**7주차 결과보고서**상징, 로고, 폰트, 원이(가) 표시된 사진

자동 생성된 설명

10조 (001분반)

과목명 : 임베디드시스템설계및실험

담당 교수 : 정상화 교수님

담당 조교 : 최호진 조교님

조원 : 201924603 하규승(조장)

201727102 강준혁

201924525 이광훈

202023139 박지원

제출 날짜 : 2023.10.01

**목차**

1. 실험 목표

2. 세부 목표

3. 실험 기구

4. 실험 과정

5. 실험 결과

6. 결과에 대한 논의

**1. 실험목표**

**1-1. interrupt와 Polling**

CPU와 입/출력 장치 사이에는 통신이 필요하고, 주된 통신 방법이 2가지가 있는데, 크게 인터럽트(Interrupt)와 폴링(Polling) 방식으로 나뉜다.

각 방식에 대해 설명하면, Interrupt 방식은 **H/W가 CPU에** 예외 상황이 발생하여 처리가 필요하다고 알려주는 방식이다. 반대로 Polling 방식은 **CPU가 H/W를** 주기적으로 감시하면서 예외 상황 발생 시 처리하는 방식이다.

각 방식에 대해 차이점은 다음과 같다. Interrupt는 CPU 외부에서 신호가 들어오는 방식이고, H/W로부터 신호가 들어오게 되면 CPU는 진행 중인 작업을 잠시 멈추고 Interrupt 처리 루틴을 실행하여 신호를 처리하게 된다. 반대로 Polling은 CPU가 H/W의 변화를 주기적으로 체크해야 하므로 CPU가 다른 일을 하는 도중에 신호를 읽을 수 없고, 이에 따라 문제가 생기더라도 CPU가 일단 작업을 시작했다면, 그 작업이 끝난 이후 오류를 발견하게 된다.

이러한 Interrupt 방식과 Polling 방식의 차이점으로 인해, Polling 방식은 오류가 일어난 시점에서 즉시 반응이 힘들고, 계속 해당 H/W에 대해 상태를 체크해야 하므로 레지스터를 계속 확인해 봐야 하므로 CPU의 점유율이 높다는 단점이 있다. 반면, Interrupt 방식에 비해 Polling 방식이 좀 더 구현이 쉽다는 장점이 있다.

**1-2. H/W Interrupt vs S/W Interrupt**

인터럽트는 발생시킨 원인에 따라 하드웨어 인터럽트 또는 소프트웨어 인터럽트로 구분된다.

하드웨어 인터럽트는 CPU 외부의 디스크나 주변장치로부터 발생하는 것이다. 하드웨어가 CPU의 서비스가 필요하거나, 운영체제의 attention을 요하는 상황을 알리기 위해 발생하며, 주로 전기적인 신호를 사용해 구현된다. 하드웨어 인터럽트 방식의 예시로는 버튼을 클릭했을 때 마우스에 의해 생성되는 인터럽트, 또는 읽기, 쓰기 작업이 완료될 때 디스크 드라이브에서 생성되는 인터럽트가 있다.

소프트웨어 인터럽트는 CPU 내부에서 자신이 실행한 명령이나 실행에 관련된 모듈이 변화하는 경우 발생한다. 크게 시스템 콜(System Call), 예외상황(Exception)의 2가지 상황으로 나뉜다. 시스템 콜은 커널 영역의 기능을 사용자 모드에 있는 애플리케이션이 사용 가능하게, 즉 프로세스가 하드웨어에 직접 접근해서 필요한 기능을 사용하기 위해 발생하는 인터럽트이다. 예시로는 File Manipulation System Call, Process Control System Call 등이 있다. 예외상황은 소프트웨어 내부적으로 오류가 발생하는 명령이 처리되었을 때 발생하는 인터럽트이다. 예시로는 존재하지 않는 메모리 주소에 접근하거나, 나눗셈 연산에서 0으로 나누려고 하는 상황 등이 있다.

**1-3. EXTI(External Interrupt / event controller)**

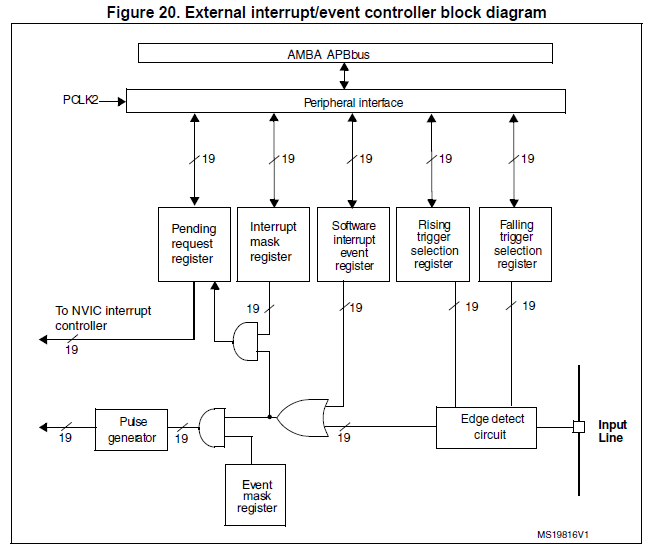
****

그림 1-3. EXTI diagram

외부 신호가 입력될 경우 디바이스는 interrupt 혹은 event가 발생한다.

Event는 pulse를 발생시켜 특정 기능을 수행하도록 하지만, Interrupt는 cpu 가 ISR 핸들러를 처리하게 한다.

외부 인터럽트/이벤트 컨트롤러는 인터럽트/이벤트 요청을 생성하는 데 사용되는 20개의 에지 감지기 라인으로 구성된다. 각 라인은 트리거 이벤트(상승 에지, 하강 에지, 둘 다)를 선택하도록 독립적으로 구성될 수 있으며 독립적으로 마스킹(masking)될 수 있다.

모든 GPIO핀들은 EXTI 라인을 통해 연결되어 있으며, 같은 번호의 핀은 같은 라인을 공유한다. 따라서, 동일한 EXTI에 연결된 핀들은 동시에 인터럽트 소스로 사용될 수 없다.

**1-4. NVIC(External Interrupt / event controller)**

NVIC는 Nested Vectored Interrupt Controller의 약어로, 중첩 벡터형 인터럽트 제어기라는 뜻이다. 여러 인터럽트가 동시에 들어왔을 때 이를 순차제어 하는 기능을 한다. STM32F4 시리즈의 모든 시스템 예외와 외부 인터럽트는 이 NVIC에서 처리한다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

그림 1-4. NVIC의 간략한 흐름도

각 ISR에 할당된 Preemption priority과 sub prioruty에 따라 우선순위가 결정된다. 더 높은 priority를 가진 IRQ가 들어오면 동작중인 ISR을 중지시키고 새 ISR을 수행한다. 이후 ISR이 끝나면, 기존의 ISR을 다시 실행한다.

**2. 세부 목표**

2-1. Datasheet 및 Reference Manual을 참고하여 해당 레지스터 및 주소에 대한 설정 이해

2-2. 스캐터 파일을 통해 플래시 메모리에 프로그램 다운로드

2-3. 플래시 메모리에 올려진 프로그램 정상적인 동작 확인

**3. 실험 기구**

* STM32F107 보드
* 브레드보드, 점프선
* 릴레이 모듈 2개
* 모터 1개
* IAR 프로그램

**4. 실험 과정**

**4-1. 프로젝트 세팅**

2주차 ppt를 참고하여 프로젝트 세팅을 동일하게 설정한다.

**4-2. main 소스코드 구현**

기존에 정의된 함수의 레퍼런스 경로는 아래와 같다.

..\Libraries\STM32F10x\_StdPeriph\_Driver\_v3.5

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

[그림 4.1] main.c의 RCC 설정 함수

사용할 포트들에 대해 clock을 enable 시켜준다. 이때, 사용되는 함수는 stm32f10x\_rcc.c파일을 참조한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

[그림 4.2] GPIO 초기화 함수

활성화된 GPIO 포트에서 사용할 각 핀에 대해 모드를 설정한다.

S/W와 관련된 핀인 C4, C13, B10, LED와 관련된 핀인 D2, D3, D4, D7, UART-TX핀인 A9, UART-RX핀인 A10을 설정한다. 각 설정 값들은 아래와 같다.

|  |  |
| --- | --- |
| 핀 | 설정 값 |
| C4, C13, B10 | input with pull–down/reset state |
| D2, D3, D4, D7 | General purpose output push-pull/Output mode, max speed 50MHz |
| A10 | input with pull–down/Output mode, max speed 50MHz |

[표 4.1] GPIO 설정

이때, 사용되는 함수는 stm32f10x\_gpio.c, stm32f10x\_gpio.h를 참조한다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

[그림 4.3] EXTI 초기화 함수

각 버튼에 EXTI(EXTernal Interrupt)핀을 할당한다. 각 버튼에 할당하는 EXTI line은 다음과 같다.

|  |  |
| --- | --- |
| 버튼 | 설정 값 |
| S1 | EXTI4 |
| S2 | EXTI10 |
| S3 | EXTI13 |

[표 4.2] EXTI 설정

이때, 사용되는 함수는 stm32f10x\_exti.c, stm32f10x\_exti.h를 참조한다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

[그림 4.4] USART 초기화 함수

Pc 와 Stm32 간 통신을 위해 USART1핀을 설정한다. 설정값은 지난 6주차 실험에 사용했던 값을 참조하여 아래와 같이 설정한다.

|  |  |
| --- | --- |
| 구조체 변수 | 설정 값 |
| WordLength | 8 bits |
| StopBits | 1 bit |
| Parity Bits | 없음 |
| Mode | RX/TX |
| HardwareFlowControl | None |

[표 4.3] USART 설정

이 때, 사용되는 함수는 stm32f10x\_usart.c, stm32f10x\_usart.h 를 참조했다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

[그림 4.5] NVIC 초기화 함수

EXTI를 NVIC과 매칭시킨다. 실험 요구사항에 따르면 버튼 S1, S2가 지연시간 없이 바로 동작해야 하기때문에, 각 버튼 별 우선순위를 설정한다. 이를 위해 Priority Group을 사용하기 위해 Preemption Priority를 1비트 제공해주는 NVIC\_PriorityGroup\_1을 사용한다. 각 포트들의 PreemptionPriority와

SubPriority 설정값은 아래와 같다.

|  |  |
| --- | --- |
| 구조체 변수 | 설정 값 |
| Preemption Priority | NVIC\_PriorityGroup\_1 |
| NVIC | Enable |

[표 4.4] 구조체 변수별 설정

|  |  |  |
| --- | --- | --- |
| 포트명 | PreemptionPriority | SubPriority |
| S1 | 0 | 0 |
| S2 | 0 | 1 |
| S3 | 1 | 0 |
| UART1 | 1 | 1 |

[표 4.5] 포트별 priority 설정

이 때, 사용되는 misc.c, misc.h를 참조했다.

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

각 Interrupt가 들어왔을 때 동작을 설정한다. USART1으로부터 putty를 통해 입력을 받아 ‘a’가 입력되었을 경우 전역변수인 LED\_mode가 1로, ‘b’가 입력되었을 경우 –1로 바뀌게 하였다. EXTI 4번 라인을 통해 GPIO\_C4가 입력되었을 때 LED\_mode값을 1로, EXTI 10번 라인을 통해 GPIO\_B10이 입력되었을 때 LED\_mode값이 –1로 바뀌게 설정해 주었다. EXTI 13번 라인을 통해 GPIO\_C13이 입력되면 미리 정의된 sendDataUART()를 통해 “TEAM10.\r\n”을 putty로 출력하였다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

마지막으로, LED의 동작을 LED\_Loop라는 함수로 정의하여 while문 안에서 LED\_mode의 값에 따라 1이면 오름차순 순서대로, -1이면 내림차순 순서대로 출력하게 한다.

**5. 결과**

전자제품, 전자 공학, 텍스트, 컴퓨터 하드웨어이(가) 표시된 사진

자동 생성된 설명

보드 작동 시 4개의 LED가 순차적으로 켜졌다 꺼지는 것을 확인하였다. Switch 1, Switch 2를 눌렀을 때 정방향/역방향으로 점멸 방향이 바뀌는 것을 확인하였다. 또한, 동작 중 switch를 눌렀을 때 지연시간이 발생하지 않고 점멸 방향이 즉시 바뀌는 것을 확인할 수 있었다.

텍스트, 실내, 사람, 개인용 컴퓨터이(가) 표시된 사진

자동 생성된 설명

Switch 3을 눌렀을 때, putty 화면에 TEAM10.이 개행 문자를 포함하여 정상적으로 출력되었고, putty 화면에 a, b를 입력했을 때 LED의 점멸 방향이 정방향/역방향으로 바뀜을 확인할 수 있었다.

**6. 결과에 대한 논의**

각 스위치를 눌렀을 때, putty에서 입력 값을 주었을 때 정상적으로 반응하는 것으로 보아 미리 정의된 함수들을 통해 각 포트들이 잘 설정 되었음을 알 수 있다.

또한 4주차의 실험과 달리 이번 실험에서 버튼을 눌렀을 때 지연 없이 바로바로 반응하는 모습을 볼 수 있었는데, 이는 실험 목표 문단에서 설명한 것과 같이 보드가 인터럽트 방식으로 잘 동작함을 알 수 있었다.

**7. 결론**

이번 주차에서는 기존과 같이 주소에 직접 값을 대입하는 방식이 아닌, 미리 정의된 함수를 이용해 각 레지스터를 활성화하고, 속성을 부여했다. 직접 주소를 대입하는 것보단 미리 정의되어 있는 함수를 사용하는 것이 더 쉬웠으며, 이후 주차는 미리 정의된 함수를 사용하는 방식으로 프로그램을 작성하는 것이 좋다고 느꼈다.

4주차의 실험 결과와 달리 다른 입력에 대한 동작이 수행되는 동안에도 새로운 입력이 발생하면 그에 반응하는 것 또한 관찰할 수 있었다. 이 결과를 통해 두 개 이상의 입력을 받는 시스템을 구성할 때 지연 없이 여러 동작을 수행할 수 있다는 면에서 인터럽트 방식이 이점을 갖는다는 것을 알 수 있었다.