마이크로프로세서설계실험

Term Project 2 조 결과 보고서

경북대학교 전자공학부
최용진, 신수진

목 차

1. Term Project 주제

마이크로프로세서설계실험 과목의 2 조 Term Project 주제는 'S32K144 Evaluation Board 를 이용한 차박 임베디드 시스템'이다.

2. 조원 소개 및 기여 내용

조원들은 경북대학교 전자공학부에 재학 중이며, 임베디드 시스템을 집중적으로 공부하고 있다. 본 Term Project 를 통해 MCU 와 임베디드 시스템 설계 및 활용 능력을 배양하고자 한다.

공통적으로 HW 디버깅과 이벤트 처리를 설계하였다.

최용진 – SW 디버깅 및 코드 최적화, 다음 모듈들 설계

야생동물 감지, 블라인드 모듈, lcd 모듈, 실내등 모듈

신수진 – 시스템 설계 (동작에 필요한 센서 및 액추에이터), 다음 모듈들 설계

AC/Heater 모듈, 블라인드 모듈, lcd 모듈, SOS 모듈

3. 구현 내용 간략 소개

차박 시스템에서는 크게 6 가지 동작을 수행한다.

3-1. 야생동물 감지

버튼을 눌러 야생동물 감지 시작하면 초음파 센서를 통해 거리 정보를 얻는다. 거리값을 바탕으로 야생동물(물체)을 감지해 사용자가 이를 알 수 있게 한다.

3-2. LCD 출력

각 기능의 동작을 LCD Interface 를 통해 출력한다.

3-3. 실내등 제어

스위치를 통해 LED 를 출력한다.

3-4. 블라인드 제어

블라인드를 내리고 올림을 DC 모터 출력 제어를 통해 표현하였다. 버튼을 누르는 동안 블라인드(DC 모터)가 동작한다.

3-5. AC/Heater 제어

VR 을 통해 ADC 값을 조절한다. 이 때 ADC 값이 특정값 이상이 되면 AC/Heater 가 켜지고 ADC 값은 설정온도가 된다. 현재 온도와 설정 온도에 따라 AC 와 Heater 가 결정되고 현재 온도는 설정 온도를 따라간다.

3-6. SOS 호출

스위치를 일정 시간 누르면 SOS 가 호출된다. FND 에 SOS 표시를 띄우고 이는 비상 상황에 대한 구조 요청을 보냈음을 의미한다.

4. 구현 과정

4-1. 주제 구상 및 사용 장치 결정

4-2. 시스템 구조 추상화 및 소프트웨어 흐름도 작성

4-3. 개별 모듈 설계 및 전체 기능 통합

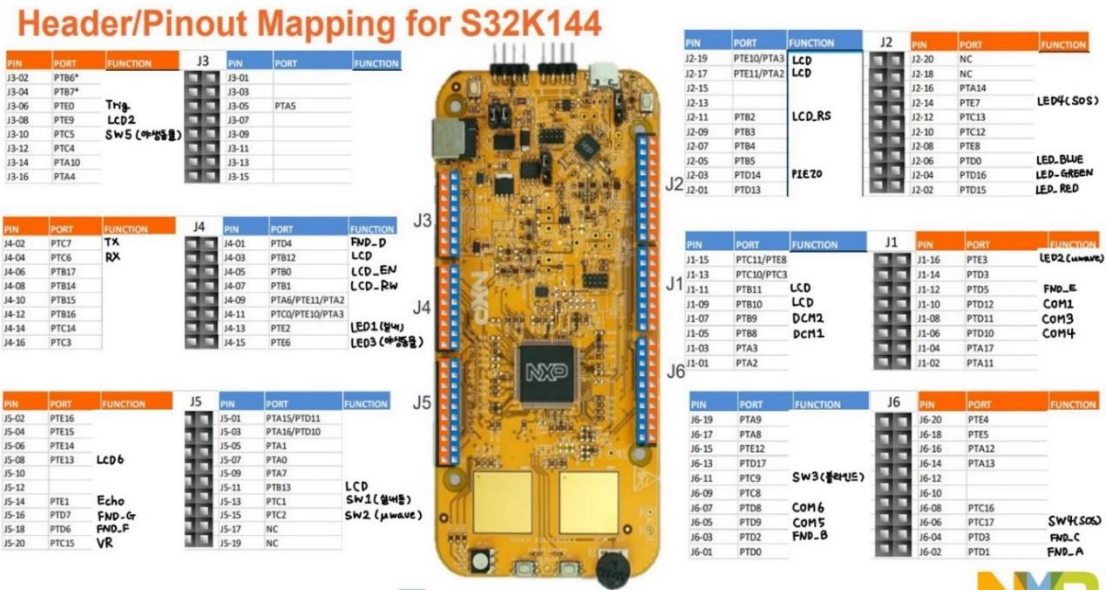
4-4. HW 및 SW 디버깅

4-5. 개발 결과 종합 및 결과 보고서 작성

5. 입출력 할당 테이블과 사용한 주변 장치

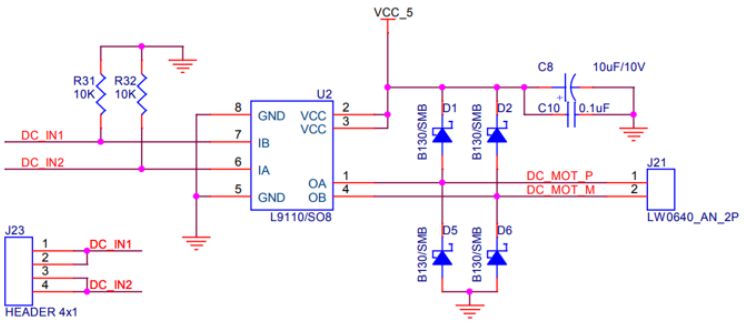
Pin	Name	Type
PTC1	indoor_sw	DIN
PTC2	uwave_sw	DIN
PTC9	blind_sw	DIN
PTC17	SOS_sw	DIN
PTC5	wildlife_sw	DIN
PTE2	indoor_led	DOUT
PTE3	uwave_led	DOUT
PTE6	wildlife_led	DOUT
PTE7	SOS_led	DOUT
PTD0	BLUE LED	DOUT
PTD15	RED LED	DOUT
PTD16	GREEN LED	DOUT
PTD1~7	FND_ALPHA	DOUT
PTD8~12	FND_COM	DOUT
PTC15	VR	ADC
PTD14	PIEZO	PWM
PTE0	TRIG	DOUT
PTE1	ECHO	DIN
PTB8	DCM1	DOUT
PTB9	DCM2	DOUT
PTB0~2,10~13	LCD	DOUT
PTC7	UART_TX	DIN
PTC6	UART_RX	DOUT

- **하드웨어 Pin Mapping**

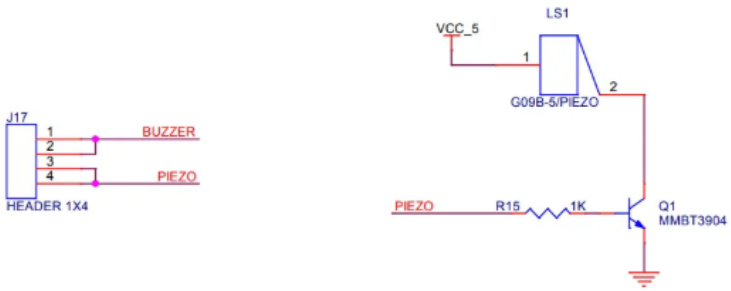


- **주변 장치 회로도**

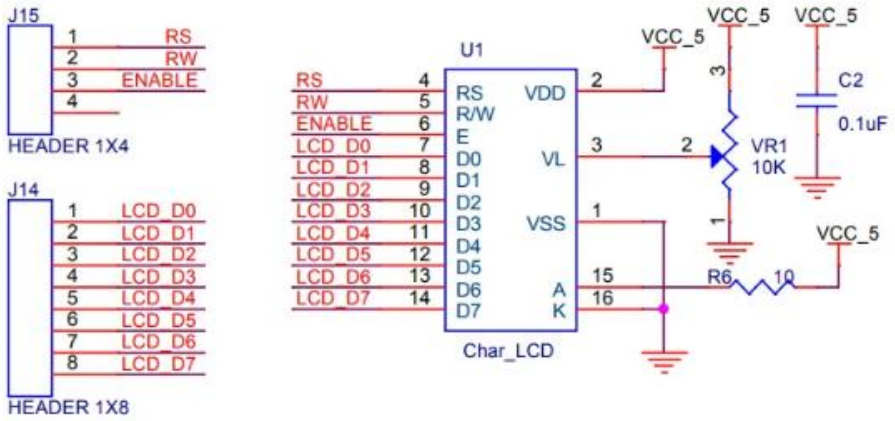
1) DC MOTOR



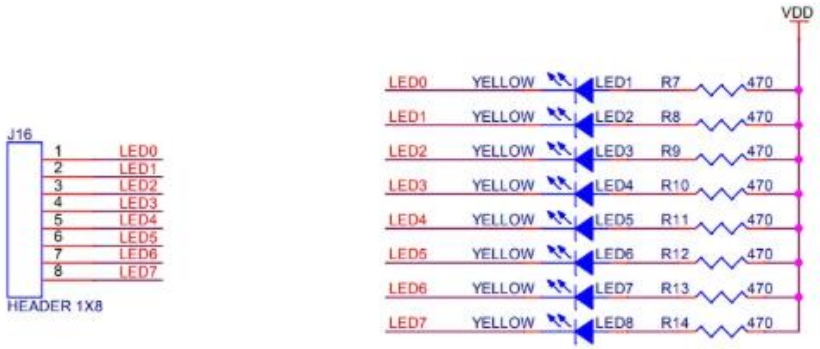
2) PIEZO



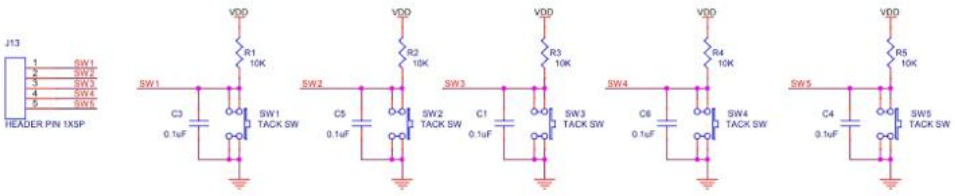
3) TEXT LCD



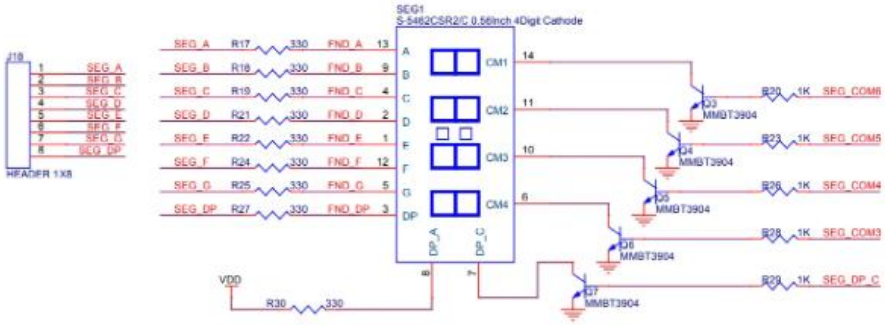
4) LED



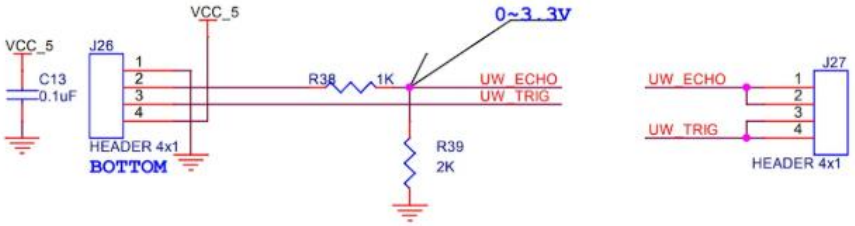
5) SWITCH



6) 7-Segment

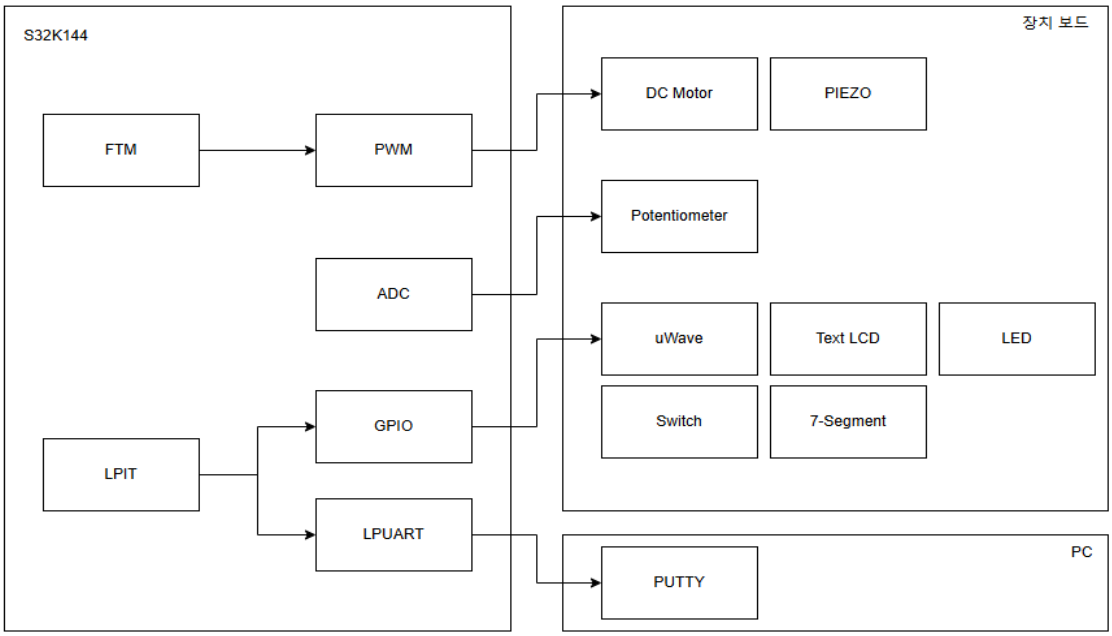


7) uWave Sensor



6. 하드웨어 및 소프트웨어 구성도

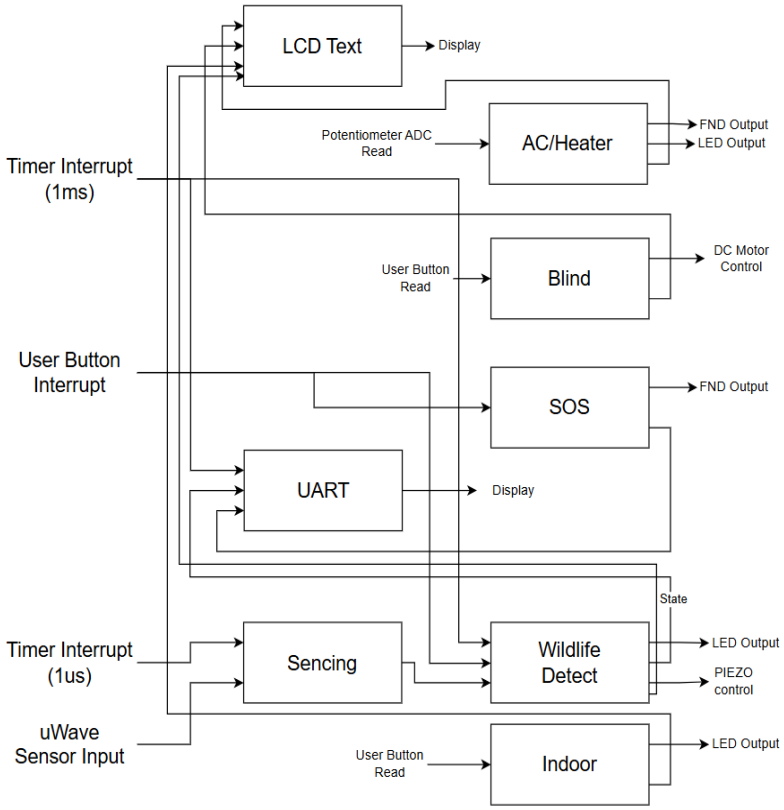
1. 하드웨어 구성도



S32K144 보드는 ADC 모듈로 가변저항 값을 읽어 PWM 을 통해 DC 모터를 제어한다. 또 다른 PWM 채널에서는 PIEZO 를 제어한다.

LPIT 타이머의 여러 채널을 통해 GPIO 와 LPUART 가 동기화된다. 이때 GPIO 모듈은 버튼 인터럽트, LCD, 초음파 센서, 7-segment 를 제어하고, LPUART 모듈을 통해 PC 에 데이터를 전송한다.

2. 소프트웨어 구성도



각 블록은 하나의 소프트웨어 모듈을 의미한다. 각 블록에는 상태, 컨트롤 신호, 데이터 입력이 들어가고 제어 신호와 데이터 출력이 나오게 된다.

7. 소스코드

소스 코드 전문은 아래 GitHub 링크를 통해 확인할 수 있다.

https://github.com/yjyjyjyjjjjyj/TermProject.git

8. 기능별 동작 설명

모듈 자체의 알고리즘에만 집중하기 위해 여러 기능에 따른 모듈화를 구현하였고, 이 모듈들은 총 4 가지의 트리거 조건에 따라 호출이 결정된다. 첫 번째로 LPIT CH0 ISR 은 매 1ms 마다 호출되며, 짧은 주기를 갖고 동작하는 모듈들이 포함된다. 다음으로 LPIO CH1 ISR 은 1us 마다 호출되며 10us 펄스를 트리거 하는 uWave 센서에 의해 구현되었다. PORTC ISR 에는 사용자 입력에 반응하여 동작해야 하는 모듈들이 포함되어 있다. 마지막으로 main 에 트리거 조건이 따로 없이 항상 실행되어야 하는 모듈들이 포함된다.

A. 에어컨 제어

① 온도 설정 및 ADC 입력

- 설정 온도를 입력 값으로 받아오기 위해, 가변 저항 값(전압)을 S32K144 의 ADC0 모듈에서 읽어온다. 입력 채널은 채널 13 으로 설정되어 있다.
- ADC_init() 함수는 ADC 클럭 소스를 SOSCDIV2 로 설정하고, 12 비트 해상도에서 동작하도록 ADC 모듈을 초기화한다.
- ADC 변환은 convertAdcChan(13) 함수로 시작되며, 변환이 완료되면 read_adc_chx() 함수를 호출하여 밀리볼트(mV) 단위의 변환 값을 얻는다.

● 온도 설정 매핑:

- ADC 값은 0-5000 mV 범위에서 측정되며, 이를 18 도에서 30 도 사이의 설정 온도로 매핑한다.
- 온도 설정 값은 LCD 및 FND 에 표시되며, 시스템 상태에 따라 변경된다.

② 현재 온도와 설정 온도 비교

- 현재 온도(curtemp)는 초기값 24 도로 고정되어 있다.
- ac_cur_target() 함수는 현재 온도와 설정 온도를 비교하여, 1 도씩 증가 또는 감소하며 목표 온도에 점진적으로 도달하도록 설계되었다.

● 상태 기반 제어: ac_oper() 함수

- 설정 온도가 현재 온도보다 낮으면 에어컨이 작동하며, 파란색 LED 가 점등된다.
- 설정 온도가 현재 온도보다 높으면 히터가 작동하며, 빨간색 LED 가 점등된다.
- 설정 온도와 현재 온도가 같으면 모든 장치가 비활성화되며, 초록색 LED 가 점등된다.

③ 상태 시각화: LED 제어

- 빨간색 LED (RED_ON): 히터 작동 중.
- 파란색 LED (BLUE_ON): 에어컨 작동 중.
- 초록색 LED (GREEN_ON): 목표 온도에 도달.
- LED 상태는 ac_oper() 함수 내에서 온도 비교 결과에 따라 동적으로 제어된다.

④ 현재 상태 표시: LCD 및 FND

● 온도 출력

- FND 는 4 개의 7-segment 를 사용하여 설정 온도와 현재 온도를 각각 두 자릿수로 표시한다.
- 왼쪽 두 자릿수(COM6, COM5): 설정 온도.
- 오른쪽 두 자릿수(COM4, COM3): 현재 온도.
- ac_temp_seg() 함수는 현재 온도와 설정 온도를 FND 에 출력하는 기능을 수행한다.

● LCD 출력: 상태 메시지

- AdcResultInMv >= 938: "AC Heater Turn On" 출력.
- AdcResultInMv < 938: "AC Heater Turn Off" 출력.
- 설정 온도가 현재 온도보다 낮을 경우: "AIRCONDITIONER ON".
- 설정 온도가 현재 온도보다 높을 경우: "HEATER ON".
- 설정 온도와 현재 온도가 같을 경우: "TARGET == TEMP".
- LCD 데이터 입력은 lcdcharinput() 함수로 제어되며, LCD 초기화는 lcdinit() 함수에서 이루어진다.

⑤ 온도 제어 타이밍: LPIT 타이머

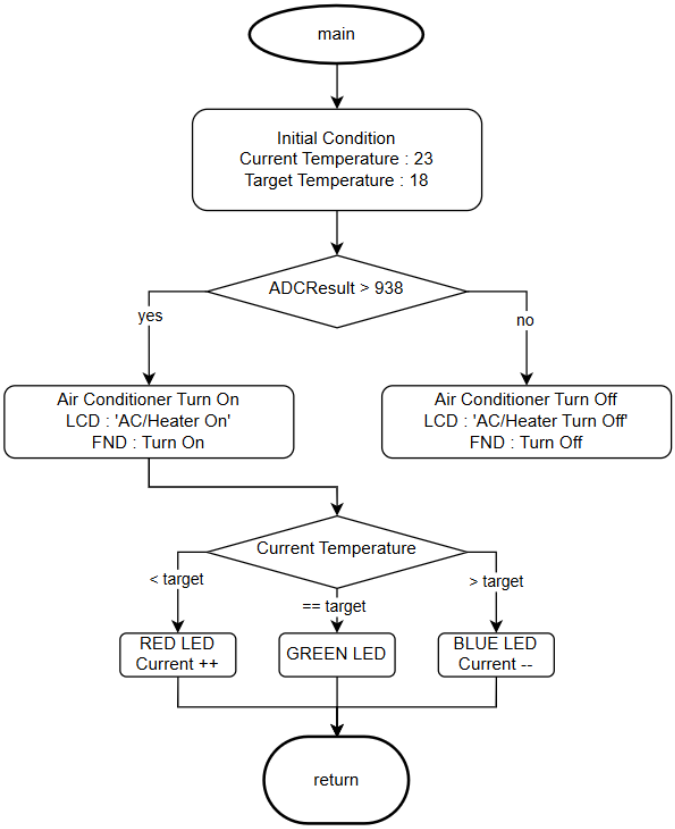
● 지연 시간 관리: LPIT 모듈 설정

- 온도 조정 및 ADC 변환은 LPIT0 모듈의 1ms 주기 타이머에 의해 동기화된다.
- delay_ms() 함수는 LPIT 채널 0 을 활용하여 지연 시간을 정확히 측정한다.

⑥ 동작 상태 관리

- 설정 온도에 도달하면 모든 LED 및 FND 가 꺼지고, 시스템은 대기 상태로 전환된다.

- ADC 값이 특정 기준치(예: AdcResultInMv < 938) 이하로 떨어지면 에어컨 및 히터는 비활성화된다.



<AC/Heater 모듈의 flow chart>

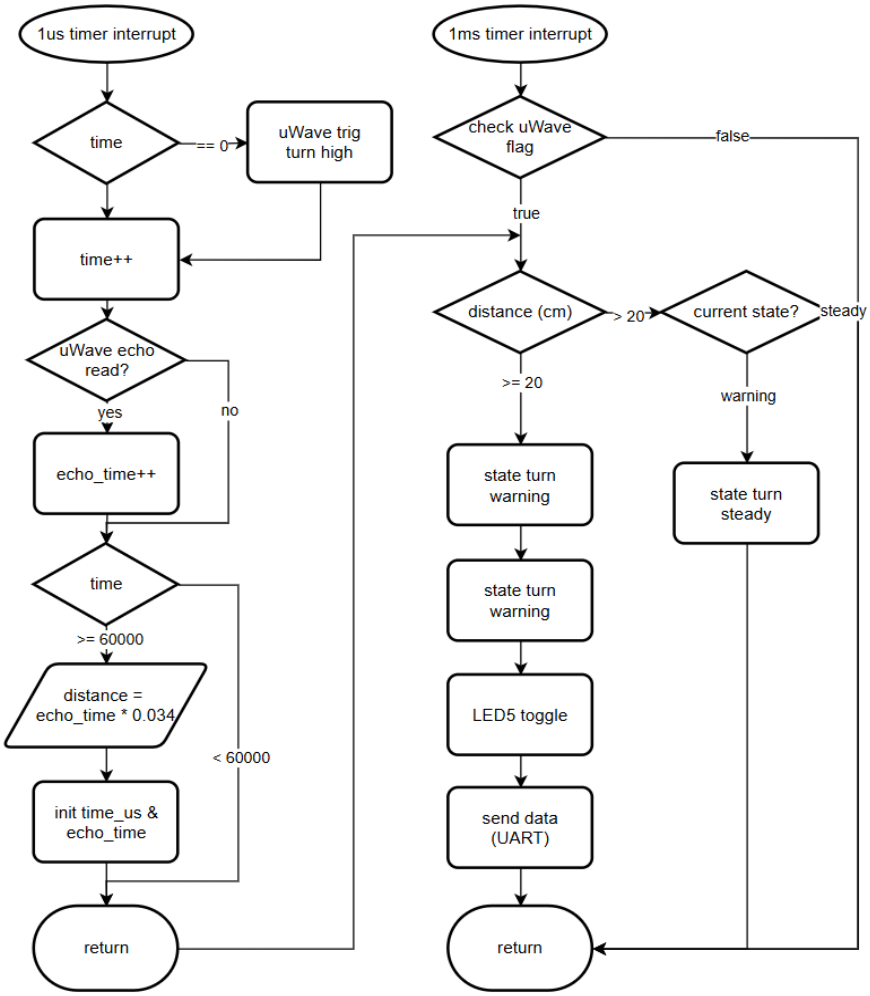
B. 야생동물 감지

① Trig 송신 / Echo 수신

- LPIT0 모듈의 채널 1 핸들러가 1us 마다 동작
- 채널 1 타이머에 따라 10us 동안 Trig 핀이 HIGH 상태로 유지
- 물체에서 반사된 신호가 Echo 핀에 의해 읽히는 시간 카운트
- 읽은 시간을 바탕으로 물체와의 거리 계산
- 1ms 주기의 LPIT0 채널 0 핸들러를 통해 현재 상태 업데이트

② WARNING-STATE

- 감지된 거리가 20cm 이내라면, WARNING-STATE 으로 현재 상태 변화
- LED5 를 연속적으로 toggle 하여 깜빡거리도록 함
- PIEZO 는 FTM2 의 채널 5 에 매핑됨
- 20cm 이내 거리 값에 반비례하여 부저 음량 변화



<uWave와 야생동물 감지 모듈의 flow chart>

③ STEADY-STATE 로의 복귀

- WARNING-STATE 일 때 SW5 을 3 초간 눌러 STEADY-STATE 로 복귀
- 1ms 타이머 인터럽트를 사용해 카운트가 3000 만큼 증가하면 상황 해제
- 부저와 LED5 를 OFF

④ UART 디버깅

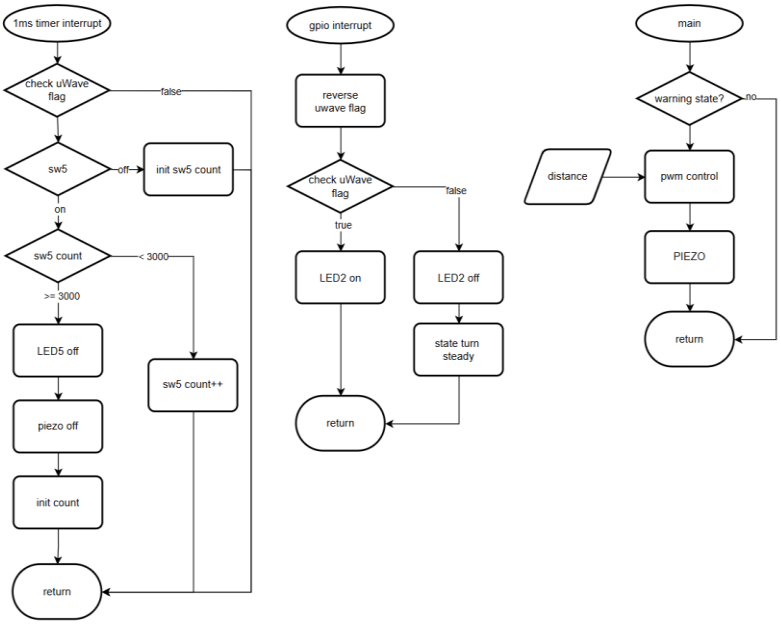
- LPUART 를 사용하여 현재 상태와 감지 거리를 시리얼 모니터로 출력

⑤ SW2 로 센서동작을 제어

- 위의 모든 동작들은 SW2 를 통해 제어
- SW2 를 누르면 인터럽트가 동작 시작 / 'WildLife Turn On' 출력 / LED2 Turn ON
- 후에 다시 SW2 를 누르면 인터럽트가 동작 시작 / 'WildLife Turn OFF' 출력 / LED2 Turn OFF

⑥ LED2 는 센서동작을 하고 있다는 것을 표시

- LED2 가 켜져있다면 Sensing 이 되고 있다는 의미 (야생동물 감지 시스템이 동작하는 것을 의미)
- LED2 가 꺼져있다면 Sensing 이 되지 않고 있다는 의미 (야생동물 감지 시스템이 동작하고 있지 않다는 것을 의미)



<야생동물 감지 모듈(2)의 flow chart>

C. 실내등 제어

① SW1 을 이용한 LED1 제어

- SW1 의 input 데이터를 처리하기 위해, 내부 상태 변수를 사용하여 토글 기능을 구현하였다.
- SW1 이 눌러지면 현재 상태를 확인하고, LED1 이 켜지거나 꺼지도록 제어한다. 이 동작은 inner_outter_control() 함수에서 구현된다.
- SW1 의 현재 입력 상태(current_state1)와 이전 상태(previous_state1)를 비교하여 SW1 의 토글 동작 여부를 판단한다.

● LED1 제어

- LED 제어는 LED1_ON 및 LED1_OFF 매크로를 사용해 구현되었다.
    - LED1_ON: PTE2 핀을 LOW 로 설정하여 LED 를 켬.
    - LED1_OFF: PTE2 핀을 HIGH 로 설정하여 LED 를 끔.

● 토글 기능 구현

- SW1 을 눌렀을 때 LED1 의 상태는 이전 상태와 반대가 되도록 설계되었다.
- SW1 의 입력이 감지되면, inner_outter_control() 함수 내에서 LED_state1 변수가 반전된다.
- LED_state1 이 1 일 경우 LED1 을 켜고, 0 일 경우 LED1 을 끈다.

② LED1 을 통한 현재 상태 시각화

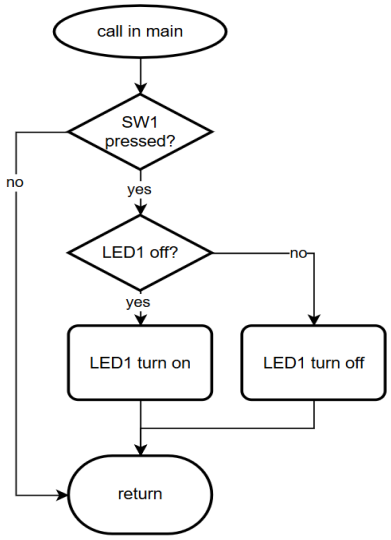
● ON/OFF 상태 표시

- LED1 의 상태는 실내등의 현재 상태를 나타낸다.
- LED1 이 켜지면 실내등이 ON 상태임을 나타낸다.
- LED1 이 꺼지면 실내등이 OFF 상태임을 나타낸다.
- 상태 전환은 SW1 의 입력에 따라 실시간으로 변경되며, 이를 통해 사용자는 현재 실내등 상태를 직관적으로 확인할 수 있다.

③ LCD 를 통한 상태 출력

- SW1 이 눌러져 LED1 이 켜질 경우: "Indoor LED Turn ON" 메시지를 출력한다.

- SW1 이 다시 눌러져 LED1 이 꺼질 경우: "Indoor LED Turn OFF" 메시지를 출력한다.
- 메시지 출력은 lcdcharinput() 함수와 연계되며, 문자열 데이터를 LCD 에 표시하도록 구현하였다. 출력된 메시지는 1 초 동안 유지되며, 이를 위해 delay_ms() 함수가 사용된다.
- 타이머는 LPIT 모듈의 1ms 주기를 기반으로 동작한다.



<실내등 모듈의 flow chart>

D. 블라인드 제어

① SW3 을 이용한 블라인드 동작 제어

- SW3 를 누르고 있을 때, 내부 카운터(cnt_blind)가 증가하도록 구현하였다.
- 카운터 값은 blind_control() 함수에서 증가하며, 특정 임계값에 도달하면 블라인드 상태가 변경된다.
    - cnt_blind == 0: 블라인드가 멈춘 상태.
    - cnt_blind == 3500: 블라인드가 내려가는 상태.
    - cnt_blind == 7000: 블라인드가 올라가는 상태.

● 상태 전환 로직

- 블라인드는 스위치를 한 번 떼었다가 다시 누르면 상태가 전환되며, SW3 의 입력 상태는 switch_released 플래그를 통해 관리된다.
    - SW3 처음 눌림: 블라인드 상태는 "내려가는 상태"로 전환.
    - SW3 다시 눌림: 블라인드 상태는 "올라가는 상태"로 전환.

② 7-Segment 를 통해 DC 모터 방향에 따른 로딩 표시

- 블라인드가 내려가는 동안 (DC 모터가 반시계 방향으로 회전): Loading_seg() 함수가 호출되어 반시계 방향 애니메이션을 출력한다.
- 블라인드가 올라가는 동안 (DC 모터가 시계 방향으로 회전): Inver_Loading_seg() 함수가 호출되어 시계 방향 애니메이션을 출력한다.

● 로딩 애니메이션 제어

- 로딩 애니메이션은 7-segment 의 소수점 자리를 사용하여 표시된다.
- loading_anim_index 변수는 애니메이션 프레임을 관리하며, delay_us() 함수로 프레임 간 지연 시간을 제어한다.

③ DC 모터 방향에 따른 블라인드 상태 제어

- PWM 신호는 FTM3->CONTROLS[2].CnV 및 FTM3->CONTROLS[3].CnV 값을 통해 제어된다.

● 블라인드 상태 제어:

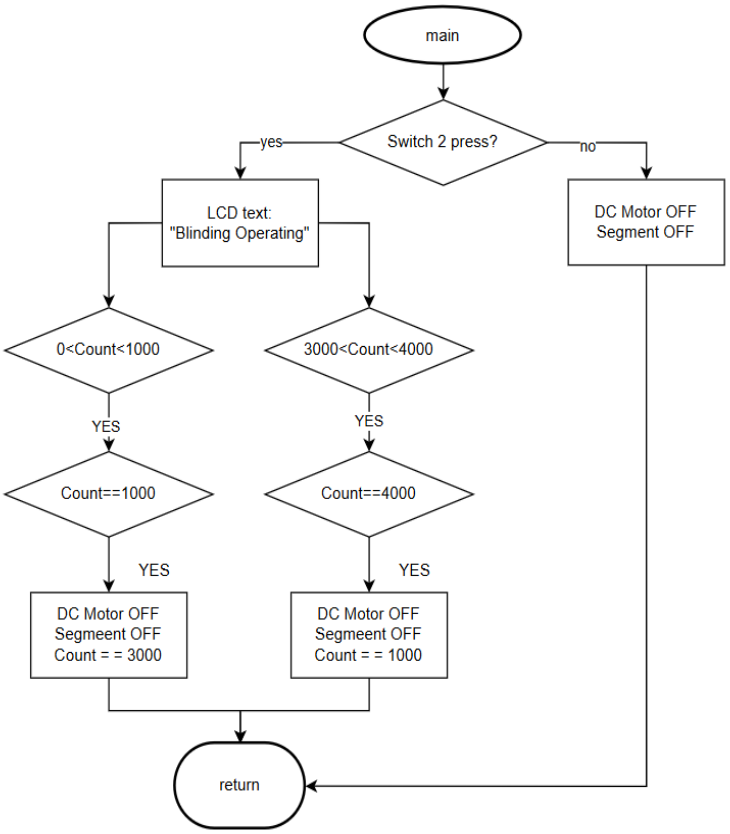
1. 내림 상태
    - DC 모터가 반시계 방향으로 회전하면 블라인드가 내려온다.
    - FTM3->CONTROLS[2].CnV 값은 최대, FTM3->CONTROLS[3].CnV 값은 0 으로 설정된다.

2. 올림 상태
    - DC 모터가 시계 방향으로 회전하면 블라인드가 올라갑니다.
    - FTM3->CONTROLS[2].CnV 값은 0, FTM3->CONTROLS[3].CnV 값은 최대값으로 설정된다.
3. 상태 전환

- blind_state 변수는 블라인드의 현재 상태를 나타내며, 값은 다음과 같이 정의된다.
    - blind_state == 0: 블라인드 멈춤.
    - blind_state == 1: 블라인드 내림.
    - blind_state == 2: 블라인드 올림.

④ LCD 를 통한 블라인드 상태 표시

- 블라인드가 동작하기 전에 LCD 에 "Blind Operating.." 메시지가 출력된다.
- 메시지는 lcdcharinput() 함수를 통해 출력되며, LCD 초기화는 lcdinit() 함수에서 이루어진다.
- 상태 출력은 블라인드의 동작 방향에 관계없이 동일한 메시지를 출력한다.



<블라인드 모듈의 flow chart>

E. SOS 모드

① SW4 를 이용한 SOS 동작 감지

● SW4 입력 감지 및 시간 추적:

- SW4 가 눌렸을 때 Falling Edge Interrupt 가 발생하며, 이는 PORTC_IRQHandler() 함수에서 처리된다.
- SW4 의 눌림 상태는 포트 C17 의 GPIO 입력 핀을 통해 확인되며, 눌림 시간은 LPIT0_Ch0_IRQHandler()에서 추적된다.
- LPIT0 타이머 채널 0 은 1ms 주기로 발생하여 sw4_counter 를 1 씩 증가시킨다.

● SW4 눌림 상태 분류 및 처리

- 3 초 미만 눌림 처리
    - SW4 가 3 초(3000ms) 미만으로 눌려 있는 경우, sw4_counter 와 sw4_flag 는 초기화되며 SOS 이벤트는 트리거되지 않는다.
    - SW4 가 떼어진 상태(SW4_ON == false)가 감지되면 플래그와 카운터를 초기화하여 불필요한 SOS 이벤트 트리거를 방지한다.
- 3 초 이상 눌림 처리:
    - ○ SW4 가 3 초 이상(3000ms 이상) 눌린 상태를 유지하면 sw4_counter >= 3000 조건에서 sos_event_flag 가 활성화된다.
    - ○ SOS_process() 함수가 호출되어 SOS 동작이 수행된다.
    - ○ 이후 카운터는 초기화되며, 시스템은 SOS 이벤트 처리를 시작한다.

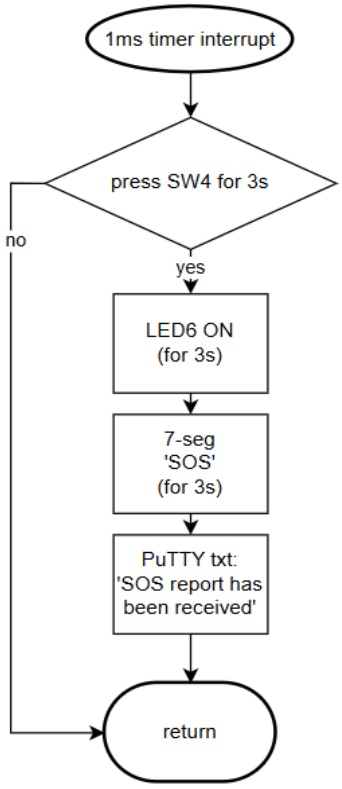
② SOS 발생 시 동작 처리

- SOS 이벤트가 감지되면 sos_event_flag 를 확인하여 모든 현재 동작을 중지한다.
- 블라인드, 실내등 등 다른 기능이 일시 중단되며, 시스템은 SOS 모드로 전환된다.
- SOS 이벤트가 발생하면 7-세그먼트를 사용해 "S", "O", "S" 문자를 반복적으로 출력한다.

- Segment_display() 함수는 각 문자를 개별적으로 출력하며, 짧은 지연을 추가해 SOS 신호를 시각적으로 명확히 전달한다.
- LPUART1 을 통해 "SOS report has been received" 메시지를 전송합니다.
- 긴급 상태를 시각적으로 표시하기 위해 LED5 가 켜진다.

③ SOS 종료 처리

- SOS 상태가 종료되면 LED5 를 끄고 시스템이 정상 상태로 복귀했음을 나타낸다.
- SOS 메시지 출력이 종료되면 7-세그먼트 디스플레이를 초기화하여 데이터를 삭제한다.
- sos_event_flag 를 초기화하여 시스템이 대기 상태로 복귀하도록 하여 다른 기능들이 정상적으로 작동할 수 있도록 보장한다.



<SOS 모듈의 flow chart>

9. 결과 및 고찰

이번 프로젝트에서 구현한 개별 모듈들은 독립적으로는 기대한 만큼의 동작이 확인되었지만, 모듈 통합 이후 몇 가지 매끄럽지 못한 부분이 있었다. 이러한 문제는 주로 딜레이 시간 설정과 타이머 인터럽트 주기 설정의 모호성 때문이라고 판단했다.

특히 초음파 센서의 트리거와 에코 핀 제어를 위해 설정된 타이머 인터럽트의 주기 선택에서 많은 고민이 있었다. 1us 단위와 10us 단위 중 어떤 주기를 사용할지 명확하게 결정하지 못한 점이 시스템 성능과 정확성에 영향을 미쳤다고 생각한다. 결국 센서 작동을 켜고 끄는 결정을 하여 어느 정도 미흡한 점을 해결하였다. 이러한 주기 설정은 시스템 성능과 실시간 처리에 따라 더욱 정밀한 테스트와 세부적인 모듈화를 통해 최적화할 필요가 있음을 몸소 깨달았다.

SOS 모듈과 AC/Heater 모듈과 같이 7-Segment 를 사용할 때와 LCD 를 사용할 때 딜레이 함수(타이머 인터럽트)로 인해 많은 CPU 리소스를 사용하게 되었다. 이로 인해 시스템이 다른 작업을 수행할 충분한 시간을 확보하지 못하게 되었고, 결과적으로 다른 모듈들의 동작에 지연이 발생하였다. 이를 해결하기 위해 딜레이 시간을 조절하였다. 그 결과, 7-Segment 의 출력은 명확하게 나타났지만, 다른 모듈의 동작에 여전히 영향을 미치는 듯 보였다.

결론적으로, 더 체계적이고 나은 임베디드 시스템을 구축하기 위해서는 세분화된 모듈화가 필요하다는 것을 깨달았다. 또한 MCU 와 사용자 프로그램 간의 계층을 명확하게 분리하고 독립적인 사용자 프로그램을 구현함으로써 코드의 유지 및 보수를 더욱 원활하게 하고, 시스템의 신뢰성을 높일 수 있겠다고 생각했다.

또한, 각 소프트웨어 모듈과 하드웨어 장치의 동작 원리를 충분히 이해하지 않으면 문제 발생 시 이를 파악하고 해결하는 데 많은 시간이 소요된다는 것을 깨달았다. 구현 초기에는 GPIO 핀을 회로도를 기반으로 설정했지만, 프로그램 실행 후 예상대로 동작하지 않아 원인을 확인해 보니 특정 핀에 여러 기능이 중첩되어 있었다. 이를 통해, 잘못된 핀 선정으로 하드웨어와 소프트웨어의 동작 간 불일치가 발생해 시스템 동작에 문제가 발생할 수 있다는 점을 체감했다. 향후에는 초기 설계 단계에서 데이터 시트와 핀 매핑을 꼼꼼하게 검토할 필요성을 느꼈다.