**Embedded System Software 과제 3**

**(과제 수행 결과 보고서)**

**과목명: [CSE4116] 임베디드시스템소프트웨어**

**담당교수: 서강대학교 컴퓨터공학과 박 성 용**

**학번 및 이름: 20211584 장준영**

**개발기간: 2024. 5. 23. -2024. 5. 30.**

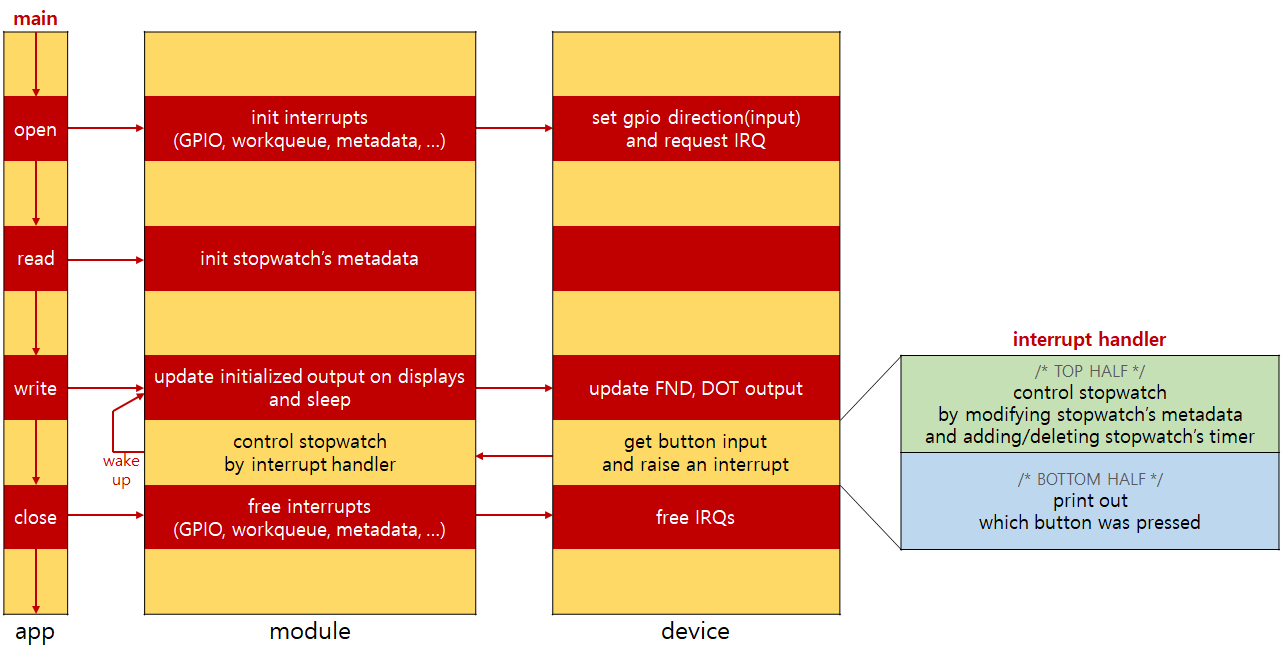
**최 종 보 고 서**

**I. 개발 목표**

이번 프로젝트에서는 FPGA 디바이스와 함께 운용되는 스탑워치 디바이스 드라이버를 구현하고, 해당 디바이스 드라이버를 테스트할 수 있는 유저 어플리케이션을 개발한다. 저번 프로젝트인 타이머 디바이스와 매우 유사하지만, 아주 중요한 차이점인 ‘GPIO를 통한 IRQ 핸들링’을 구현해야 한다. 기존에 구현해두었던 모듈 및 디바이스 코드에, 버튼 입력으로 인터럽트를 발생시키고 해당 인터럽트를 알맞게 처리하는 루틴을 추가해야 한다. 핵심적으로 버튼 GPIO-IRQ 매핑, 인터럽트 핸들링, top half/bottom half 등의 개념이 이용된다.

**II. 개발 범위 및 내용**

**가. 개발 범위**



ㅊㅌㅌㅊㅍㅇㄹㄴㄹㄴㅇ

**나. 개발 내용**

ㅊㅌㅍㅌㅍㅊ

**III. 추진 일정 및 개발 방법**

**가. 추진 일정**

|  |  |
| --- | --- |
| **2024. 5. 23.** | 저번 프로젝트의 코드에서 app.c와 device.c를 수정하였다. app.c를 수정하면서 어떤 file operation을 이용할지 계획하였다. device.c에서는 이번 프로젝트에서 사용하는 FND와 DOT matrix의 동작부를 수정하였는데, 스탑워치의 elapsed time만을 인자로 받고 내부에서 포매팅하도록 구현하였다. |
| **2024. 5. 24. ~ 2024. 5. 25.** | 디바이스 드라이버 모듈의 코드를 module.c에 작성하였다. 기존의 타이머 디바이스 드라이버의 코드에서 달라진 점은, GPIO를 통해 하드웨어 인터럽트를 관리해야 한다는 것이다. GPIO/IRQ, 인터럽트 핸들링, workqueue 등을 유념하여 구현하였다. |
| **2024. 5. 29. ~ 2024. 5. 30.** | **V.** 에서 후술할 문제로 인해 스탑워치의 동작 방식을 모두 바꾸었다. 실제 동작 중에는 문제가 전혀 없었지만, 주석을 추가하기 위해 코드를 리뷰하는 중에 잠재적인 동기화 문제의 가능성을 찾아 이를 개선하였다. 이후 완성된 코드를 컴파일하여 최종적으로 테스트하고 주석을 추가하였다. |

**나. 개발 방법**

ㅎㅇㄹㅇㅀ

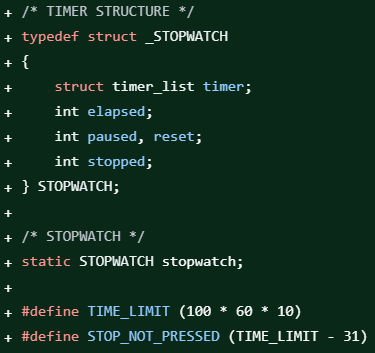
**IV. 연구 결과**

해당 프로젝트가 요구하는 사항을 모두 만족하도록 구현하였다.

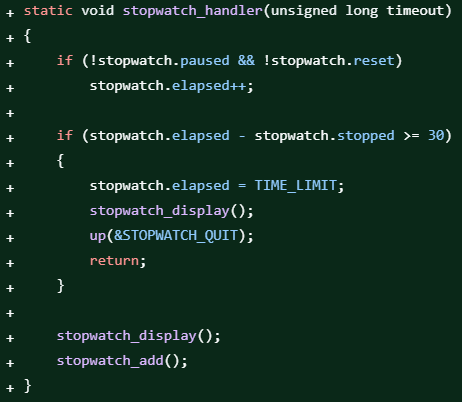
|  |
| --- |
| FPGA device, interrupt, stopwatch를 제어하는 하나의 모듈을 구현하였다. |
| interrupt handler를 top half와 bottom half로 나누어 구현하였다. |
| 보고서에 top half와 bottom half로 구분한 명확한 기준을 제시하였다. |
| 소수점 첫 번째 자리까지의 시간이 유지된다. |
| VOL- 버튼을 3초 이상 누른 채 유지하여 스탑워치가 종료될 때까지 유저 어플리케이션이 종료되지 않는다. |
| 프로그램 종료 시 출력 디바이스를 초기화한다. |
| 그 외 사소한 요구 사항을 만족하도록 구현하였다. |

**V. 기타**

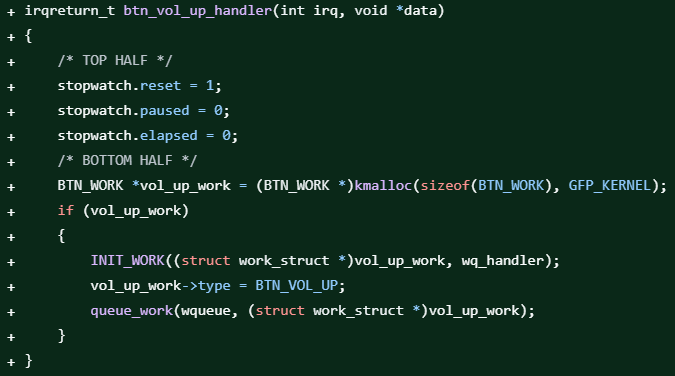
Github에 커밋한 프로젝트 버전 1을 확인하면, 문제 가능성이 있는 스탑워치 동작의 구현을 확인할 수 있다.



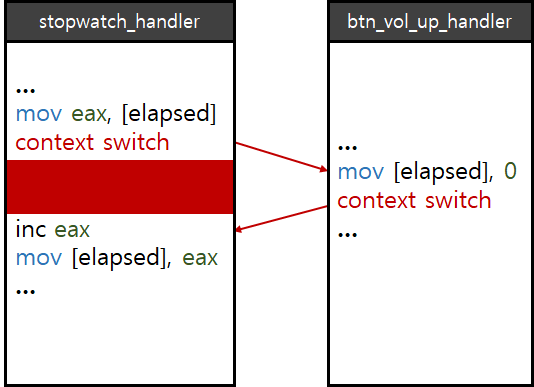
먼저 STOPWATCH 구조체에 paused, reset, stopped를 관리하고 타이머 핸들러에서 이를 각각 체크해 스탑워치가 어떤 상태에 있는지를 확인하고자 했다. back 버튼이 눌려 정지된 상태인지 저장하는 paused, vol+ 버튼이 눌려 초기화 되었거나 막 시작하였는지 저장하는 reset, vol- 버튼이 처음 눌렸을 때의 elapsed를 저장하는 stopped 필드를 통해 추가적인 타이머 선언 없이 모든 동작을 구현할 수 있을 거라고 생각했다.



스탑워치의 타이머 핸들러에서는 각각의 메타데이터 필드에 대한 처리를 한다. 멈춰있어야 하는 상황이라면 elapsed를 증가시키지 않고, stopped가 처음 눌린지 3초가 지났다면 중지한다.



인터럽트 핸들러의 일례로, vol+ 버튼이 눌렸을 때 각각의 필드를 초기값으로 설정한다. 이렇게 구현하면 이론적으로 동작의 문제는 없으나, 여러 context flow를 가지고 실행되는 커널 프로그램의 특성상 동기화 문제가 발생한다.



코드를 읽다가 추측한 문제 상황 중 한 가지로, stopwatch\_handler가 호출되어 elapsed가 1 증가하는 동안 인터럽트가 발생하여 btn\_vol\_up\_handler가 호출되는 상황이다. 인터럽트 핸들러에서는 스탑워치를 초기화하기 위해 elapsed를 0으로 설정하지만, eax 레지스터에는 reset이 불리기 전의 elapsed가 저장되어 있어 이 값을 1 증가시키고 다시 elapsed에 쓴다. 결국 reset이 제대로 동작하지 않고, 스탑워치는 정지해있지만 디스플레이에 표시되는 elapsed는 0이 아닌 경우가 발생할 수 있다. 실제 동작 중에 문제가 발생한 적은 없기 때문에 이 추측이 정확한 지는 모르겠지만, 만에 하나 있을 지도 모르는 문제 상황을 없애기 위해 개선하였다.