**REPORT**

**| Embedded System**

**| 최종 프로젝트**

**| 밥 빨리 먹기 게임**

**| 컴퓨터정보공학과**

**| 12141540 박영창**

**| 12151355 김다솔**

목차

1. **개요**

* 개요
* 사용한 기능 및 간단한 설명

1. **프로젝트 내용 설명**

* 밥 빨리 먹기 게임

1. **프로젝트 실행**

* 실행 과정 스크린샷 및 진행 순서

1. **소스 코드**

* 변수 설명
* 함수 설명
* IPC
* 코드 진행

# **개요**

**개요**

2018년도 2학기 임베디드 시스템 최종 프로젝트는 자유 주제로 임베디드 보드 ( ATmega128 ) 프로그램을 설계하는 것이었습니다. 프로젝트 주제는 ‘밥 빨리 먹기’ 게임으로 프로젝트 내용 설명 부분에 자세히 적혀 있습니다.

**사용한 기능 및 간단한 설명**

이번 과제에서는 CPU에서의 개발과 임베디드 보드에서의 실행을 하는 교차 개발 환경을 사용했고, AVR Studio, Visual Studio를 사용해 MicroC/OS-II를 ATmega128에 Porting하여 개발하였습니다.

MicroC/OS-II의 주요 기능인 real-time scheduling 및 IPC ( Message Box, Message Queue, Semaphore, Flag )를 사용해 우선 task 실행 및 task 간의 통신 기능을 사용하였고, ATmega128의 센서 ( FND , LED , Buzzer , Switch ) 와 인터럽트 ( INT4, INT5, Timer 2 Overflow ) 기능을 사용하였습니다.

FND, LED, Buzzer, Switch, Interrupt에 대한 레지스터는 critical section에 진입해 초기값으로 지정해 두고, interrupt의 발생과 ISR의 실행 위주로 코드가 진행되며, ISR의 결과로 바뀐 변수 값을 확인하는 조건문에서 task 간의 통신 ( IPC )를 진행합니다.

Task는 총 4개를 구현하였으며, 모든 Task는 각자의 역할에 따른 FND/LED 출력을 가집니다.

Task의 수행은 IPC를 사용해 동기화 시켰으나, 보드의 문제로 특정 인터럽트가 한 번에 여러 번 발생하는 문제나 delay 시간으로 인한 출력 오류는 막을 수가 없었습니다.

소스 코드와 영상 결과물을 함께 첨부하였으니 참고해주시면 감사하겠습니다.

# **프로젝트 내용 설명**

**밥 빨리 먹기 게임**

**게임 진행**

1. FND에는 현재 먹은 밥의 수와 남은 시간이 표시됩니다.
2. 게임이 시작되면 LED에는 현재 위치와 밥의 위치가 표시됩니다.
3. 스위치를 눌러 현재 위치를 밥의 위치로 움직여야 합니다.
4. 제한 시간 내에 밥을 10개 먹으면 성공이며, 먹지 못하면 실패입니다.
5. 총 3 스테이지가 있으며 모두 클리어 시 배가 불러 끝이 납니다 ( 게임 클리어 ).

**세부 내용**

* 각 스테이지 별로 제한 시간이 다릅니다 ( 각 20초, 15초, 10초 )
* 게임을 시작하기 위해서 SW1을 누릅니다.
* LED에 현재 위치는 선명하게, 밥의 위치는 깜빡이면서 표시됩니다.
* 밥의 위치는 무작위로 결정되기 때문에 게임 시작 전에는 모든 LED가 켜져 있습니다.
* SW1을 누르면 오른쪽, SW2를 누르면 왼쪽으로 움직입니다.
* 밥을 먹을 때, 제한 시간이 줄어드는 것이 아주 잠시 멈추며 꿀꺽 소리가 납니다
* 제한 시간 내에 밥을 10개 먹는 데에 성공하면 GOOD을 출력하며 스테이지 클리어 음악이 나옵니다.
* 제한 시간 내에 밥을 10개 먹는 데에 실패하며 DEAD를 출력하며 게임 오버 음악이 나옵니다.
* 게임 오버 시, reset 버튼을 눌러 처음부터 다시 시작해야 합니다.
* 3 스테이지까지 모두 클리어 시 FND에 FULL을 출력하며 게임 클리어 음악이 나옵니다.
* 모두 클리어한 뒤 다시 플레이하려면 reset 버튼을 눌러 처음부터 플레이해야 합니다.

# **프로젝트 실행**

1. 

FND : 먹은 밥의 수 – 남은 시간

게임 시작 전 LED는 모두 켜져 있음

SW1을 눌러 게임 시작 시 제한 시간이 줄어들며 LED에 현재 위치와 밥의 위치가 표시됩니다.

SW1을 누르면 오른쪽으로 이동

SW2를 누르면 왼쪽으로 이동

밥을 먹을 때 마다 먹은 밥의 수가 증가하며 꿀꺽 소리가 납니다

1. 
2. 

제한 시간 내에 10개의 밥을 먹었을 경우 성공으로, FND에 GOOD을 출력하며 스테이지 클리어 음악이 나옵니다.

제한 시간 내에 10개의 밥을 먹지 못했을 경우 실패로, FND에 DEAD를 출력하며 ( D 표현 시 O처럼 보여 소문자로 표시했습니다. ) 게임 오버 음악이 나옵니다. 다시 플레이 하려면 reset 버튼을 눌러 처음부터 해야 합니다.

3 스테이지를 모두 클리어한 경우 FND에 FULL을 출력하며 게임 클리어 음악이 나옵니다. 다시 플레이 하려면 마찬가지로 reset 버튼을 눌러 처음부터 해야 합니다.

1. 
2. 

# **소스 코드**

**변수 설명**

**IPC 관련**

* OS\_EVENT \*Mbox : message box를 사용하기 위함
* OS\_EVENT \*Sem : semaphore를 사용하기 위함
* OS\_EVENT \*Mqueue : message queue를 사용하기 위함
* void \*MQueue\_Table[MSG\_QUEUE\_SIZE] : message queue를 사용하기 위함
* OS\_FLAG\_GRP \*f\_grp : flag를 사용하기 위함
* char res

**FND 관련**

* unsigned char FND\_DATA[] : FND로 출력할 숫자, 알파벳의 정보가 들어있다
* unsigned char FND\_sel[] : FND 출력 위치 값이 들어있다.
* int stage : 현재 stage를 저장하는 변수
* int timer[] : 각 stage 당 제한 시간을 저장하는 변수
* volatile int flag : 현재 게임이 진행 중인지를 나타내는 변수
* volatile int cnt : 먹은 밥의 개수

**LED 관련**

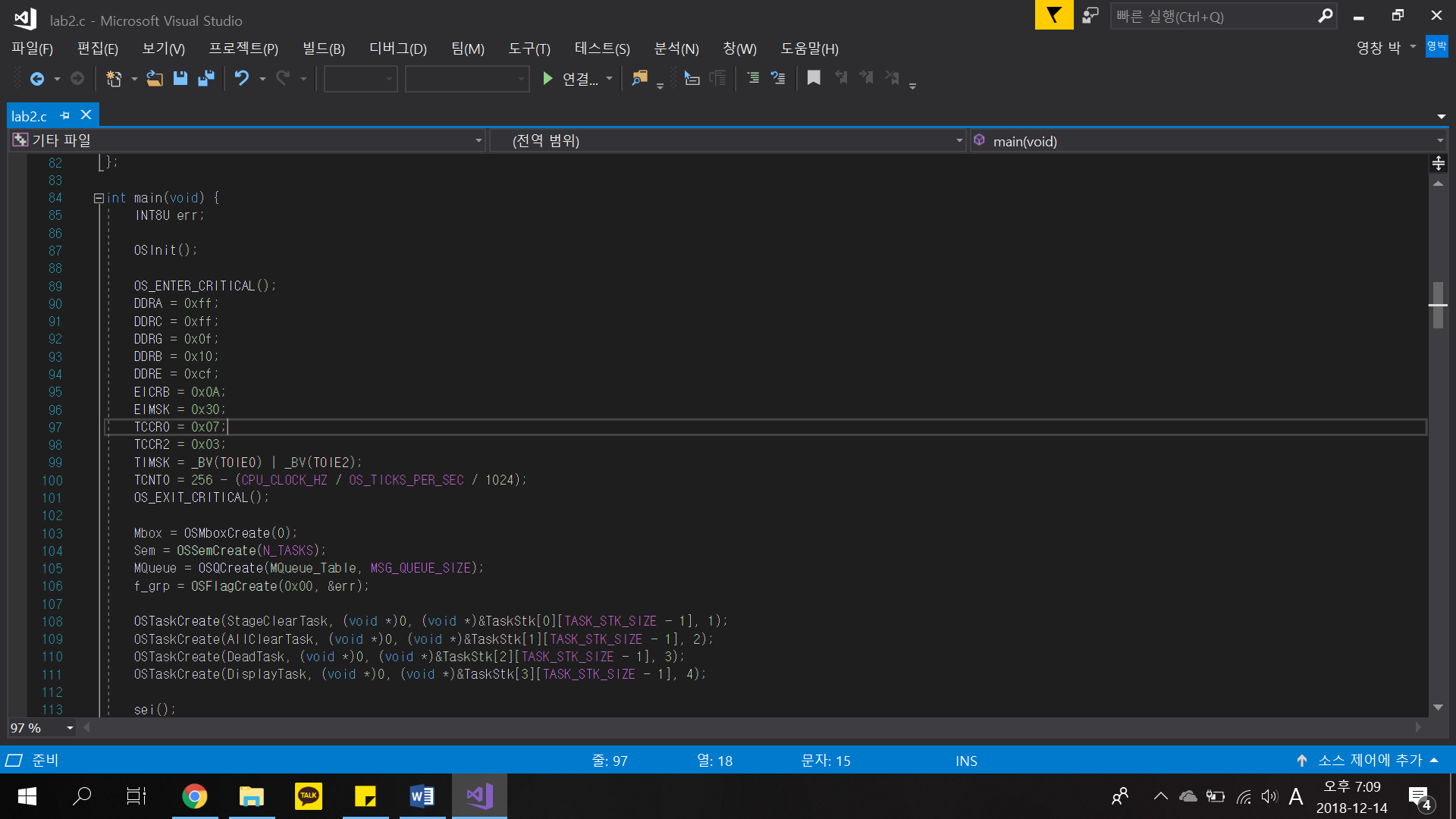
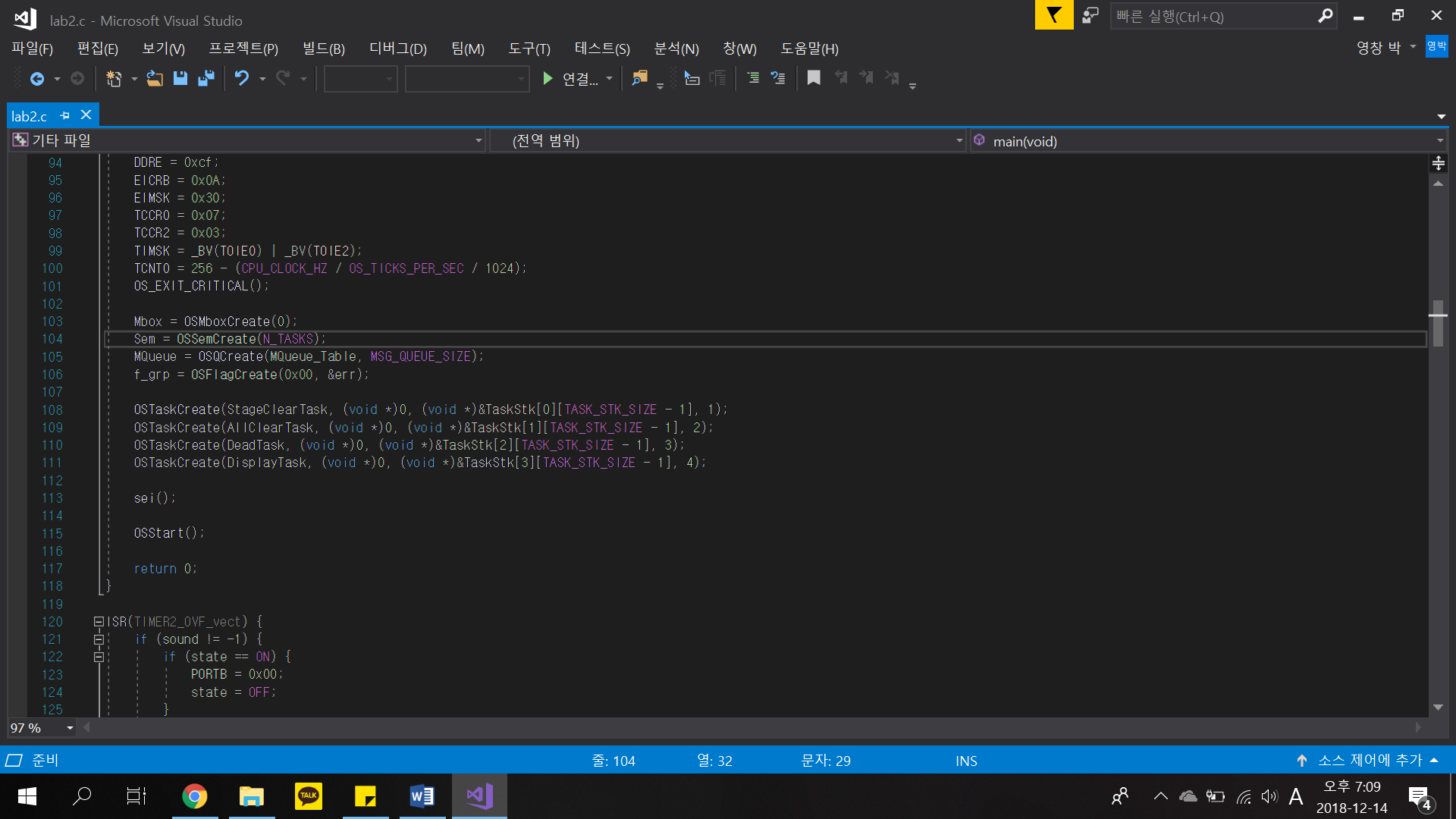
* unsigned char LED\_sel[] : LED 출력 위치 값이 들어있다
* volatile unsigned char curr : 현재 위치를 나타내는 인덱스
* volatile unsigned char meal : 밥의 위치를 나타내는 인덱스

**버저 관련**

* int eat[] : 밥을 먹었을 때 꿀꺽 소리의 음계가 들어있다
* int clear[] : 스테이지 클리어 음악의 음계가 들어있다
* int dead[] : 게임 오버 음악의 음계가 들어있다
* int full[] : 게임 클리어 음악의 음계가 들어있다
* volatile int sound : 버저에 출력할 음계의 주파수 지정을 위한 변수
* volatile int state : 버저에 출력할 음계의 파형 생성을 위한 변수
* int Time : 특정 상황에 시간을 저장해 음악을 출력하기 위한 변수

**함수 설명**

**메인, 초기화**



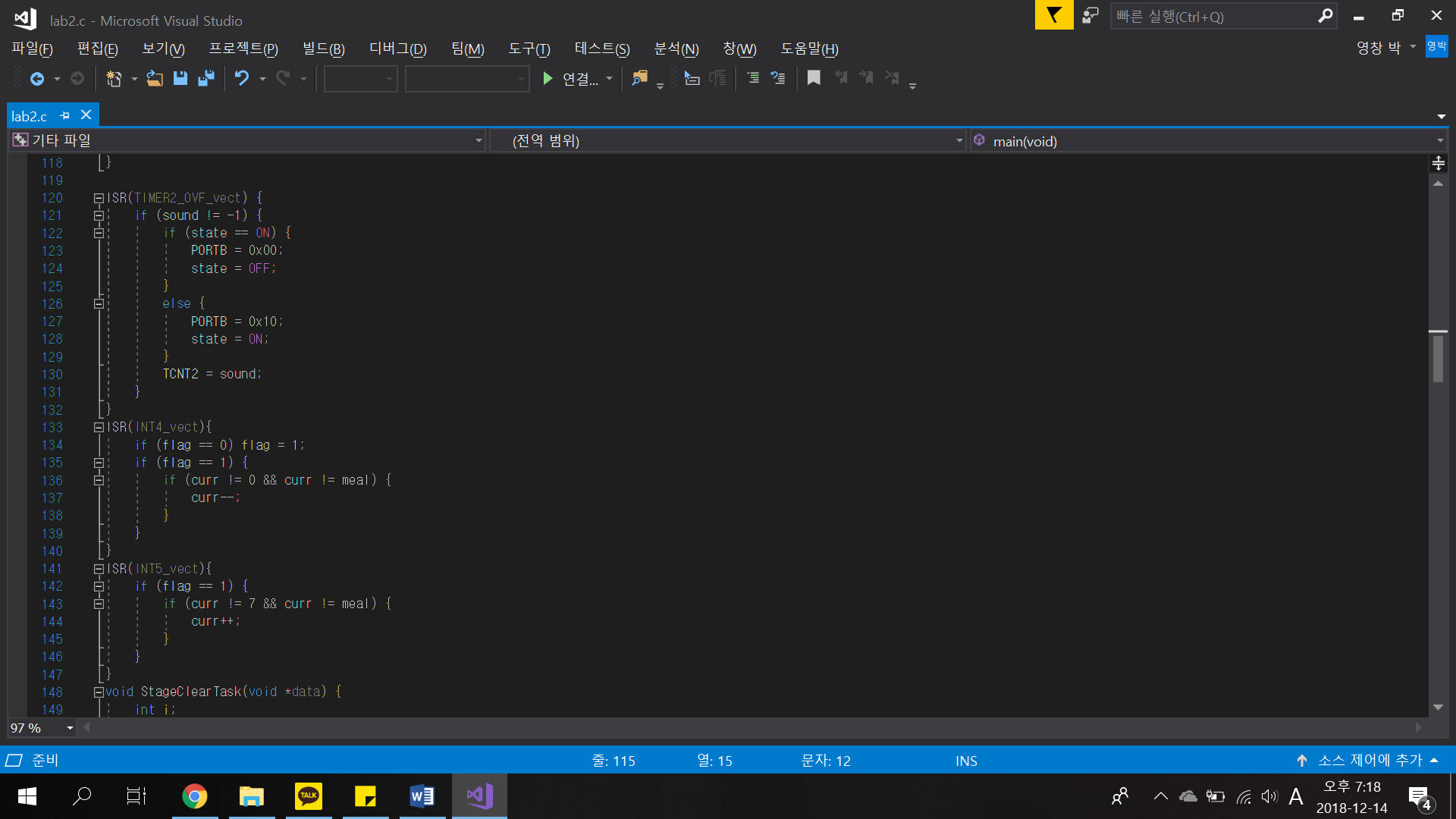
1. **Critical Section에서 각 레지스터 값 초기화**

* DDRA : LED 전부 출력
* DDRC : FND 전부 출력
* DDRG : FND 출력 방향 전부 출력
* DDRB : 버저 출력
* DDRE : SW1, SW2 사용
* EICRB, EIMSK : interrupt 초기 설정
* TCNT2 : 2번 타이머 사용

1. **각 ISR 선언 ( Message box, Semaphore, Message Queue, Flag )**
2. **Task Create : 지속적으로 Display하는 task를 가장 낮은 우선 순위로**
3. **인터럽트 활성화 ( sei() )**
4. **OS에게 제어권을 넘겨 줌**

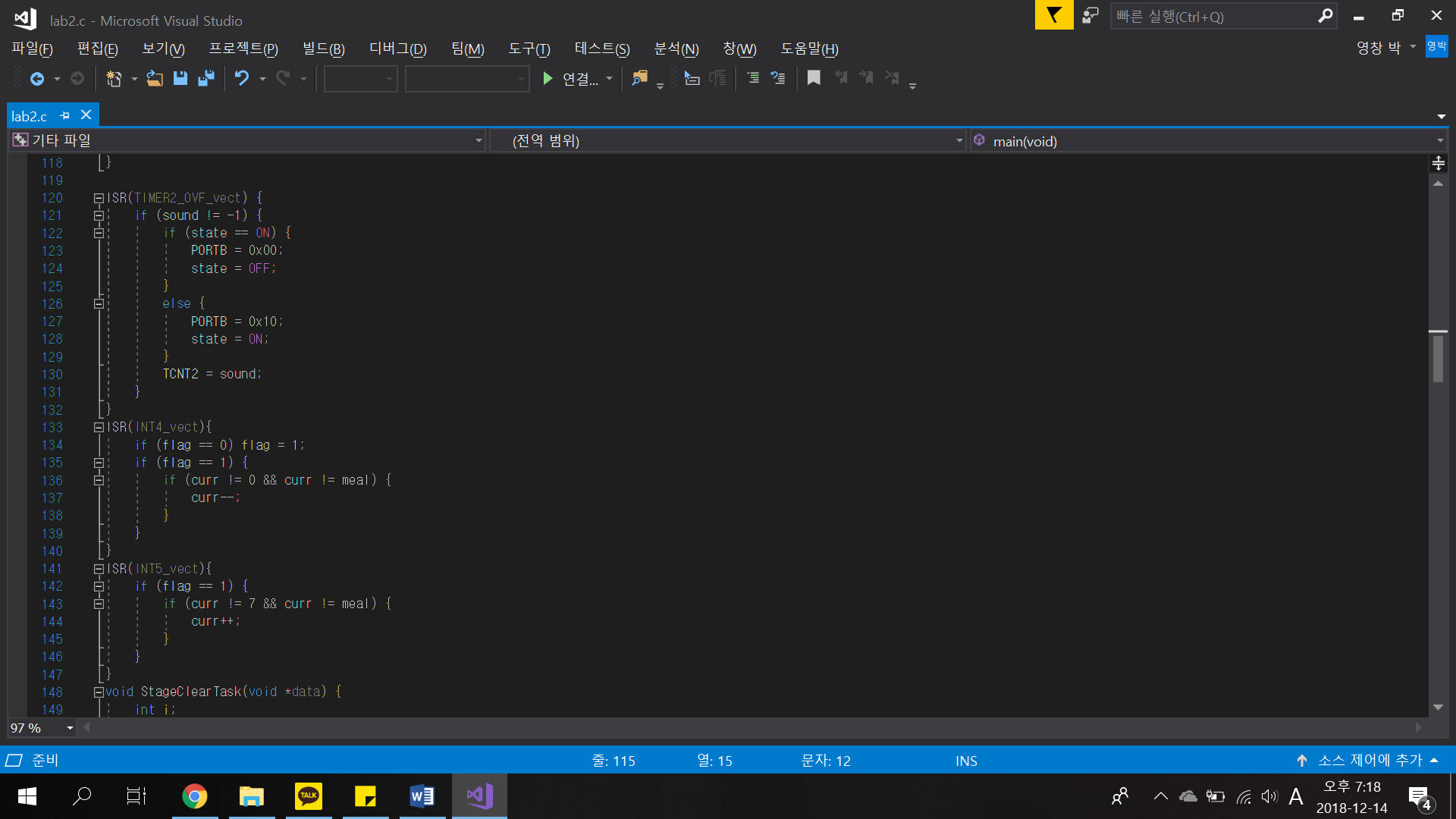
**ISR**

**2번 타이머 ISR**



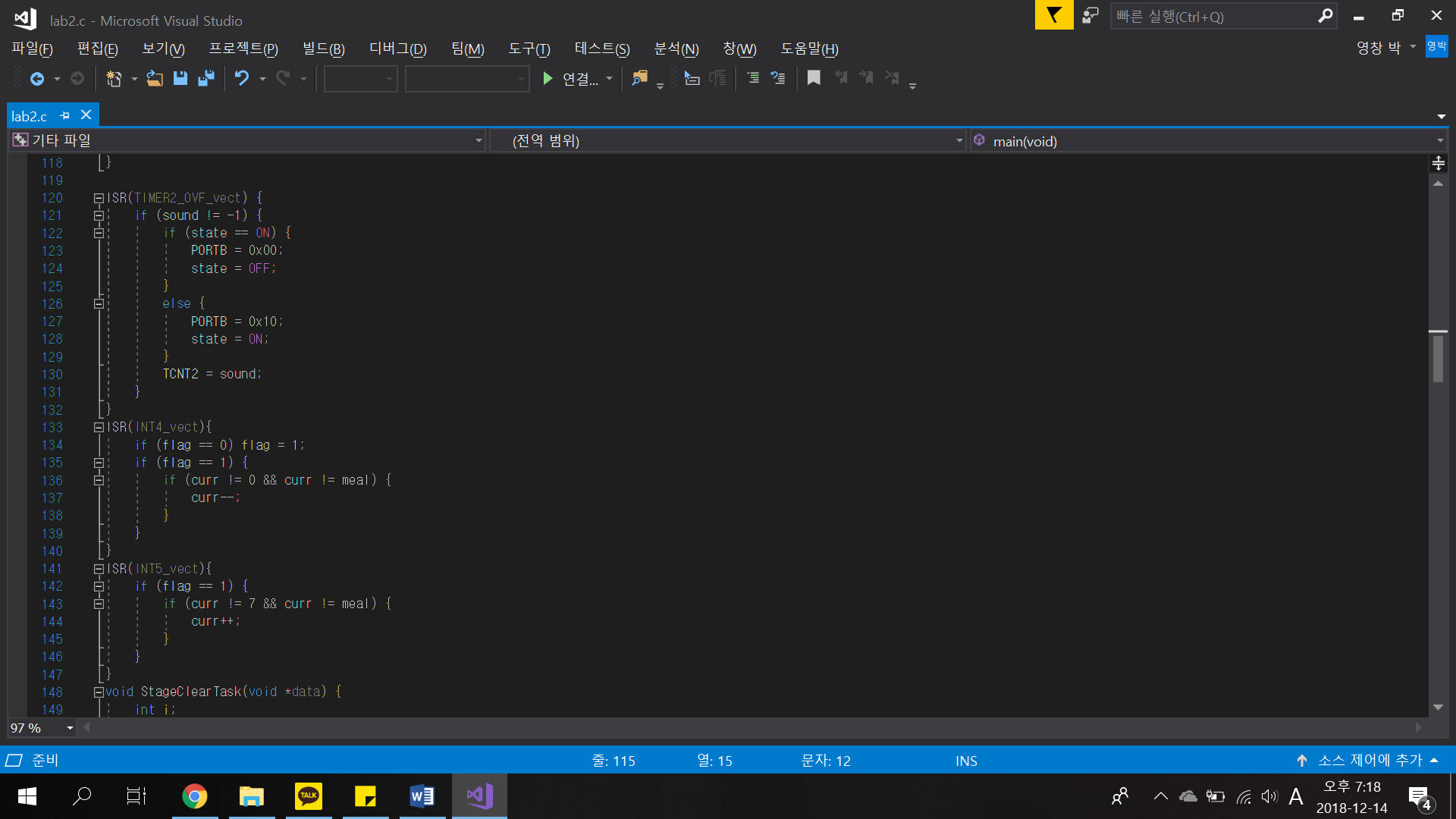
1. **2번 타이머 인터럽트 발생 시 현재 sound가 -1 이라면 아무 것도 하지 않는다 ( 무음 )**
2. **sound 값이 -1이 아니라면 TCNT2값을 sound로 지정해 음계의 파형을 형성한다**

**SW1 INT4 ISR**



1. **SW1이 눌렸을 때 flag가 0이라면 ( 게임이 진행 중이 아니라면 ) flag를 1로 설정한다. ( 게임을 실행한다. )**
2. **게임 실행 중에 SW1이 눌린다면, curr 값을 1 감소시킨다. ( 인덱스를 감소시켜 오른쪽으로 이동 )**
3. **현재 위치가 밥 위치와 같다면 curr 값은 변하지 않는다.**

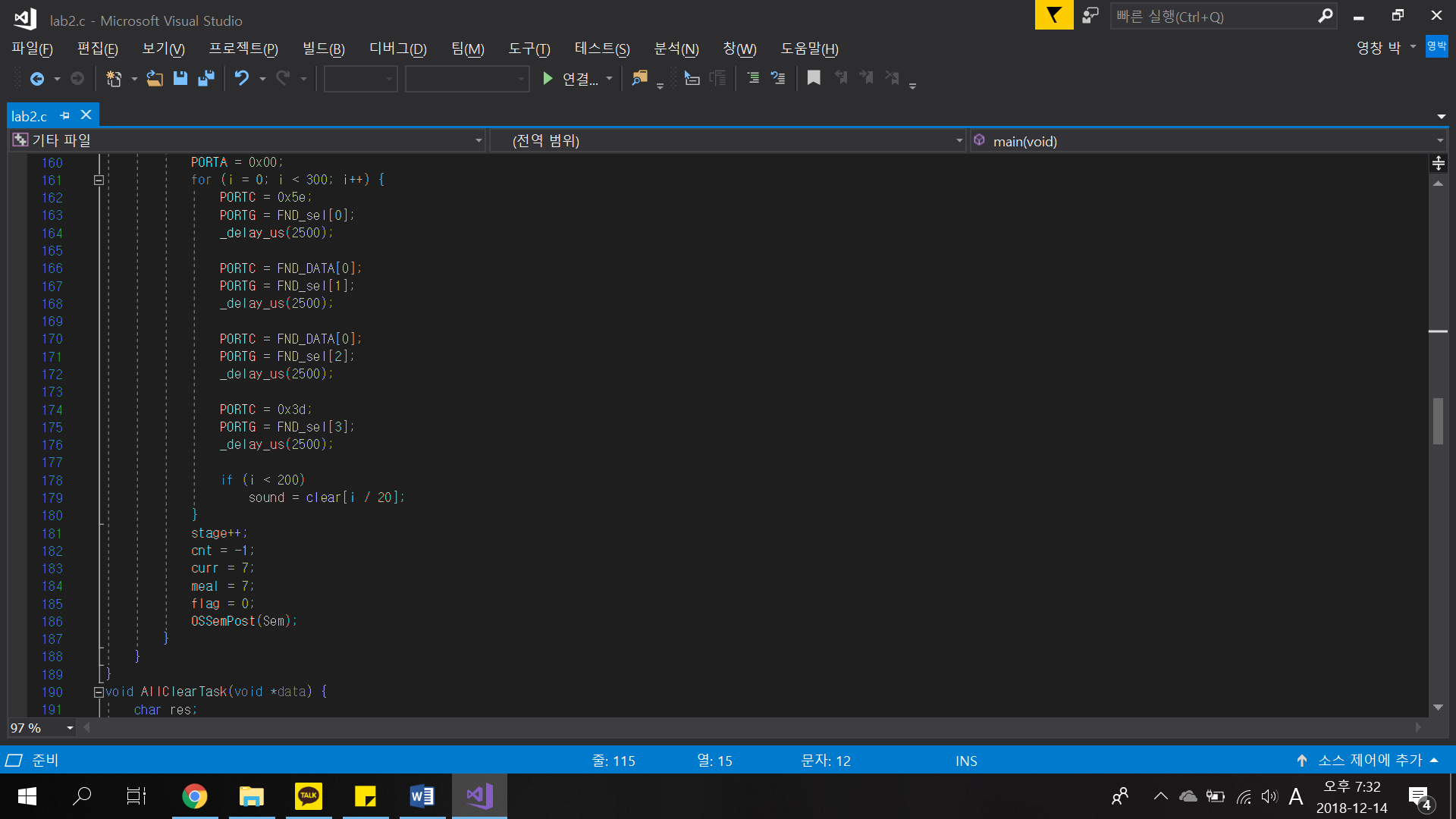
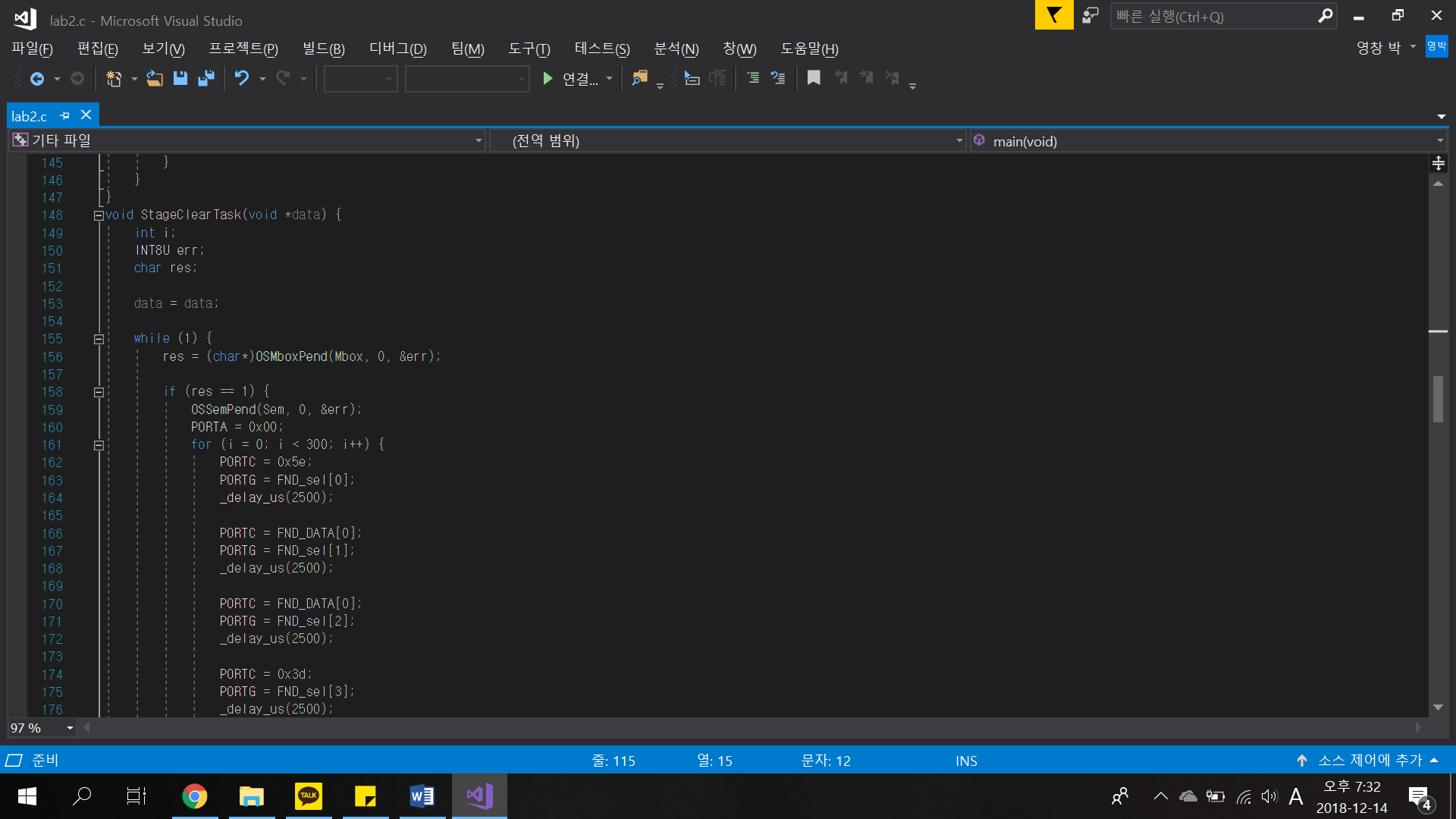
**SW2 INT5 ISR**



1. **게임이 실행 중일 때 SW2가 눌린다면, curr 값을 1 증가시킨다. ( 인덱스를 증가시켜 왼쪽으로 이동 )**
2. **현재 위치가 밥 위치와 같다면 curr값은 변하지 않는다.**

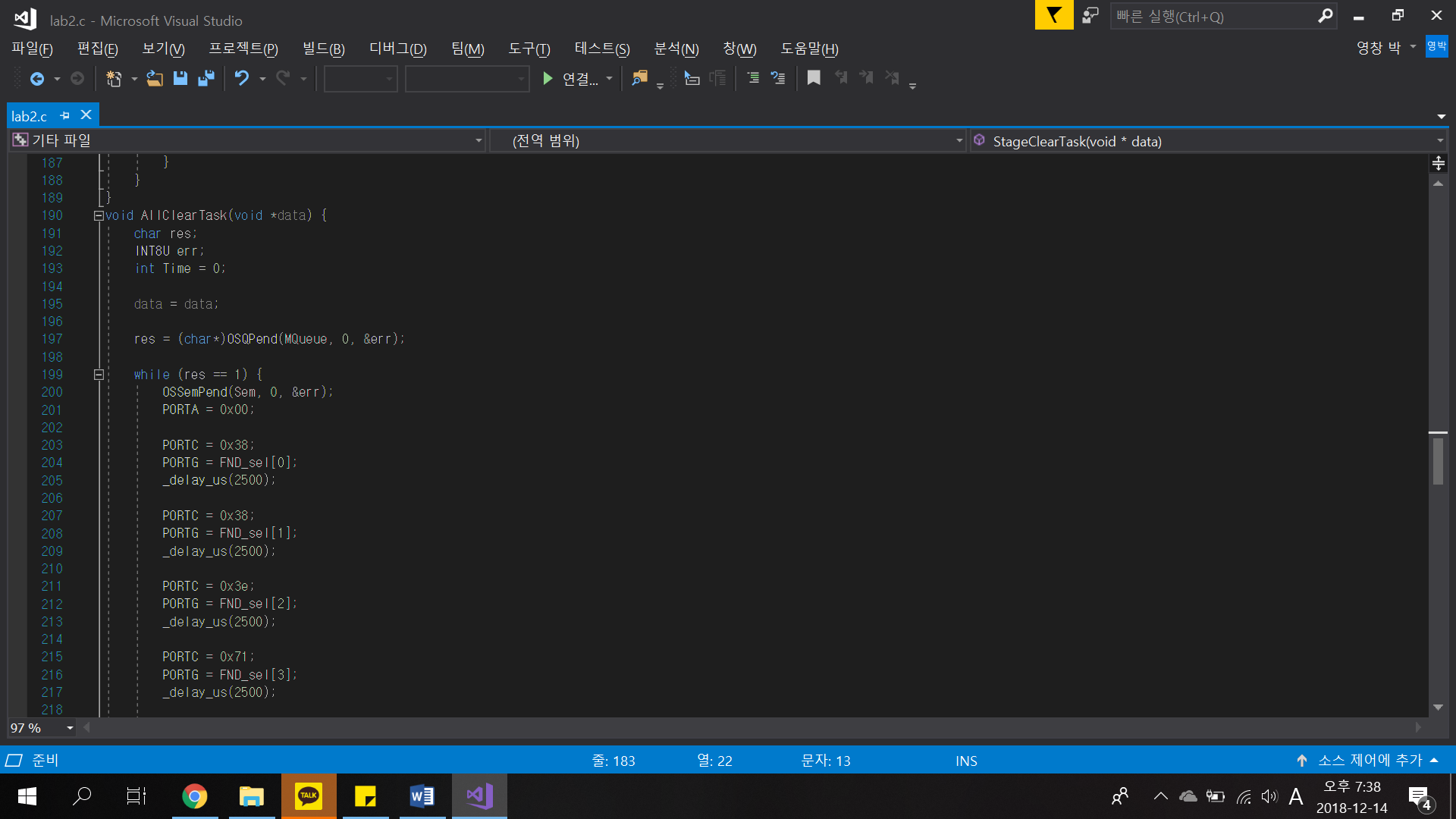
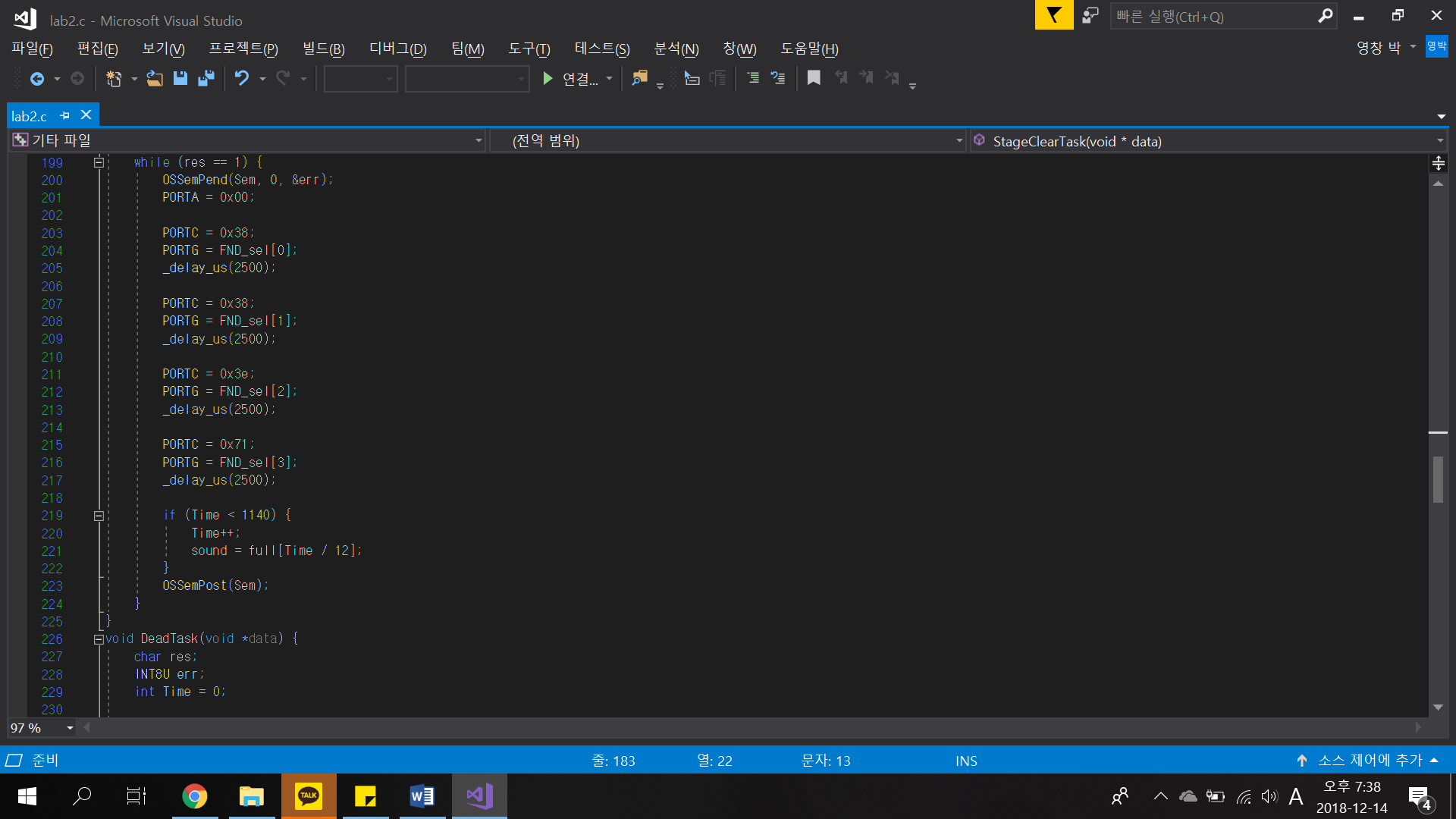
**Task**

**StageClearTask**



1. **Message box에 신호가 오기를 pend한다.**
2. **전역변수에 접근하기 위해 semaphore를 pend하고 FND에 GOOD을 출력하면서 clear[]에 저장된 음계를 버저로 출력한다.**
3. **다음 스테이지로 이동하기 위해 stage를 1 증가시키고, 관련 변수들을 초기값으로 지정한 뒤 semaphore를 post한다.**

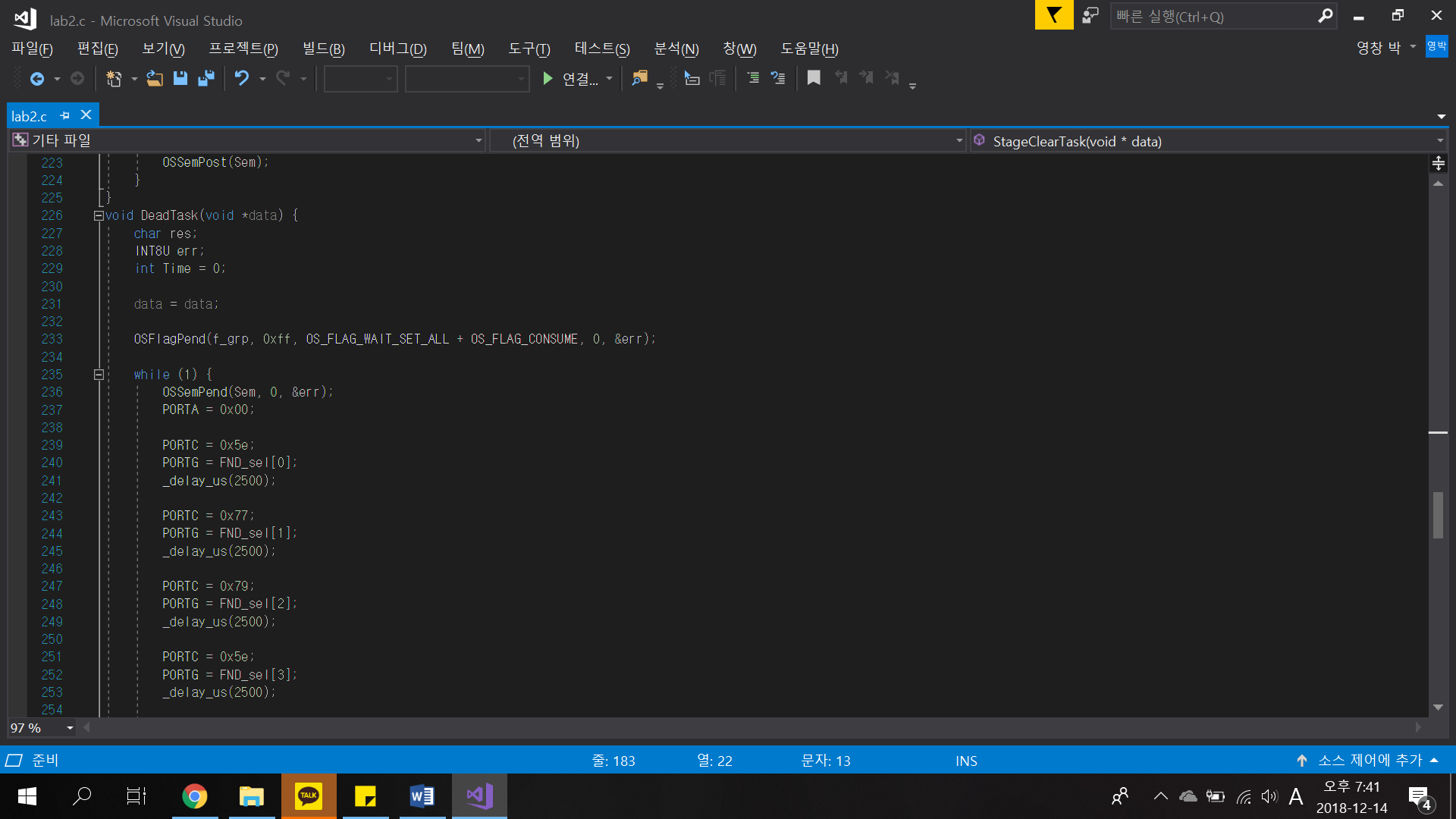
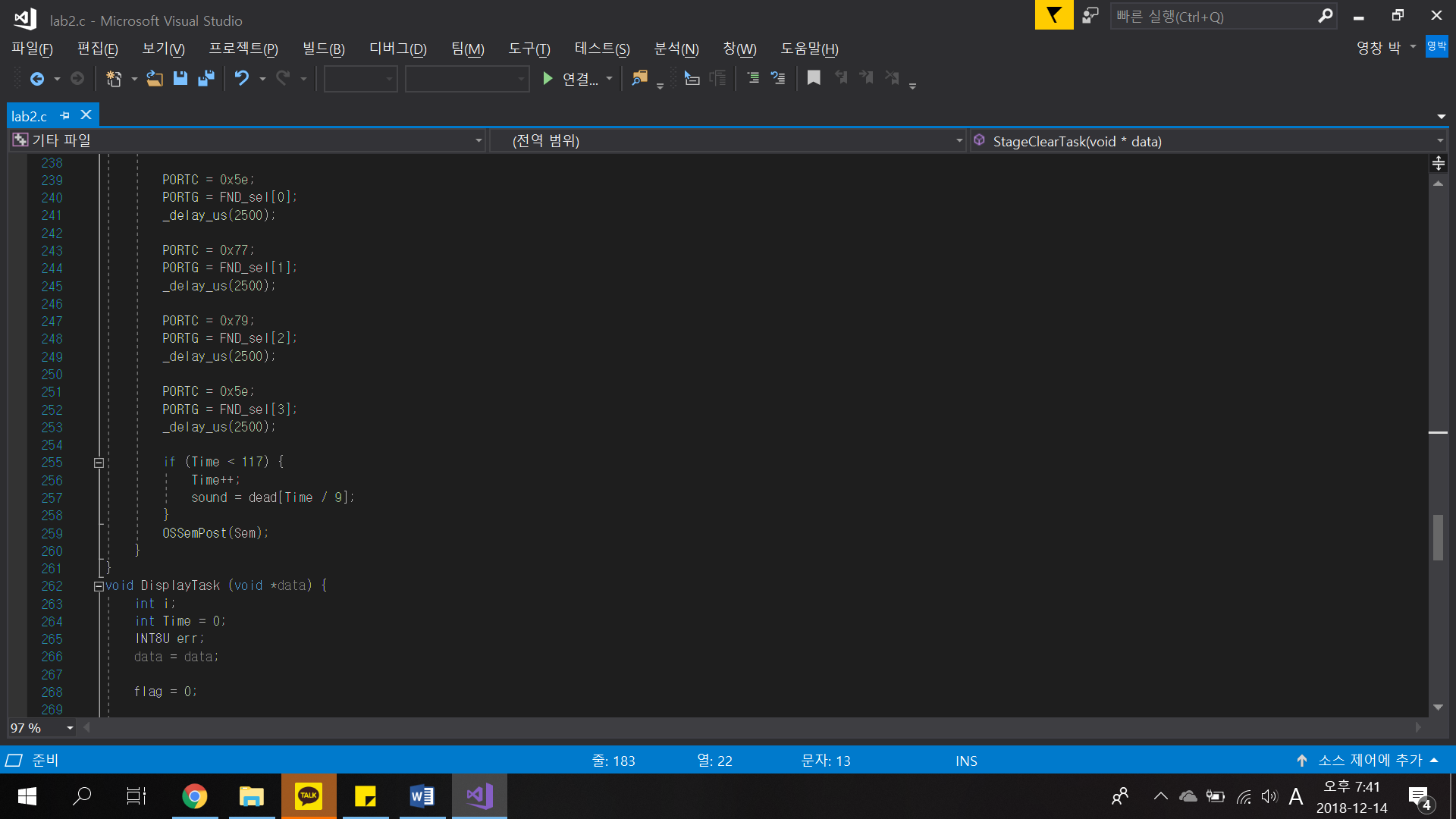
**AllClearTask**



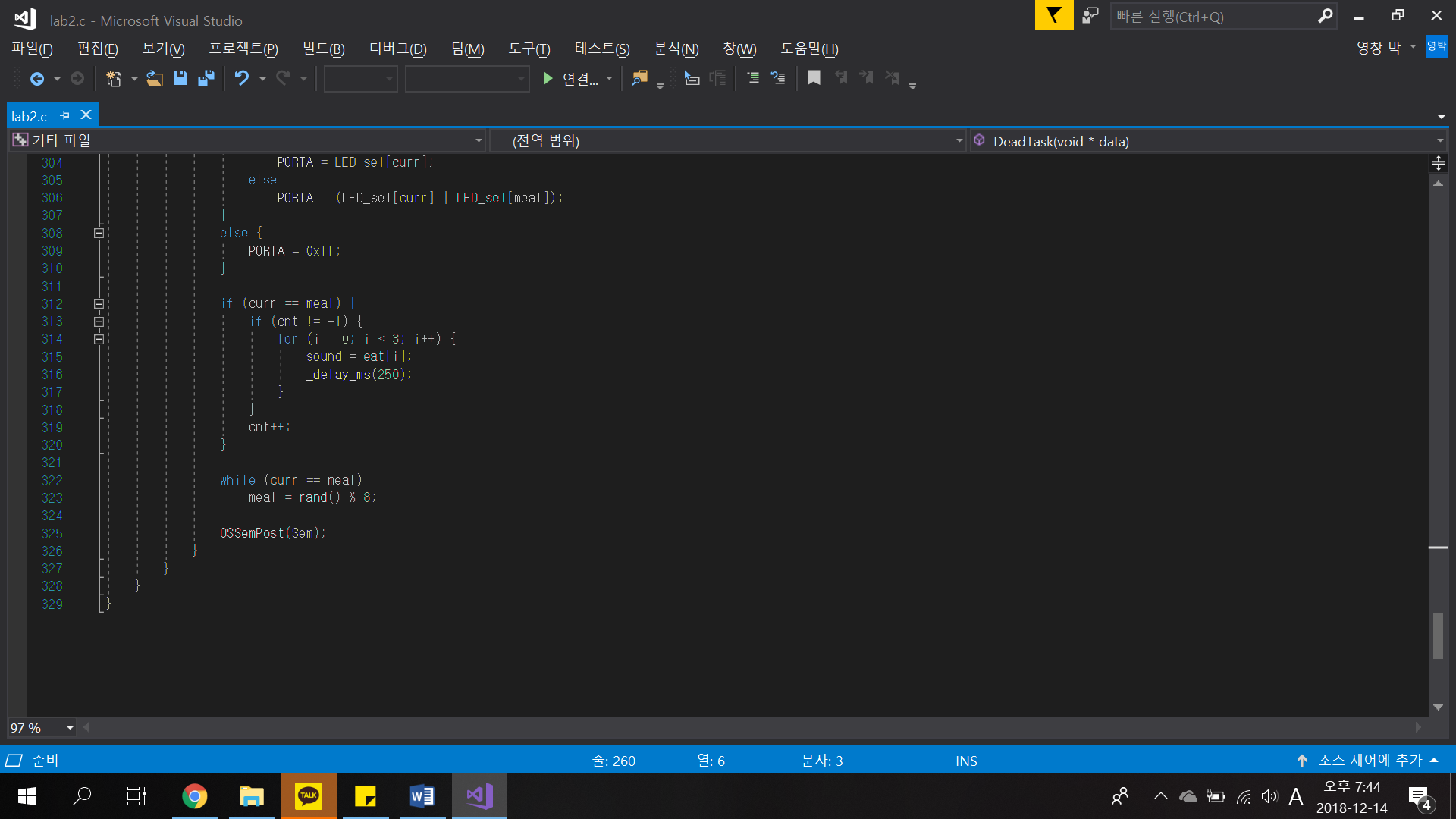
l

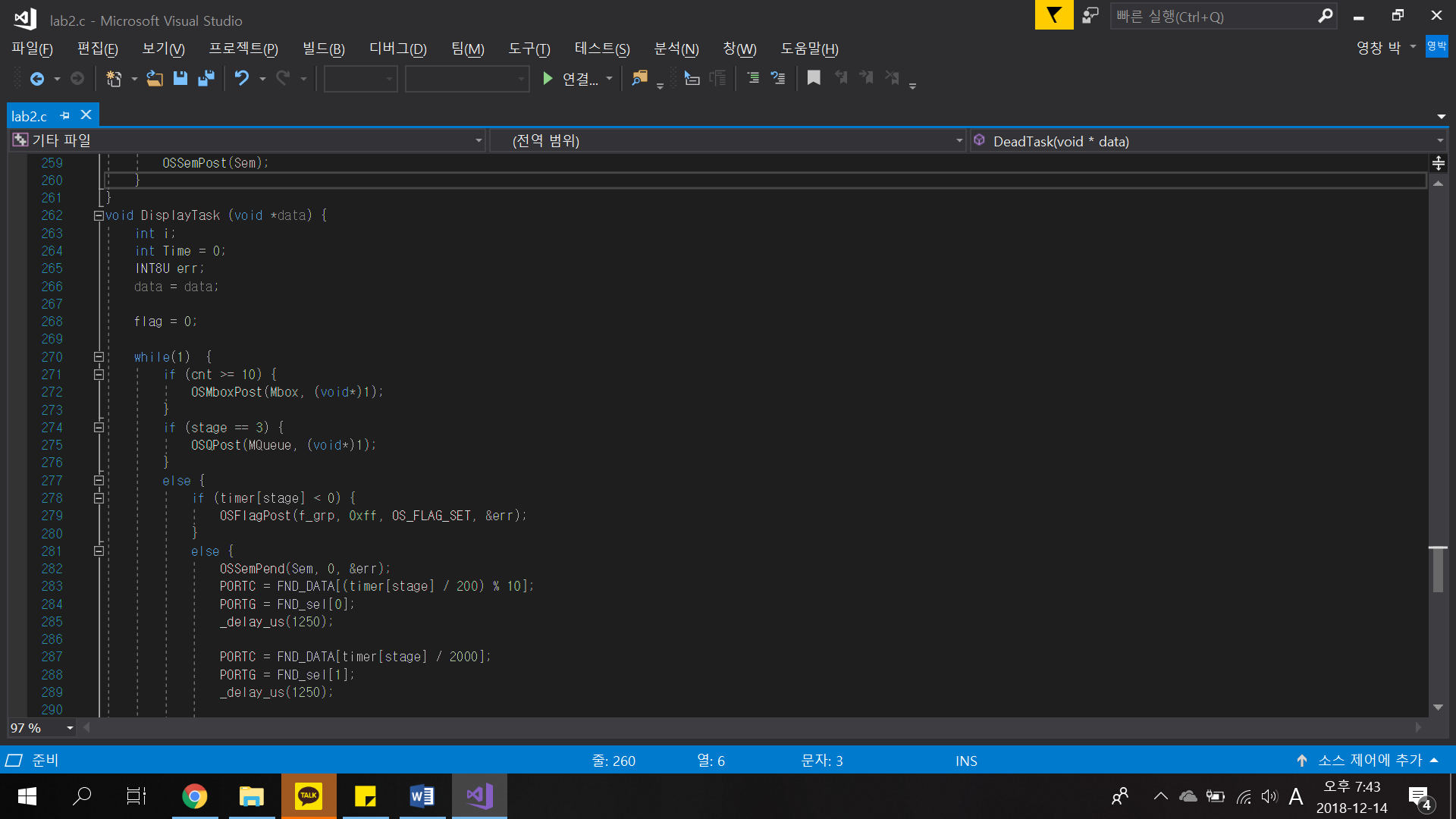
1. **Message queue에 신호가 오기를 pend한다**
2. **전역변수에 접근하기 위해 semaphore를 pend하고 FND에 FULL을 출력하면서 full[]에 저장된 음계를 버저로 출력한다.**
3. **Semaphore를 post한다.**

**DeadTask**



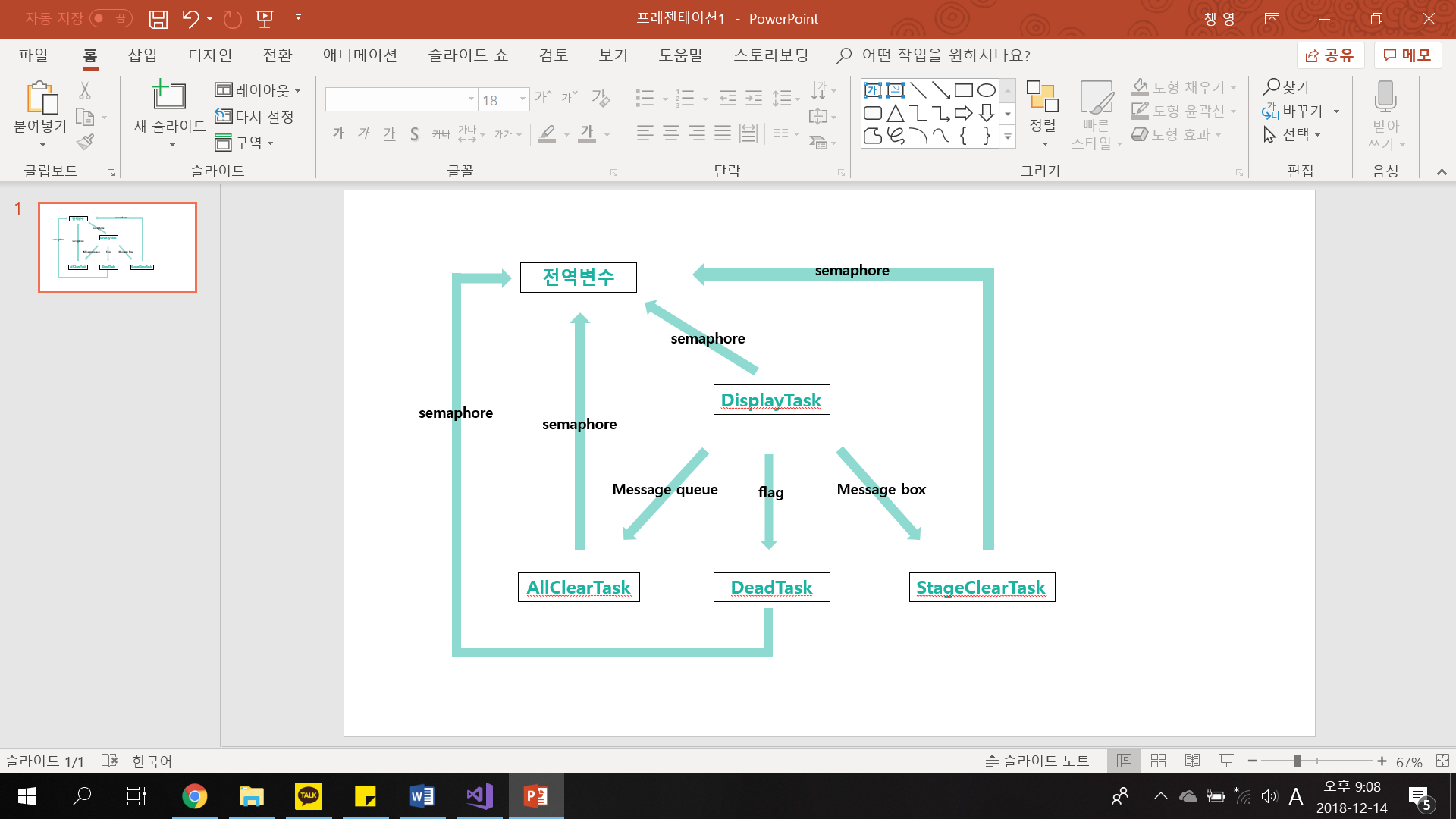
1. **Flag에 신호가 오기를 pend한다.**
2. **전역변수에 접근하기 위해 semaphore를 pend하고 FND에 DEAD를 출력하면서 dead[]에 저장된 음계를 버저로 출력한다.**
3. **Semaphore를 post한다.**

**DisplayTask**



1. **Cnt가 10 이상이면 ( 밥을 10개 먹었으면 ) message box에 신호를 보낸다.**
2. **Stage가 3이면 ( 모든 스테이지를 클리어 했으면 ) message queue에 신호를 보낸다.**
3. **Stage가 3보다 작고 timer의 시간이 0보다 작아지면 flag에 신호를 보낸다.**
4. **위 조건을 모두 만족시키지 못하면 전역변수에 접근하기 위해 semaphore를 pend하고 FND에 먹은 밥의 수와 남은 시간을 출력한다.**
5. **만약 flag가 1이면 ( 게임이 진행 중이면 ) timer는 1씩 감소한다.**
6. **timer값을 확인해 현재 위치는 선명하게, 밥의 위치는 깜빡거리게 나타낸다. 게임이 진행 중이 아니라면 모든 LED를 켠다**
7. **현재 위치와 밥의 위치가 같다면 ( 밥을 먹었다면 ) eat에 저장된 음계를 버저로 출력하고 cnt값을 증가시킨다. 그 다음 밥의 위치를 현재 위치와 다른 위치로 무작위로 생성하고, semaphore를 post한다.**

**IPC**



**코드 진행**

1. **main함수에서 각 레지스터 값 초기화, IPC 초기화, task 생성을 한다. 이 때, DisplayTask를 가장 낮은 우선순위로 생성한다.**
2. **OS에게 제어권을 넘긴다**
3. **StageClearTask, AllClearTask, DeadTask 의 우선순위 순으로 실행되지만 세 task 모두 IPC를 pend하므로 가장 우선순위가 낮은 DisplayTask가 실행된다.**
4. **Interrupt의 발생을 받아들이며 ISR을 실행하며 volatile 변수가 변경됨에 따라, 이를 확인하는 조건문 내에서 각각 message box, message queue, flag로 신호를 보낸다.**
5. **IPC의 신호를 받은 task가 실행된다. StageClearTask의 경우, while 문 내의 코드를 한번 실행하고 다시 pend하지만 AllClearTask, DeadTask의 경우 게임을 계속 진행하려면 reset을 눌러야 하므로 한 번만 pend하고 더 이상 pend하지 않는다.**