**Embedded System Software HW#1**

**Document**

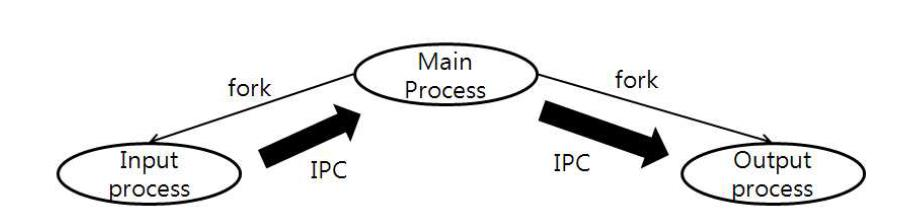
**20131540**

**김준호**

**1. 목표**

**디바이스 컨트롤과 IPC를 이용하여 주어진 Clock, Counter, Text Editor, Draw Board를 구현한다.**

**2. 구현**

****

**Main process는 Input, Output Process를 하나씩 fork한다. 본 프로젝트는 총 3개의 process로 구성되며 process 간 통신은 IPC 중 Massage queue를 이용한다.**

**typedef struct {**

**long mtype;**

**int buff[256];**

**}msg, \*msgPointer**

정의한 massge structure

**Massage는 위와 같은 구조이고 massage를 send할 땐 msgsnd함수를, recieve할 땐**

**msgrcv함수를 사용했다.**

**massage안의 buffer의 사용 방법은 다음과 같다.**

**- buff[0] : Event Key 입력 값 ( 각 button이 눌렸을 때 return되는 정수값은 main에 define 되어있다.)**

**- buff[1 ~ 9] : 각 SW key 입력 값 ( 각 button이 눌리면 해당 index 값이 0에서 1로 바뀜 )**

**- buff[10] : mode**

**- buff[11] : LED 값**

**- buff[12 ~ 15] : 각 FND 값 ( 0 ~ 9 )**

**- buff[16 ~ 47] : LCD에 띄우는 character들**

**- buff[48] : output.c에 정의되어 있는 DotInfo의 index값 ( 0, 1, 2 )**

**- buff[49 ~ 58] : Dot Maxtrix 의 값들 (10 lines)**

**- buff[59 ~ ] : unused**

**buff[0 ~ 9]는 input과 main process가 사용하며 buff[10 ~ 58]는 output과 main process가 사용한다.**

**- Main Process :**

**main이 실행되면 먼저 process 하나를 fork한 후에 input을 실행시킨다. 이후 다른 하나의 process를 fork한 후에 output을 실행시킨다.**

**main은 2개 type의 massage queue를 사용하며 하나는 input과 통신(recieve only)하는 massage queue (type 1001)이고 다른 하나는 output과 통신(send only)하는 massage queue (type 1002) 이다. type 번호는 massage queue에서 사용하는 key값과 동일하며 queue id는 msgget 함수에 각각의 key값을 전달해 얻은 값을 사용한다.**

**main함수에는 BACK button이 입력될 때 까지 계속 도는 while loop이 존재하고 while loop 안에서 input massage를 recieve하도록 설계했다. input으로 부터 massage를 받으면 구현한 handler들을 통해 현재 모드에 맞게 해당 button입력에 대한 처리를 한다. 처리가 끝나면 변경된 값들을 massage의 buffer에 저장하여 output process에 send한다.**

**while loop이 break되면 fork했던 두 child process를 종료(KILL)시키고 두 process의 종료가 끝나면 main이 종료된다.**

**- Input Process :**

**main으로 부터 fork된 process로 device의 button 입력이 발생하면 해당 button값을 read함수를 통해 리눅스의 input event handler의 structure로 읽어온다. SW button의 경우 제공된 fpga\_push\_switch driver를 사용하여 read함수를 통해 읽어온다.**

**이후 해당 button을 index값으로 하여 massage의 buffer를 0에서 1로 수정하고 main process로 send한다.**

**- Output Process :**

**main으로 부터 fork된 process로 main에서 massage를 받으면 먼저 massage의 buffer에 저장된 값으로 현재 mode를 결정하고 각 mode에 맞게 buffer에 저장된 정보들을 device file에 넘겨준다. (LED device의 경우 device driver를 사용하지 않고 mmap함수를 이용)**

**- Device Driver 사용 예시 :**

**<output.c>**

**void writeLcd()**

**{**

**char \*filename\_lcd = "/dev/fpga\_text\_lcd";**

**//LCD Device File name**

**int fd = open(filename\_lcd, O\_WRONLY);**

**//open device file**

**ssize\_t res;**

**//store a result of write function**

**if( fd < 0 ){**

**perror("File Open Error");**

**exit(1);**

**}**

**//handling open error**

**res = write(fd, str, 32);**

**//write 32 characters in str to Lcd Device File**

**//str -> global variable in output.c**

**if( res < 0 ){**

**perror("File Write Error'");**

**exit(1);**

**}**

**//handling write error**

**close(fd);**

**//close device file**

**}**

**- MMAP 사용 예시 :**

**<output.c>**

**void writeLed(int n)**

**{**

**char \*filename\_mem = "/dev/mem";**

**//Memory device file**

**int fd;**

**unsigned long \*base\_addr = 0;**

**//logical address which map to physical starting address of Led Device memory**

**unsigned char \*target\_addr = 0;**

**//logical address which map to physical target address of Led Device memory**

**//want to write n in target address**

**fd = open(filename\_mem, O\_RDWR | O\_SYNC);**

**//open memory device file**

**if( fd < 0 ){**

**perror("File Open Error");**

**exit(1);**

**}**

**//handling open error**

**if( n > 255 || n < 0 ){**

**printf("Long Led Data");**

**close(fd);**

**return ;**

**}**

**//verify input data**

**base\_addr = (unsigned long\*)mmap(NULL, 4096, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, LED\_BASE\_ADDR);**

**//mapping logical address to physical address**

**if( base\_addr == MAP\_FAILED ){**

**printf("MMAP Error");**

**close(fd);**

**return ;**

**}**

**//handling mmap error**

**target\_addr = (unsigned char\*)((void\*)base\_addr + LED\_ADDR);**

**//calculate target address**

**\*target\_addr = n;**

**//write n in target address**

**munmap(base\_addr, 4096);**

**//release memory mapping**

**close(fd);**

**//close memory device file**

**}**

**각 mode와 Device에 맞는 ouput handler는 output.c에 정의되어 있으며 각 기능이 주석에 기술되어 있으므로 나머지에 대해서 document에선 생략한다.**

**3. 기능**

**각 모드의 processing handler들을 main.c에 정의된 함수들 중 mode1, mode2, mode3, mode4에 해당하며 각 함수에 대한 설명은 main.c의 주석에 기술 했으므로 document에선 생략한다.**

**3.1. Clock - 모드 1 (Default Mode)**

**- FND : 시간을 출력한다.(앞 두 자리는 시간(24시간), 뒤 두 자리는 분(60분)). 초기**

**상태는 보드의 시간.**

**- LED : 초기 상태는 (1)번 LED 에만 불이 들어온 상태. 보드 시간을 변경하기**

**시작하면 (3)번 (4)번 LED에 불이 들어오고 1초에 하나씩 번갈아 가면서 불이**

**들어오게 한다.(한 버튼을 1초씩 켰다 껐다를 반복) 보드 시간 변경이 끝나면**

**(1)번 LED 에만 불이 들어오게 한다.**

**- SW(1) : 보드의 시간을 변경할 수 있도록 하고,변경된 시간으로 저장하는 버튼.**

**SW(2) :보드의 시간으로 reset시켜주는 버튼. 시간 변경 시 사용.**

**SW(3) :시간을 1시간 증가 시키는 버튼.**

**SW(4) : 분을 1분 증가 시키는 버튼.**

**3.2. Counter - 모드 2**

**- FND : 카운팅된 숫자를 출력한다.십진수의 숫자는 끝의 세 자리에만 입력된다.초기상태는 0000. 999를 넘어갈 경우, 천단위는 생략한다(ex. 1000→ 0000만 출력).**

**각 자리의 9에서 1이 증가하면, 앞자리의 숫자가 증가한다(ex.0090→ 0100).**

**SW(1)를 누르면 십진수로 입력된 수가 진수변환을 한다.**

**- LED : 초기 상태는 (2)번 LED 에만 불이 들어온 상태. 각 LED는 진수를 나타낸다.**

**10진수는 (2)번, 8진수는 (3)번, 4진수는 (4)번, 2진수는 (1)번 LED이다. 진수가**

**바뀔 때마다 LED 또한 바뀐다.**

**- SW(1) : 진수 변환을 하는 버튼.그 순서는 십진수, 8진수, 4진수, 2진수이다.**

**SW(2) :백의 자리 숫자를 1 증가시키는 버튼**

**SW(3) :십의 자리 숫자를 1 증가시키는 버튼.**

**SW(4) : 일의 자리 숫자를 1 증가시키는 버튼.**

**- EX) 다른 진수일 때도 SW(2)~SW(3)키를 누르면 각 자리수는 다음과 같이**

**증가합니다. 8진수 : 123 -> SW(2) -> 223**

**EX) 다른 진수일 때도 증가에 의해 앞의 자리수가 증가할 수 있음.**

**8진수 : 127 -> SW(4) -> 130**

**3.3. Text Editor - 모드 3**

**- FND : fpga의 Switch가 현재 text를 입력하기 위해 몇 번 눌렸는지 count한 값을**

**출력한다. 초기 상태는 0000. 9999를 넘어갈 경우, 만 단위는 생략한다. (ex.**

**10000→0000만 출력)**

**- SW : 알파벳과 숫자를 입력 받는 버튼.**

**초기 상태는 알파벳 입력. 새로운 버튼의 입력이 한 번 들어올 때마다 한 글자씩**

**출력한다. 한번 눌렀던 버튼을 다시 누를 때마다 해당 알파벳을 입력 수에 맞게**

**바꾸어준다. 즉 (2)번 버튼을 1번 누르면 A를 출력하고, (2)번 버튼을 3번 입력하면 C를출력한다.(8),(9)번 버튼의 입력이 한꺼번에 들어오지 않으면 text LCD에 출력되는**

**string은 항상 덧붙여 출력한다.**

**(5), (6)번 버튼을 한꺼번에 누르면 알파벳 입력에서 숫자 입력으로 바꾼다. text LCD에**

**출력되는 string의 값은 변하지 않고, 기존에 입력된 text에 덧붙여 입력한다. 한 번**

**누를 때마다 해당하는 숫자를 출력한다. 숫자 입력 시에는 같은 버튼을 여러 번**

**누르면 버튼을 누를 때마다 새로운 숫자를 출력한다.**

**ex ) AMXB : (2)(6)(9)(2)(2)**

**(2) (2) (2) (2) (5) (5) (4) (3) (1) (1) : AKGDQ**

**버튼 2개를 동시에 누를 때, 각 기능을 수행한다(FND count에 포함).**

**(2), (3) : text LCD clear. 이 입력이 들어오면 text LCD에 있던 값을 없애고 다시 빈**

**상태로 만들어준다.**

**(5), (6) : 영어 → 숫자, 숫자 → 영어의 입력을 바꿔준다.**

**(8), (9) : 한 칸 띄운다.**

**- Text LCD : 스위치를 통해 생성된 text 값을 출력한다. 초기 상태는 LCD가 빈 상태.**

**Text LCD의 최대 출력 범위를 넘어가면 기존의 string에 가장 앞에 있던**

**문자를 제거하고 한 칸씩 앞으로 밀고 나서, 새로 들어온 text를 출력한다.**

**text LCD의 최대 출력 범위를 8이라고 가정하고 기존의 text를 ABCDEFGH라할**

**때, 9번째 text인 W가 들어오면 BCDEFGHW만 출력한다.**

**- Dot Matrix : 현재 switch로 받는 입력이 알파벳인지 숫자인지 나타낸다. 초기 상태는**

**A. 입력 받는 값이 알파벳이면 Dot Matrix에 A를, 입력 받는 값이 숫자면 Dot**

**Matrix에 1을 출력한다.**

**3.4. Draw Board - 모드 4**

**FND : fpga의 Switch가 몇 번 눌렸는지 count한 값을 출력한다. 초기 상태는 0000.**

**9999를 넘어갈 경우, 만 단위는 생략한다(ex. 10000→0000만 출력).**

**- SW : 스위치는 아래와 같은 기능을 담당.**

**SW(2), (4), (6), (8) : 방향키로 사용.커서를 이동.**

**SW(1) : 현재 그리고 있는 그림을 지우고 초기 상태로 리셋.**

**SW(3) : 커서 표시/숨기기 기능.**

**초기 상태는 커서를 왼쪽 첫 줄에 표시하여 깜빡이는 상태이다(1초마다**

**깜빡임).**

**SW(7) : 현재 그리고 있는 그림을 Clear. 현재 커서의 위치나 커서 표시등은 바뀌지**

**않음.**

**SW(9) : 현재 그림을 반전시킴.**

**- Dot Matrix : 그림을 출력함.초기 상태는 커서를 왼쪽 첫 줄에 표시하여 깜빡이는**

**상태(1초마다 깜빡임).**

**3.5. 추가구현**

**해당 없음**

**4. 기타**