**REPORT**



|  |  |
| --- | --- |
| **과목명** | 임베디드 소프트웨어 |
| **학과** | 컴퓨터공학과 |
| **학번** | 12161658 |
| **이름** | 조재민 |
| **제출일자** | 20.12.11 |

**목차**

[1]: 프로젝트 설명

[2]: 코드 분석

[3]: 문제점 발견

[4]: 실행 결과

[5]: 프로젝트 소감

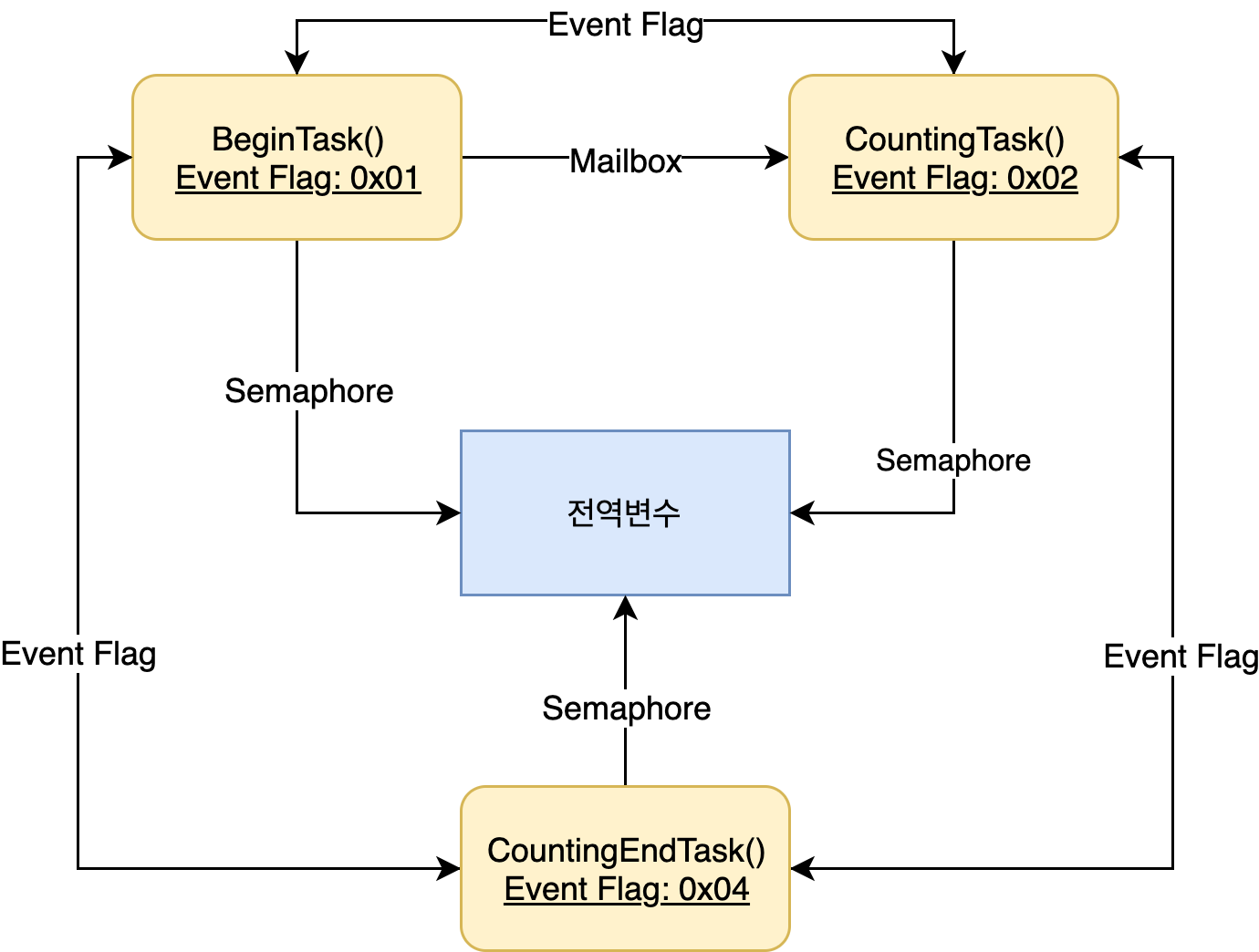
**[1]: 프로젝트 설명**

이번 2020년도 2학기 임베디드 소프트웨어 최종 프로젝트로, ATmega 128과 RTOS 환경에서 실생활에 사용할 수 있는 ‘인원수 카운트 센서기’ 를 만들었습니다. ATmega128 보드에 탑재된 조도센서 ‘CDS’를 활용해 사람이 센서기 앞을 지나가면 설정된 조도 값 이하로 내려가며 카운트 수를 증가시킵니다.

현 코로나 시국에 한 장소에 40명, 50명 이상의 인원이 차면 안된다는 아이디어에서 부터 시작하여, 추가적으로 이러한 조도 센서를 통한 수 카운트는 우리 실생활에서 유용하게 사용될 수 있습니다. 예를 들어, 주차장입구에 설치하여 차량 진입 수를 카운트하여 초과 시 이를 알려줄 수 있으며, 물류센터 컨베이어 벨트 등에 설치하여 물류 량을 측정할 수 있습니다.

초기에는 스위치 버튼을 이용하여 사용자가 최대 설정 값을 설정합니다. ‘코로나 시국’에 맞춰 저는 최대 설정 값을 40으로 하였습니다. 스위치 2번을 눌러 설정값을 저장한 뒤, 센서를 부착하여 조도 값을 이용해 사람/물체의 진입을 감지합니다. 순간적으로 센서에 들어오는 조도 값이 낮아지면 이는 사람/물체가 지나갔다는 것을 의미하며 카운트를 올립니다. 이때 사용자의 초기 설정 값에 비례해서 현재 어느정도 카운트가 된 건지를 확인하기 위해 FND에 현재 인원 수 뿐 아니라 LED를 이용해 퍼센트를 나타내 줍니다.

카운트로 증가 된 값이 초기 설정값을 넘어가면 FND에 ‘OVER’ 표시와 함께 부저 (알람)을 울립니다. 실 사용에서는 이렇게 부저를 울려 인원 수 및 물건의 수가 초과하였음을 알릴 수 있습니다.

**IPC Flowchart**

**[2]: 코드 분석**

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

사용 된 헤더 파일 및 각종 공유변수들 입니다.

**총 3가지 Task를 활용하였으며, Task 및 변수들간의 동기화를 위해 Mailbox, Semaphore, Flag Group 을 사용하였습니다.**

FND에 사용되는 FND\_DATA, FND\_sel, FND\_OVER 등의 변수가 있고,

LED에 사용되는 LED\_sel, LED\_DATA가 있습니다

버저에는 울릴 값 alarm, sound, OVER 등의 전역 변수가 활용되며,

maxCnt 변수는 사용자가 설정하는 최대값 입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

메인함수의 내용 입니다. 코드의 가독성과 각종 작동 함수들을 최대한 캡슐화 하여 나누었습니다. 각종 사용 될 Register를 Set 해주는 FND\_SET, LED\_SET, BUZZER\_SET, CDS\_SET 함수를 호출합니다. 그 후 Switch를 사용하기 위하여 DDRE, EIMSK, EICRB 값을 설정해 주었습니다. Sei() 함수를 통해 외부 인터럽트 허용을 해주었으며 그 후 사용될 Synchronization Mechanism인 Mailbox, Semaphore, FlagGroup을 생성 해 주었습니다.

프로그램에서 사용 될 세가지 Task를 생성한 뒤 실행시켜주었습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

메인함수에서 불러온 각종 레지스터 값들을 설정해주는 함수들 입니다. 각각은 모두 전역 변수로 설정되었으며 모든 Task에서 접근 가능한 변수이기에 각각을 OS\_ENTER\_CRITICAL, OS\_EXIT\_CRITICAL로 막아주었습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

구동되는 첫 Task입니다. Task의 실행 순서는 초기 Task Creation에서 설정한 Priority 뿐 아니라 OS Flag를 활용하여 Flag의 Setting에 따라서 Task 구동에 동기화를 주었습니다. 첫 Task는 사용자가 maxCnt 변수 (최대 허용 수)를 설정 하는 Task 로써, while 문 안의 flag 값이 변형될 때까지 Switch1 인터럽트에 따라 값을 증가시키고 Switch 2 인터럽트로 Set 하도록 하였습니다. 또한 설정 된 값을 다음 Task로 보내주기 위하여 Mailbox를 사용하였습니다. 텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

Switch 1, 2 인터럽트 함수입니다. Start 라는 전역변수가 현재 Task 1에서의 발생임을 인지시켜주며, Switch 1이 눌렸을 경우 maxCnt 값을 증가시켜주고, Switch 2로 start를 OFF로, flag 를 1로 바꿔주어 Task의 종료시점을 정해줍니다. 두 외부 인터럽트에서 사용되는 공유 변수들은 Semaphore를 사용하여 race condition을 방지해 주었습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

사용자가 Task 1에서 값을 증가시킬 때 해당 값을 FND에 표시해주기 위한 함수입니다. 최소 0에서 최대 40까지의 값을 설정할 수 있으며 이를 PORTC와 PORTG를 활용하여 FND에 출력해 주었습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

첫 번째 Task에서 사용자가 설정한 값을 넘겨받으며 실행되는 두번째 Task 입니다. 이 Task 에서는 조도센서를 활용하여 값을 넘겨받아 미리 설정된 특정 값 (조도값) 보다 작으면, 즉 어떠한 물체가 센서앞을 지나갔다면 현재 카운트 값을 증가시켜주는 Task 입니다.

Mailbox에서 Pend 상태로 Task 1에서 값을 넘겨받기를 기다린 뒤, FND와 LED, CDS를 활용하여 작업합니다.

이때의 FND에는 현재 카운트 되고있는 값을 출력해주며, 8개의 LED로 최대 허용 값 대비 현재 카운트 값을 나타냅니다.

현재 카운트 값이 초기 설정값보다 커지게 되면 이러한 Task의 종료를 위해 OVER라는 전역변수를 ON (1)로 설정해 준 뒤, 세 번재 Task의 Flag값인 0x04를 POST 해 줍니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Task 2에서 현재 카운트 값이 전체 허용값 대비 어느정도 인지를 나타내주는 Display LED 함수입니다. 8개의 LED를 활용하기 때문에 각 1칸의 값을 계산하여 그 값이 증가할 때 마다 PORTA를 활용하여 LED로 출력해 줍니다. 이러한 방식으로 사용자는 현재 설정한 수용인원의 방에 몇 퍼센트의 인원이 차있는지를 LED를 통하여 알 수 있습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

CDS 센서에서 값을 읽어오는 함수 입니다. ADCSRA의 6번 Bit에 OR연산으로 ADSC를 1값으로 바꿔주어 A/D 컨버터 변환을 시작합니다. ADIF 비트가 1로 SET 될 때까지 While 문을 통해서 대기시켜 줍니다. ADCL과 ADCH에 컨버팅 결과를 저장한 값을 value에 저장해 주고 이를 return 해 줍니다.

이 함수에서 받은 조도 값을 설정한 CDS\_VALUE (971)과 비교하여, 작다면 빛의 감지가 어둡다는 의미로 어떠한 물체가 센서 앞을 지나갔다고 해석합니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

세 번째 Task의 구성입니다. 세 번째 Task는 현재 초기 설정값 보다 많은 인원/물체가 감지되어서 알림을 주는 Task 입니다. Flag group의 0x04가 set 될 때까지 OSFlagPend 함수를 통해 대기하며, while문을 통하여 FND에 ‘OVER’ 문구 표시와 함께 버저에서 알림 음을 울립니다.텍스트이(가) 표시된 사진

자동 생성된 설명

TIMER2 Overflow 인터럽트 발생 함수입니다. 두 번째 Task에서 OVER라는 지역변수를 설정하였을 때 버저가 울릴 수 있도록 설계 하였습니다. 또한 최대한 경고음을 구현하기 위해 ‘높은 도 (UDO)’를 반복적으로 내도록 설계 하였습니다.

이러한 구현을 위하여 Task 3에서 200번 정도의 For 문을 돌리며 음정을 바꿔 주었습니다. 이 정도의 값을 설정 하였을때 높은 도의 반복이 최대한 자연스러웠 습니다.

**[3] 문제점 발견:**

이번 프로젝트를 진행하며 이론상으로 배운 각종 레지스터들과 주변기기 (Peripheral devices)를 원할 하게 작동시키기를 원했으나, 여러가지의 문제점이 발견되었고, 실질적으로 해결 못한 문제가 있었습니다.

첫 번째로, 두 번째 Task 에서 CDS 센서에서 조도값을 입력받아 카운트 수를 증가시키며 이를 FND에 출력할 때, ‘CDS의 입력 Interrupt Service Routine과 FND의 출력의 간섭으로 인해 또렷한 숫자 출력이 안되는 문제’가 존재합니다. CDS 센서에서 계속해서 (Task 2가 진행되는 동안) 값을 받아오고, 이와 동시에 FND에 출력 해 주어야 하기에, 이러한 동기화 과정에서의 문제점이 발생하는 것으로 추측하고 있습니다. 이러한 문제를 해결하기 위해 FND Port의 종료, LED port의 종료 등의 함수를 구현하여 동기화에 사용하기위해 노력해 보았으나, 실패하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명전자기기이(가) 표시된 사진

자동 생성된 설명

<<사용되지 못한 FND & LED Port Reset 함수와 문제점>>

두 번째로는 알람의 불안전함 이었습니다. FND에 OVER라는 값 출력을 위해 최초로 구성한 FND\_OVER 함수가 존재하였으나, TIMER Interrupt에 맞춰서 작동하는 알람에 의해 FND 출력이 일정하지 못한 버그가 발생하였습니다. 현재 이러한 문제는 두 출력을 같은 반복문 안에 넣음으로써 해결하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

<<사용되지 못한 Display\_OVER 함수>>

또한, FND의 출력 및 버저를 울릴 때 사용되는 각종 변수들을 **Semaphor를 이용하여 보호**하였을때 FND 출력과 버저의 출력이 일정하지 못하는 문제가 발생하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

PORTC, PORTG와 버저에 출력되는 sound를 각종 Semaphor로 보호하였을때 반복문과 충돌하는 (추측) 문제로써 해결되지 못하였습니다. 이러한 보호해야하는 변수들을 다른 Task가 아닌 Task 3에서만 접근 할 수 있도록 **ISR에서 OVER 라는 공유변수로** 1차적 보호를 해주었습니다.

전자기기, 회로이(가) 표시된 사진

자동 생성된 설명**[4]: 실행 결과**

스위치1과 2를 사용하여 사용자가 최대 허용 값을 설정합니다. 코로나 시국에 맞춰진 구성으로 최대 40명의 인원을 허용한다는 목적 하에 0 ~ 40까지의 값을 설정할 수 있습니다.

전자기기이(가) 표시된 사진

자동 생성된 설명

예시로 8의 값을 최대 값으로 설정 하였을 때, 조도 센서에 설정 보다 적은 양의 빛이 감지되면 사람이 통과했다는 의미를 갖고 카운트를 합니다. 또한 LED 를 통하여 ‘전체 허용 인원수 대비 현재 인원수’ 를 보여줍니다.

전자기기이(가) 표시된 사진

자동 생성된 설명인원 수가 8명을 초과 하였다면 FND에 ‘OVER’라는 경고문과 함께 버저를 크게 울립니다.

경고문을 통하여 현재 내부에 인원 수가 허용치를 초과하였음을 알 수 있습니다.

**[5]: 프로젝트 소감**

이번 2020년도 2학기 임베디드 프로그래밍 실습을 진행하며, ATmega 128과 여러 센서들 뿐 아니라 우리 주변의 많은 임베디드 RTOS 기반의 장치들에 대해 보다 깊게 알게 된 계기인 것 같습니다. ATmega 128로 만들어진 최종 결과물은 실제 우리가 실생활에 보이는 많은 장치들의 기능보단 한정적이며 단촐하지만, 이러한 기능들이 어떤 구성으로 만들어지며 각 Task들이 RTOS 환경에서 어떻게 동기화 되어 작동하는지를 알게 되었습니다.

예를 들어, 수많은 IoT 기기들을 개발자들이 어떤 식으로 코딩을 하였는지, 각종 센서들을 통하여 데이트를 뽑아내어 동작을 어떻게 관리하는지를 직접 느끼게 되었습니다. 그동안 소프트웨어에 대한 분석 (언어적 측면) 을 수도 없이 해 보았지만, 무심하게 작동법만 익힌 수 많은 디바이스 들의 보다 디테일한 작동 원리 및 구성에 대한 분석을 처음으로 해 보았습니다.

또한 프로그래밍 측면에서도 C, C++, Java, Python 등 많은 언어를 이용한 코딩을 해 보았지만 이번 프로젝트와 같이 RTOS 라는 환경에서 CPU와 Flash Memory의 연산 뿐 아닌 센서를 활용한 프로그래밍과, 공유 변수에 대한 Race Condition 발생 방지를 위한 각종 Synchronization Mechanisms 활용 등 제게는 새로운 경험이었습니다.

이번 프로젝트를 계기로 Embedded System에 대해 보다 더 깊은 지식과 경험을 쌓을 수 있었으며, 실제로 개발 과정에서 매우 큰 즐거움을 느낄 수 있었습니다.