Embedded 1st HW. Calculator on Board with IPC

(설계 프로젝트 수행 결과)

과목명: [CSE4116] 임베디드시스템소프트웨어
담당교수: 서강대학교 컴퓨터공학과 박 성 용

학번 및 이름: 20121608, 오병수
개발기간: 2017.03.25.토 - 2017.04.07.금

최 종 보 고 서

I. 개발 목표

디바이스 컨트롤과 IPC를 이용하여 주어진 Clock, Counter, Text editor, Draw board를 구현한다.

II. 개발 범위 및 내용

가. 개발 범위

1. 입력 받기
2. Clock 모드
3. Counter 모드
4. Text editor 모드
5. Draw board 모드
6. 추가 구현

나. 개발 내용

1. 입력 받기
   - key 입력 / switch 입력 받아서 main process에게 shared memory를 통해서 넘겨주는 기능
2. Clock 모드
   - 시간 출력
   - 시간 변경 모드로 진입하여 시간, 분 변경
3. Counter 모드
   - 카운팅된 숫자 출력
   - 카운트되는 숫자의 진수를 변경하는 기능
4. Text editor 모드
   - 알파벳 / 숫자 입력 모드 변경하면서 글자 입력
   - space 출력
5. Draw board 모드
   - switch를 통해 커서를 움직이며 dot matrix의 값 변경하는 기능
   - 커서 켜기/끄기
6. 추가 구현

III. 추진 일정 및 개발 방법

가. 추진 일정

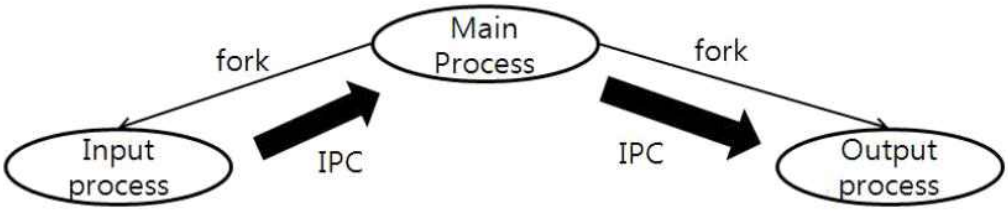
- 3/25(토): 프로그램 명세서 분석 및 전체 흐름 구상
- 3/26(일) ~ 4/5(수): 프로그램 구현, 테스트 및 디버깅
- 4/6(목) ~ 4/7(금): 보고서 작성, 전체 프로그램 테스트 및 디버깅

나. 개발 방법

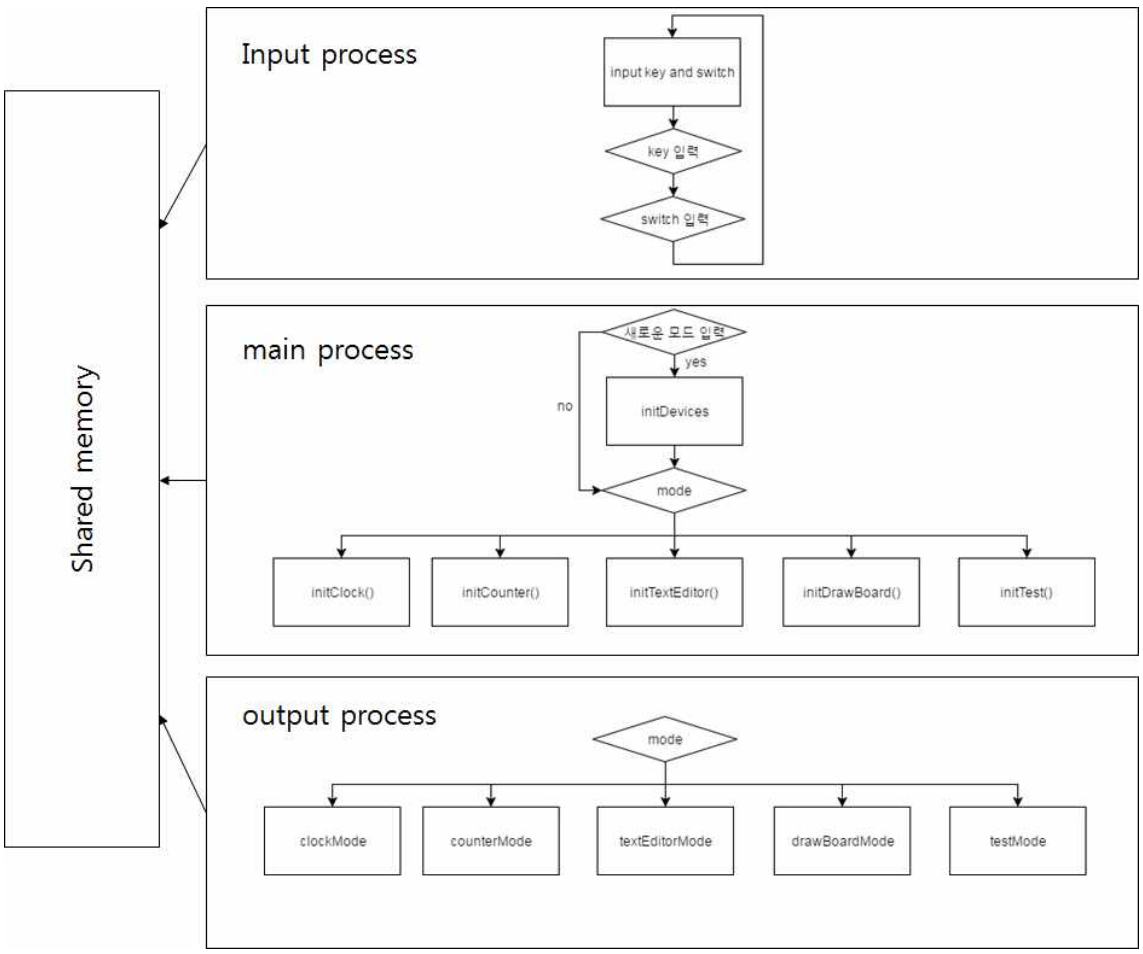
- Linux 환경에서 vim 에디터를 통해 프로그램 작성
- Cross-compilation 활용:
  - Linux: vim에서 코드 작성 후 ARM 컴파일 하여 보드의 sdcard로 usb port를 통해서 전송
  - Huins board: - Linux terminal에서 minicom 명령어를 통해, Linux 컴퓨터와 serial port로 연결된 보드를 host computer에 띄운다. 우선 보드의 kernel 영역에 device driver module을 insmod 시키고, host machine에서 usb port를 통해 전송한 기계어 코드를 보드에서 실행시킨다.

IV. 연구 결과

1. 합성 내용:



[그림 1] 프로세스 간의 구조 및 IPC 채널 방향



[그림 2] 전체 프로그램 구성도

2. 제작 내용: 개발 결과

- 구현한 컴포넌트들에 대한 역할 및 구현 방법에 관한 내용을 기술할 것. (예: 자료구조, 알고리즘).

□ IPC 방식

<shared memory 사용>

- shared memory를 physical memory에 할당

```
// Initialize shared memory
if((shmid = shmget(IPC_PRIVATE, 1024, IPC_CREAT|0666)) == -1) {
        puts("Failed to create a shared memory (shmget)");
        return -1;
}
```

- physical memory에 할당된 shared memory 공간을 virtual memory에 mapping

```
/* Map shared memory segment to virtual address space */
if((shmaddr = (int*)shmat(shmid, (int *)NULL, 0)) == (void*)-1) {
        puts("Failed to create a shared memory (shmat)");
        return -1;
}
```

- shared memory에 저장된 데이터 종류

shmaddr index = i	shmaddr content = shmaddr[i]	process 이동 방향
[0]	mode	input -> main
[1]	눌린 switch의 개수	input -> main
[2..10]	switch(1) ~ switch(9)이 눌렸는지 아닌지 여부 (0 or 1)	input -> main
[11]	mode	main -> output
[12]	clock 시간 수정 여부 (0 or 1)	main -> output
[13]	현재 출력해야 하는 clock 시간	main -> output
[14]	현재 진법	main -> output
[15]	counter	main -> output
[16]	<Text editor> switch 눌린 횟수 누적 양	main -> output
[17..49]	Text LCD에 출력하는 string의 버퍼	main -> output
[52]	<Text editor> 알파벳인지 여부	main -> output
[53]	dot matrix의 커서 깜빡일지 여부	main -> output
[54..123]	dot matrix에 출력할 데이터의 버퍼	main -> output
[124]	커서의 row	main -> output
[125]	커서의 column	main -> output
[126]	<Draw board> switch 눌린 횟수 누적 양	main -> output
[127]	<추가 구현> FND에 써줄 값	main -> output
[128]	<추가 구현> LED에 써줄 값	main -> output
[129]	<추가 구현> dot matrix에 대한 flag	main -> output
[130]	<추가 구현> 눌린 switch 번호	main -> output

□ 초기화 기능: 모드가 변경되면, 각 모드의 초기상태가 되어야 하고, 현재 사용 중인 모드를 제외한 다른 모드에서 사용하는 디바이스들은 모두 초기화시키는 작업을 수행해야 한다. 이러한 기능은 initDevices() 함수에서 수행한다. input process가 key 입력을 받아서 main process에게 모드 정보를 넘겨주면, main process에서 현재 입력된 mode가 바로 이전에 입력된 mode와 일치하는지 비교하여 모드가 변경되었음을 감지했을 때 initDevices()가 호출된다.

□ 입력 받기

- 부모 프로세스의 첫 번째 자식 프로세스인 input process에서 key와 switch device를 open하여 프로그램이 종료될 때까지 asynchronous(non-blocking)하게 입력을 받아서 결과를 shared memory에 저장한다.

```
devSW = open("/dev/fpga_push_switch", O_RDWR);  // open device file of switch with "nonblocking" mode
devKey = open("/dev/input/event0", O_RDONLY | O_NONBLOCK);      // open device file of key with "nonblocking" mode
```

key 입력을 받기 위해서 /dev/input/event0 라는 device file을 통해서 open을 수행하는데, 이 때 옵션에 O_NONBLOCK을 명시해 주어야 non-blocking 방식으로 key 입력을 받을 수 있다.

- key 입력 받기

```
/* read from keys without blocking: ?? implemented as interrupt(NOT polling) */
size = sizeof(struct input_event);
int rd = read(devKey, ev, size * BUFF_SIZE);
if(rd == -1)    continue;
keyPressed = ev[0].value;
input = ev[0].code;
if(!keyPressed || input == PROG_KEY)    continue;
if(input == END_KEY) {  // terminating condition
        shmaddr[0] = -1;
        puts("[input process] Good bye!");
        usleep(400000);
        break;
}
```

- switch 입력 받기

```
/* read from switches without blocking: ?? implemented as interrupt(NOT polling) */
buff_size = sizeof(push_sw_buff);
usleep(200000);
read(devSW, &push_sw_buff, buff_size);
```

□ Clock 모드

- main process에서 수행하는 연산

clock mode가 입력되면 main process에서는 현재 입력된 switch를 바탕으로 현재 시간과 시간 수정 모드인지 여부를 결정하여 이 결과를 output process에게 넘겨주는 기능을 수행한다. clock mode에 대한 연산을 수행하는 initClock() 함수의 입력 및 출력은 다음과 같이 정리할 수 있다.

- input: 현재 입력된 switch에 대한 정보 (switch(1) ~ (4) 입력에 대한 연산만 처리)
- output: 현재 시간을 초로 환산한 값(time_t 형식), 시간 수정 모드인지 여부

```
void initClock() {
        //static int isModifying = 0;
        //time_t second = time(NULL);
        int switchPressed = (shmaddr[1]) ? 1 : 0;
        second = shmaddr[13];

        if(!isModifying) {      // switch(1) is not pressed
                //printf("[initClock] shmaddr[1]: %d,  shmaddr[2]: %d\n", shmaddr[1], shmaddr[2]);
                if(switchPressed && shmaddr[2] == 1) {  // when switch(1) is pressed for the first time
                        shmaddr[12] = isModifying = 1;
                        //printf("[initClock] isModifying: %d, shmaddr[12]: %d\n", isModifying, shmaddr[12]);
                }
                second++;
        }
        else {  // switch(1) is once pressed. i.e. modifying mode is on
                if(switchPressed) {
                        if(shmaddr[3] == 1)             // reset time to the board's current time
                                second = time(NULL);
                        else if(shmaddr[4] == 1)        // add 3600s(= 1 hour) to the current second
                                second += 3600;
                        else if(shmaddr[5] == 1)        // add 60s(= 1 min) to the current second
                                second += 60;
                        else if(shmaddr[2] == 1)        // modifying mode is over -> clear 'isModifying'
                                shmaddr[12] = isModifying = 0;
                }
                else {
            // if user does not press any switch after he pressed switch(1),
            // the current time still needs to elapse
                        second++;
                }
        }
        shmaddr[13] = second;   // store the calculated second in the assigned shared memory
}
```

시간 수정 모드인지 아닌지의 여부에 따라서 flow가 달라진다.

- output process에서 출력하는 방식

output process에서도 main process에서 연산을 수행한 것과 유사한 방식으로 flow가 진행된다. 크게 시간 수정 모드인지 아닌지에 따라서 흐름이 구분되는데, 시간 수정모드가 아닌 경우에는 (1)번 LED(값 = 128)만 켤 수 있도록 하고 (3), (4)번 LED가 깜빡이고 있었다면(시간 수정모드에서 전환된 직후인 경우) 깜빡이는 작업을 중지할 수 있도록 코드를 구성했다. 반면 시간 수정모드인 경우, 수정모드로 전환된 직후에는 (3), (4)번 LED에 동시에 불이 들어오게 하고 깜빡임을 on/off 하는지 관리하는 변수인 blinkOn을 1로 set한다. 이 과정 이후에

수정모드에서는 dataLED의 값은 32와 16을 번갈아 이동하면서 LED의 (3), (4)번이 1초 간격으로 깜빡일 수 있 수 있게 된다.

```
/* depending on whether SW(1) is pressed once, decide LED value */
if(!isModifying) {
        dataLED = 128;
        blinkOn = 0;    // should be initialized to 0
}
else if(isModifying == 1) {
        if(!blinkOn) {  // right after SW(1) is pressed, turn both LED(3) and LED(4) on
                dataLED = 48;
                blinkOn = 1;
        }
        else {  // after a moment, alternate between LED(3) and LED(4)
                dataLED = (!alternateLED) ? 32 : 16;
                alternateLED = -1 * alternateLED + 1;   // 0 -> 1, 1 -> 0
        }
}
```

☐ Counter 모드

- (주의) 다음 사항에 대해서는 명세서에서 명확하게 명시된 내용이 없어서 스스로 기준을 정해서 구현했다.

1. 카운트 횟수가 10진수로는 세 개 자릿수(000-999)로 출력 가능하지만 다른 진수 체계로는 이를 넘어가는 경우 최대 4자리(최하위 digit 4개)까지 출력할 수 있도록 프로그램을 구현했다.

- main process에서 수행하는 연산

counter mode가 입력되면 main process에서는 현재 입력된 switch를 바탕으로 현재 진법, count 횟수(10진수)를 결정하여 이 결과를 output process에게 넘겨주는 기능을 수행한다. counter mode에 대한 연산을 수행하는 initCounter() 함수의 입력 및 출력은 다음과 같이 정리할 수 있다.

- input: 현재 입력된 switch에 대한 정보 (switch(1) ~ (4) 입력에 대한 연산만 처리)
- output: 현재 진법, count 횟수

```
void initCounter() {
        int sw2 = shmaddr[3], sw3 = shmaddr[4], sw4 = shmaddr[5];       // switch(2),(3),(4): number increment
        int sw1 = shmaddr[2];   // switch(1): converting number system
        int sumToAdd;
        int base;

        switch(numSystem) {
                case NUM_DEC:  base = 10;       break;
                case NUM_OCT:  base = 8;        break;
                case NUM_FOUR: base = 4;        break;
                case NUM_BIN:  base = 2;        break;
        }
        if(sw1) {       // convert the number system
                shmaddr[14] = numSystem = (numSystem+1) % 4;
        }
        sumToAdd = (base*base) * sw2 + base * sw3 + sw4;
        counter = (counter + sumToAdd) % 1000;
        shmaddr[15] = counter;
}
```

- output process에서 출력하는 방식

```
void counterMode() {
        unsigned char dataFND[4], dataLED;
        char strCounter[5];
        int base, convertedCounter;
        unsigned char fpga_set_blank[10] = {
                0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
        };

        numSystem = shmaddr[14];
        counter = shmaddr[15];

        // initialize Dot matrix
        write(devDOT, fpga_set_blank, sizeof(fpga_set_blank));

        /* LED */
        switch(numSystem) {
                case NUM_DEC:  dataLED = 64;  base = 10;  break;
                case NUM_OCT:  dataLED = 32;  base = 8;   break;
                case NUM_FOUR: dataLED = 16;  base = 4;   break;
                case NUM_BIN:  dataLED = 128; base = 2;   break;
        }
        write(devLED, &dataLED, 1);

        /* FND */
        printf("counter: %d (in decimal)\n", counter);
        convertedCounter = convertNumSystem(base);
        sprintf(strCounter, "%04d", convertedCounter);
        printf("strCounter: %s\n", strCounter);
        dataFND[0] = strCounter[0]-'0';
        dataFND[1] = strCounter[1]-'0';
        dataFND[2] = strCounter[2]-'0';
        dataFND[3] = strCounter[3]-'0';
        write(devFND, &dataFND, 4);
}
```

main process의 counter 관련 연산 결과 output process가 입력으로 제공받은 현재 진법, 카운트 횟수를 바탕으로 FND에 카운트 횟수를 현재 진법에 맞게 출력하고 현재 진법에 따라 적절한 LED를 출력한다. FND에 출력하는 카운트 횟수는 10진수로 저장된 카운트 횟수 값을 numSystem에 따라 구분된 base 진법으로 변환하는 과정을 거쳐서 출력해야 한다. 10진수 값 counter를 base 변수의 값으로 변환하여 반환해주는 함수는 다음과 같이 정의할 수 있다.

```
/* convert decimal to base system */
int convertNumSystem(int base) {
        int ret = 0;
        int numBuf[20], cnt = 0;
        int tmp = counter, i;

        while(tmp) {
                numBuf[cnt++] = tmp % base;
                tmp = tmp / base;
        }
        for(i = cnt-1; i >= 0; i--) {
                ret = ret * 10 + numBuf[i];
        }
        return (ret % 10000);
}
```

□ Text editor 모드

- (주의) 다음 사항에 대해서는 명세서에서 명확하게 명시된 내용이 없어서 스스로 기준을 정해서 구현했다.

1. 알파벳 글자가 입력되는 방식: 한 switch를 계속해서 누르면 해당 switch에 해당하는 알파벳들이 순차적으로 바뀌고, 여기서 다른 switch를 입력하거나 space에 해당하는 (8), (9)번 switch를 동시에 누르는 등의 입력이 들어오면 커서를 다음으로 이동시켜 해당 알파벳/문자를 출력하는 방식으로 구현했다.

2. 한 알파벳을 입력하다가 (5), (6)를 눌러서 숫자모드로 전환했다가 다시 (5), (6)를 눌러서 알파벳 모드로 돌아오는 경우 항상 현재 입력된 switch의 첫 번째 알파벳이 입력되도록 구현했다.

예:

switch (2) -> (2) - LCD: "B"

switch (5),(6) -> (2) - LCD: "BA"

3. FND에 출력하는 카운트 횟수는 현재까지 입력된 모든 switch(두 개가 동시에 입력된 경우, 둘 다 카운트 횟수에 포함)를 누적한 값으로 결정했다. 즉, switch (2), (3)를 동시에 입력하여 clear를 수행해도 현재 FND에 출력되는 카운트 횟수는 0이 되지 않고 계속해서 값을 누적해 나간다.

- main process에서 수행하는 연산

text editor mode가 입력되면 main process에서는 현재 입력된 switch를 바탕으로 text LCD에 출력해야 하는 string 정보와 현재 입력되는 문자가 알파벳인지 숫자인지를 나타내는 정보를 output process에게 넘겨주는 기능을 수행한다. text editor mode에 대한 연산을 수행하는 initTextEditor() 함수의 입력 및 출력은 다음과 같이 정리할 수 있다.

- input: 현재 입력된 switch에 대한 정보 (switch(1) ~ (9) 입력에 대한 연산만 처리)
  (이 모드의 경우 switch가 동시에 두 개까지 입력될 수 있다.)
- output: text LCD에 출력해야 하는 string 정보와 현재 입력되는 문자가 알파벳인지 숫자인지를 나타내는 정보

- output process에서 출력하는 방식

main process로부터 넘겨받은 string과 현재 입력되는 문자가 알파벳인지 숫자인지 여부를 나타내는 flag 변수를 이용해 text LCD, dot matrix에 적절한 값을 넣어 출력한다.

<dot matrix API 사용방법>

```
unsigned char fpga_dot[2][10] = {
        {0x1c, 0x36, 0x63, 0x63, 0x63, 0x7f, 0x7f, 0x63, 0x63, 0x63}, // A
        {0x0c, 0x1c, 0x1c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x1e}  // 1
};
```

여기서 선언된 fpga_dot[0]에 저장된 데이터는 'A' 문자를 dot matrix에 출력해주기 위해서 필요한 bit 정보를 저장하고 있다. 즉 fpga_dot[0][0..9] 각각에 저장하고 있는 값은 dot matrix의 각 row에 써줘야 하는 bit stream을 16진수로 packing한 정보이다.
이렇게 선언된 data를 다음과 같이 write() 함수를 통해 출력하면 dat matrix에 'A'를 명시한 dot에 출력할 수 있다.

```
// dot matrix
if(alphabetMode)
        write(devDOT, fpga_dot[0], sizeDOT);
else
        write(devDOT, fpga_dot[1], sizeDOT);
```

□ Draw board 모드

- (주의) 다음 사항에 대해서는 명세서에서 명확하게 명시된 내용이 없어서 스스로 기준을 정해서 구현했다.

1. dot matrix의 가장자리에 커서가 위치하고 있을 때 index 범위(row: [0, 9], column: [0, 6])를 벗어나는 위치로 커서를 이동시키려고 하는 입력에 대한 작동: 반대 방향으로 커서가 움직일 수 있도록 구현했다.
예: (0, 0) 위치에 커서가 있을 때 switch (2)번을 누르면 (9, 0) 위치로 커서가 이동한다.

2. switch (5)번을 눌러서 '선택'을 했을 때 해당 위치의 dot matrix의 버퍼 값이 0이었으면 1로 만들고, 1이었으면 0으로 바뀌도록 구현했다.

3. 커서의 위치가 (r, c)일 때 해당 위치의 dot matrix의 값을 1로 만들면(커서는 이동시키지 않고) 커서가 깜빡이지 않고 현재 위치의 dot matrix의 상태를 그대로 유지하도록 구현했다.

4. switch (9)번 입력의 경우, 색깔 반전(비트 반전)이 되도록 구현했다. 즉, 현재까지 dot matrix에 입력된 bit pattern에서 0은 1로, 1은 0으로 바꾸는 XOR 연산을 수행했다.

- main process에서 수행하는 연산

Draw board mode가 입력되면 main process에서는 현재 입력된 switch를 바탕으로 커서를 깜빡일지 여부, dot matrix에 출력해야 하는 데이터의 버퍼, 현재 커서의 위치 등에 대한 정보를 결정하여 이 결과를 output process에게 넘겨주는 기능을 수행한다. Draw board mode에 대한 연산을 수행하는 initDrawBoard() 함수의 입력 및 출력은 다음과 같이 정리할 수 있다.

- input: 현재 입력된 switch에 대한 정보 (switch(1) ~ (9) 입력에 대한 연산만 처리)
- output: 현재 입력된 switch를 바탕으로 커서를 깜빡일지 여부, dot matrix에 출력해야 하는 데이터의 버퍼, 현재 커서의 위치, 현재까지 입력된 switch의 개수 등

- output process에서 출력하는 방식

main process로부터 받은 현재 커서의 위치, 커서를 깜빡일지 여부, dot matrix에 출력해야 하는 데이터 등의 정보에 따라서 dot matrix, FND에 적절하게 출력하는 작업을 수행한다.

dot matrix에 10x7 행렬에 저장된 0, 1 패턴을 packing하여 10개의 1차원 배열에 넣기 위해서는 다음과 같은 연산을 해야 한다.

```
for(r = 0; r < 10; r++) {
        /*for(c = 0; c < 7; c++) {
                printf("[%d][%d]: %d ", r, c, tmp[r][c]);
        }
        puts("");*/
        hex1 = 4*tmp[r][0]+2*tmp[r][1]+tmp[r][2];
        hex2 = 8*tmp[r][3]+4*tmp[r][4]+2*tmp[r][5]+tmp[r][6];

        //printf("[%d] hex1: %d, hex2: %d\n", r, hex1, hex2);
        //printf("[%d] hex1*16+hex2 = %d\n", r, hex1*16+hex2);
        //fpga_dot[r] = (4*tmp[r][0]+2*tmp[r][1]+tmp[r][2])*16
        //                        +(8*tmp[r][3]+4*tmp[r][4]+2*tmp[r][5]+tmp[r][6]);
        fpga_dot[r] = hex1*16+hex2;
}
write(devDOT, fpga_dot, sizeof(fpga_dot));
```

이 코드에서 hex1의 경우, 최상위 비트를 제외한 나머지 3개의 상위 비트를 10진수로 변환한 값, hex2의 경우 최하위 4개 비트를 10진수로 변환한 값을 의미한다.

□ 추가 구현

- 기능 및 사용 방법

<Device Testing Application>

- **구현한 기능**: board의 여러 device들(FND, LED, text LCD, dot matrix, switch)이 제대로 작동하는지 편리하게 확인할 수 있는 기능을 구현했다.
- **활용한 device**: FND, LED(1)-(8), text LCD, dot matrix, switch(1)-(9)
- **사용 방법**

Device Testing Application은 총 4가지 모드로 구성된다.

- 모드 1: 전체 디바이스 일괄적으로 테스트
  - 각 디바이스의 모든 구성 요소를 ON 시키는 모드
- 모드 2: switch 작동 테스트
  - switch가 제대로 작동하는지 테스트하기 위한 모드
  - switch (1) - (9)이 눌렸으면 dot matrix에 해당 번호가 출력된다.
  - 이 모드로부터 빠져나오기 위한 switch 입력: 9번
- 모드 3: FND 작동 테스트
  - FND가 제대로 작동하는지 테스트하기 위한 모드
  - switch (1) - (9)이 눌렸으면 dot matrix에 해당 번호가 출력된다.
  - switch (1) - (9)을 눌러서 FND의 (1) - (4)번 위치의 모든 불을 켠다. (8을 출력)
  - 이 모드로부터 빠져나오기 위한 switch 입력: 9번
- 모드 4: LED 작동 테스트
  - switch (1) - (8)을 눌러서 LED의 (1) - (8)번 위치의 불을 켠다.
  - switch (1) - (8)이 눌렸으면 dot matrix에 해당 번호가 출력된다.
  - 이 모드로부터 빠져나오기 위한 switch 입력: 9번

- main process에서 수행하는 연산

device test mode가 입력되면 main process에서는 현재 입력된 switch를 바탕으로 디바이스들에게 넘겨줄 데이터 값을 output process에게 넘겨주는 기능을 수행한다. device test mode에 대한 연산을 수행하는 initTest() 함수의 입력 및 출력은 다음과 같이 정리할 수 있다.

- input: 현재 입력된 switch에 대한 정보 (switch() ~ () 입력에 대한 연산만 처리)
- output: 디바이스들에게 넘겨줄 데이터 값

- output process에서 출력하는 방식

main process로부터 받은 데이터를 활용하여 각각의 디바이스의 상태를 출력한다.

3. 시험 및 평가 내용:

- 평가 방법:

host PC에서 컴파일을 수행하여 생성된 ARM 기계어 코드를 보드에게 전송하여 실행시켰다. 각각의 모드에 따라서 입력 스위치의 값을 다양하게 변화시키면서 목적에 부합하게 작동하는지 device의 상태를 통해서 확인할 수 있었다.

- 보건 및 안정, 생산성과 내구성

1. 프로그램의 안정성을 위해서 적절하게 초기화 작업을 수행했다. 입력된 모드가 변경되면 항상 initDevices()가 가장 먼저 수행되면서 필요한 device 외의 device는 꺼진 상태로 만들고, 현재 모드에서 필요로 하는 디바이스도 문제 요구사항에 맞게 적절하게 초기화 함으로써 프로그램 수행이 올바로 이루어지도록 프로그램을 구혀했다.

2. file system call(open, read, write, close 등) / shared memory 할당 및 mapping하는 과정에서 에러가 발생하는 경우에 대처할 수 있는 루틴을 조건문을 통해서 처리했다.

V. 기타

- 기타 관련 내용을 기술할 것.

1. 연구 조원 기여도: 오병수 100%

2. 기타 본 설계 프로젝트를 수행하면서 느낀 점을 요약하여 기술하라. 내용은 어떤 것이든 상관이 없으며, 본 프로젝트에 대한 문제점 제시 및 제안을 포함하여 자유롭게 기술할 것.

이번 프로젝트를 통해서 이론으로 알고 있던 내용을 실제 구현하는 과정의 어려움을 이해할 수 있었다. 운영체제 시간에 이론적으로 배운 fork(), Interprocess Communication 등을 실제로 구현하기 위해서 여러 가지 시행착오를 거쳐야 했다. 가령, 어떤 프로세스가 어떤 순서로 수행되는지를 정확히 몰라서 printf()를 각 프로세스에서 직접 출력해보면서 프로그램의 틀을 잡아나갈 수 있었다. 또한 입력을 blocking으로 해야 할지 non-blocking으로 해야 하는지 등의 design 이슈에 대해서도 고민하면서 프로그램 과제를 수행해야 했다.

그리고 처음으로 여러 가지 device들을 직접 제어하면서 결과를 눈으로 확인하는 과정은 무척 흥미로웠다. 이제까지 학교에서 프로젝트를 수행하면서는 콘솔, 파일 입출력밖에 제어를 해보지 않았는데 다양한 device를 system call을 통해서 제어하는 과정을 통해서 device control 원리에 대해서 정확히 이해할 수 있게 되었다.