED/FND 의 테스트
memdump	DoMemDump()	Memory Read 테스트

해당 명령어를 입력하면 명령어를 처리하는 함수를 호출하는 구조로 구현

따라서 함수 포인터에 대한 자세한 이해가 필수적으로 요구됨

Embedded 에서 사용하는 C 언어(1)

- 포인터를 사용한 **memory-mapped IO** 장치 접근
- 예제

```
#define SA_REG_READ (a, val) ((val) = *(volatile UNIT32 *)(a))
#define SA_REG_WRITE (a, val)  (*(volatile UNIT32 *)(a) = (val))

#define UART_CR         0x90003800
#define UART_SR         0x90003400
#define UART_RX_INT_EN      (1<<4)
#define UART_TX_INT_EN      (1<<8)

UNIT32 CR_word=0;

CR_word |= UART_RX_INT_EN | UART_TX_INT_EN;
SA_REG_WRITE (UART_CR, CR_word);
```

volatile 의 사용 및 의미
I/O 주소의 define 처리
Shift 연산자 사용

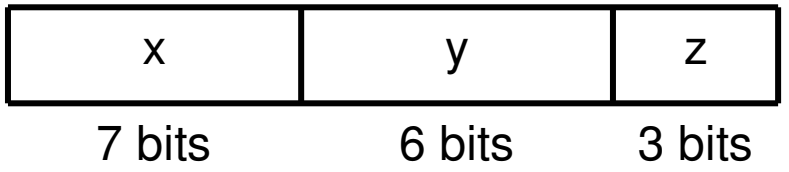
Embedded 에서 사용하는 C 언어(2)

Bit-wise 연산자 이해

Operation	Boolean op.	Bitwise op.
AND	&&	&
OR	||	|
XOR	unsupported	^
NOT	!	~

Bit field 이해

```
typedef struct
  {
  WORD16    x       :7,
            y       :6,
            z       :3;
  } IO_WORD
```



- **Can be referenced as regular structure**
  - **if** *IO_WORD* *GIO*, **then** *GIO.x GIO.y* **and** *GIO.z* **are valid**
  - **if** *WORD16* *GIO*,
    - y = (GIO >>3) & 0x003F;
    - GIO |= (x &0x007F) << 9;

CLI 관련 헤더파일(1)

사용자로 부터 명령을 읽고 분석하여 원하는 함수를 호출하는 데 필요한 자료구조 및 관련함수 prototype 선언

```
#define ADDR32(A)       (*((volatile ulong *)(A)))
#define ADDR16(A)       (*((volatile ushort *)(A)))
#define ADDR8(A)        (*((volatile uchar *)(A)))
#define MAX_COMMANDS            20
#define MAX_ARGS                20

struct _CMD_TBL {
    char        *cmd;                   // 이 함수를 호출할 명령어.
    bool        (*run)(struct _CMD_TBL *cptr, int argc, char **argv); // 함수 포인터
    char        *usage;                 // command가 잘 못되었을 때 나올 message.
    char        *help;                  // help에서 보여줄 message.
    char        *helpMore;
};

typedef struct _CMD_TBL   CMD_TBL;
extern CMD_TBL cmdTbl[];

// Prototypes.
void        DisplayPrompt(char *prompt);
int         GetCommand(char *command, int len, int timeout);
int         GetArgs(char *s, char **args);
bool        DoBootKernel(CMD_TBL *cptr, int argc, char **argv);
bool        DoReload(CMD_TBL *cptr, int argc, char **argv);
void        ClearLine(void);
```

CLI 관련 헤더 파일(2)

사용자로 부터 명령을 읽고 분석하여 원하는 함수를 호출하는 데 필요한 자료구조 및 관련함수 prototype 선언

```
bool DoResetTerminal    (CMD_TBL *cptr, int argc, char **argv);
bool DoPrintStatus      (CMD_TBL *cptr, int argc, char **argv);
bool DoTest             (CMD_TBL *cptr, int argc, char **argv);
bool DoReboot           (CMD_TBL *cptr, int argc, char **argv);

#define CMD_TBL_BOOT   \
   {"boot", DoBootKernel,"  boot  Booting the Kernel.\n",  “boot  Booting the Kernel.\n"}
#define CMD_TBL_RELOAD
   {"reload", DoReload, "  reload {kernel}  Copy to SDRAM from Flash in Area.\n", \
                        "  reload {kernel}        Copy to SDRAM from Flash in Area.\n"   }
#define CMD_TBL_STATUS   {"status", DoPrintStatus, "  status View loader status.\n",    0  \
     "  status   View loader status.\n“ }
#define CMD_TBL_REBOOT    {"reboot", DoReboot, "  reboot  Software reboot.\n", 0, \
     "  reboot                  Software reboot.\n“ }
#define CMD_TBL_TEST     {"test", DoTest,   0,  0,  0 }
#define CMD_TBL_MEMDUMP  {"memdump", DoMemDump, \
     "  memdump [addr] [len]  Dump Memory.\n",  0,  "  memdump [addr] [len] Dump Memory.\n“ }
#define CMD_TBL_END    {0, 0, 0, 0, 0}
```

CLI 관련 헤더 파일(3)

사용자로 부터 명령을 읽고 분석하여 원하는 함수를 호출하는 데 필요한 자료구조 및 관련함수 prototype 선언

```
CMD_TBL cmdTbl[] = {
        CMD_TBL_RELOAD,
        CMD_TBL_BOOTP,
        CMD_TBL_TFTP,
        CMD_TBL_FLASH,
        CMD_TBL_ERASE,
        CMD_TBL_LOCK,
        CMD_TBL_UNLOCK,
        CMD_TBL_BOOT,
        CMD_TBL_MEMCPY,
        CMD_TBL_MEMDUMP,
        CMD_TBL_HEXDUMP,
        CMD_TBL_MEMCMP,
        CMD_TBL_MEMSET,
        CMD_TBL_WRITE,
        CMD_TBL_READ,
        CMD_TBL_STATUS,
        CMD_TBL_REBOOT,
        CMD_TBL_TEST, // 명령어를 추가하는 실습시 여기에 추가하면 됨
        CMD_TBL_END};
```

CLI - 명령어 입력 처리 코드(1)

```
int GetCommand(char *cmd, int len, int timeout) {
    char        c;
    int         i, rdCnt, rdMax = len-1;
    volatile long  endTime=GetTime()+timeout*HZ;
    for (rdCnt=0, i=0; rdCnt < rdMax;){
                // try to get a byte from the serial port.
        while (!SerialInputByte(&c)){
            if (GetTime() > endTime){
                cmd[i++] = '\0';
                return rdCnt;
            }
        }
        if ((c=='\r') || (c=='\n')){
            cmd[i++] = '\0';     // print newline.
            printf("\n");   return rdCnt;
        } else if (c == '\b'){
            if(i > 0){
                i--;     rdCnt--;              // cursor one position back.
                printf("\b \b");
            }
        } else {
            cmd[i++] = c;  rdCnt++;  // print character.
            printf("%c", c);
        }
    }
    return(rdCnt);
}               // GetCommand.
```

사용자로 부터 하나의 명령을 serial port를 통해 읽어 들이는 함수

CLI - 메모리 덤프 명령어 처리 루틴 예(1)

```
bool DoMemDump(CMD_TBL *cptr, int argc, char **argv){
    ulong       i, addr, len, endptr;
    char        value;

    if (argc < 3){
        printf(cptr->usage);
        return false;
    }
    if (!HexToInt(argv[1], &addr, 32) || !HexToInt(argv[2], &len, 32)){
        printf("Illegal character is used.\n");
        return false;
    }
    endptr = addr + len;
    for (i=0; addr < endptr; i++, ((char *)addr)++){
        value = ADDR8(addr);
         if ((i)%4==0)
            printf("0x");
         printf("%02x", value);
         if ((i+1)%4==0){
            printf("  ");
         }
         if ((i+1)%16==0)
            printf("\n");
    }
    return true;
} // DoMemDump.
```

특정한 메모리에 있는 데이터를 출력하는 명령어 처리루틴 작성 예

CLI - I/O write 명령어 처리 루틴 예(1)

```
bool DoWriteToReg(CMD_TBL *cptr, int argc, char **argv){
   char *addr=0; char c;short s; long l;

   if (argc < 4) { printf(cptr->usage); return false; }
   if (!HexToInt(argv[2], &addr, 32)){
      printf("Illugal character is used.\n");
      return false;
   }
   switch (argv[1][0]){
      case 'c' : if (!HexToInt(argv[3], &c, 8)){
            printf("Illugal character is used.\n"); return false;
         }
         ADDR8(addr) = c;  break;
      case 's' :  if (!HexToInt(argv[3], &s, 16)){
            printf("Illugal character is used.\n"); return false;
         }
         ADDR16(addr) = s;  break;
      case 'l' :  if (!HexToInt(argv[3], &l, 32)){
            printf("Illugal character is useqd.\n");  return false;
         }
         ADDR32(addr) = l;  break;
      default :  printf("write [c/s/l] [addr] [value].\n"); return false;  break;
   }
   return true;
} // DoWriteToReg.
```

특정한 번지에 값을 쓰는 루틴으로 LED FND 등의 주소에 값을 써봄으로써 동작을 확인하는데 사용할 수 있다.

CLI - 명령어 인수 처리 코드(1)

```
int GetArgs(char *s, char **argv)
{
    int args = 0;

    if (!s || *s=='\0') return 0;
    while (args < MAX_ARGS){
        while ((*s==' ') || (*s=='\t')) s++; // skip space and tab.
            if (*s=='\0'){ // check end of line
                  argv[args] = 0;
                  return args;
            }
    argv[args++] = s; // start get arg.
    while (*s && (*s!=' ') && (*s!='\t')) // remove ' ' and '\t'.
        s++;
        if (*s=='\0'){// end of line
            argv[args] = 0;
            return args;
        }
        *s++ = '\0';
    }
    return args;
}  // GetArgs
```

입력된 명령어로 부터 인수를 추출하는 함수

CLI - 명령어 처리 메인 함수(1)

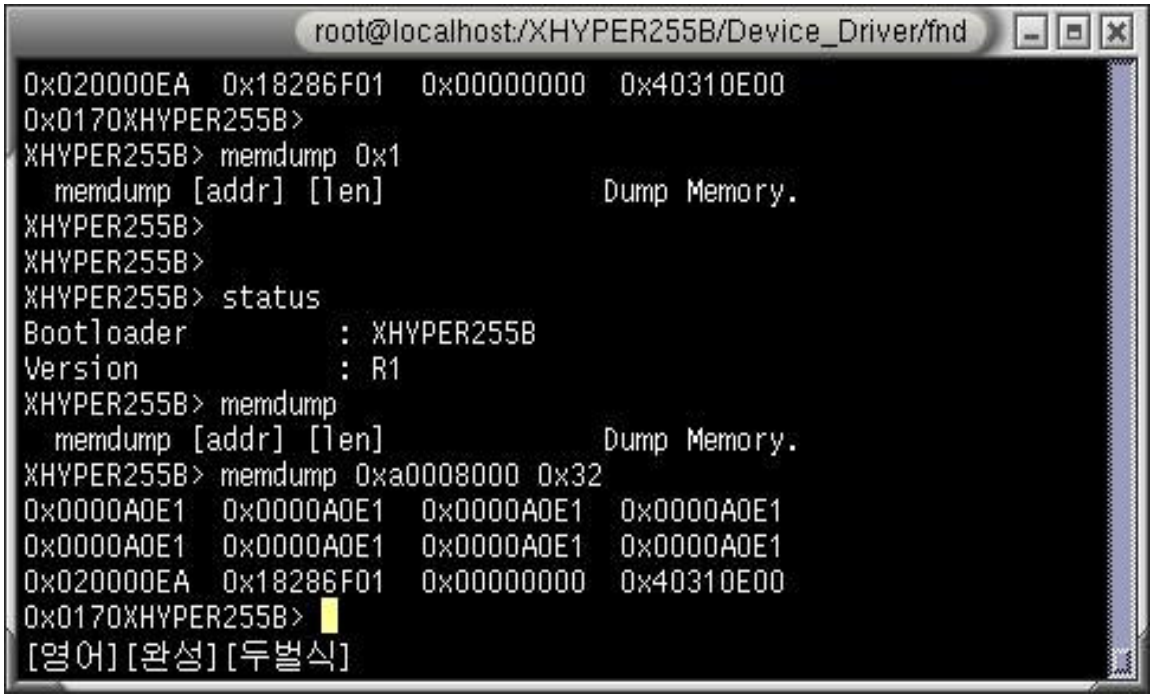
사용자로 부터 하나의 명령을 처리하는 main.c 함수

```
……..
……..
      // Key was pressed, so proceed command mode.
printf("\nAutoboot aborted\n");
printf("Type \"help\" to get a list of commands\n");

// the command loop. endless, of course.
   for(;;) {
      DisplayPrompt(NULL); // wait an hour to get a command.
      GetCommand(cmd, 128, 3600);
      if (!cmd || !cmd[0]) continue;
      argc = GetArgs(cmd, argv);
      for (cptr=cmdTbl; cptr->cmd; cptr++){
         if (!StrCmp(argv[0], cptr->cmd)){
            (cptr->run)(cptr, argc, argv);
                break;
         }
      }
      if (!StrCmp(argv[0], "help") || !StrCmp(argv[0], "?")){
         DoPrintfHelp(argc, argv);
      } else if (!(cptr->cmd)){
         printf("\tUnknown command : %s\n", argv[0]);
      }
   }
} // Cmain.
```

CLI에 시스템 동작 상태 확인(1)

▣ 메모리 read



CLI에 시스템 동작 상태 확인(2)

- **Write** 명령어에 의한 **7 segment** 동작 상황 확인