11조 임베디드시스템설계및실험 텀프로젝트 보고서

2024.12.23.

201824607 편경찬, 201812157 조영진, 202255673 최민기

요 약 사용자의 편리함을 증대시키고 자동화된 환경을 제공하는 Home IoT System Interface 구성해 센서들을 제어하고 동작 결과를 기술한다.

• 목차

1. **서론**
- 프로젝트 목적
- 세부 목표
- 핵심 용어 정리
- 시나리오

2. **모듈 별 기능 설명**
- 핀 맵핑
- 초음파 센서
- LCD 센서
- 조도센서
- LED 센서
- 모터 센서
- 버튼 인터럽트
- 블루투스 통신

3. **동작 결과**

4. **실험 결론 및 제언**

I 서론

• 프로젝트 목적

1. 스마트 홈 IoT 시스템의 설계와 구현 원리를 이해.
2. STM32 보드와 센서를 활용한 자동화 시스템의 제어 구현.

• 세부 목표

1. STM32 환경에서 GPIO, ADC, PWM, DMA의 원리를 학습
2. 공식 문서를 참조하여 핀(Pin) 설정
3. 초음파 센서, 조도 센서, 모터, LED를 통합 제어
4. 사용자 모드와 자동화 모드의 구분 및 전환 구현

• 핵심 용어 정리

1. GPIO: 범용 입출력 핀으로, 센서와 모터 등을 제어
2. PWM: 펄스 폭 변조로 LED 밝기 및 모터 속도 제어
3. DMA: 데이터 전송 속도 향상을 위한 메모리 접근 기법
4. EXTI: 외부 인터럽트를 통해 버튼 및 초음파 신호 처리

II 모듈 별 기능 설명

• 핀 맵핑

Table. 1 - Pin Mapping

Pin	00	01	02	03	04	05	06	07	08	09	10	11	12	13
PA	BT4	Led								TX1	RX1	Trg	Ech	
PB		Adc	Adc				EN1	IN1			BT3			
PC	EN2	IN3			BT1								TIM	
PD						TX2	RX2							

다수의 핀을 설정해야 했으므로, 핀 간 충돌을 방지하기 위해 레퍼런스를 참고하며 AFPP 방식의 리매핑을 활용하였다. 또한, 동일한 채널을 공유해야 하는 핀들을 적절히 배치하여 원활한 제어를 달성했다.

Table. 2 - NVIC Setting

NVIC	EXTI4	EXTI15_10	EXTI0	USART1	USART2	TIM2
Pre-emp	0	0	0	0	1	0
Sub-Pri	0	1	2	0	1	1

다수의 인터럽트 요청이 발생했을 때, 위와 같은 우선순위를 기준으로 처리 순서를 결정하였다. 이를 통해 시스템의 인터럽트 처리를 최적화하고 동작의 실시간성을 보장하였다.

• 초음파 센서

1. 역할

초음파 센서는 사람이 집에 들어왔는지 감지하여 시스템을 활성화하는 역할을 한다. 초음파 신호를 사용하여 거리 데이터를 계산하고 이를 기반으로 시스템의 활성화 유무를 결정한다.

2. 구현

- Timer_Configuration()

```
    TIM_TimeBaseInitTypeDef TIM_InitStructure;
    TIM_InitStructure.TIM_Prescaler = 72 - 1;
    TIM_InitStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_InitStructure.TIM_Period = 0xFFFF;  // 최대 타이머 값 설정
    TIM_InitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_TimeBaseInit(TIM1, &TIM_InitStructure);
    TIM_Cmd(TIM1, ENABLE);
```

- TriggerPulse() & personCheck()

```
void TriggerPulse(void) {
    GPIO_WriteBit(GPIOA, GPIO_Pin_11, Bit_SET);
    for (volatile int i = 0; i < 720; i++);
    GPIO_WriteBit(GPIOA, GPIO_Pin_11, Bit_RESET);
}
```

```
void personCheck(void) {
    while (1) {
        TriggerPulse();
        for (volatile int i = 0; i < 1000000; i++) ;
        float distance = (pulseWidth * 0.0343) / 2;
        if((int)distance < 10) {
            return;
        }
    }
    return;
}
```

- EXTI15_10_IRQHandler()

```
void EXTI15_10_IRQHandler(void) {
    // ECHO - TIM1 PA10
    if (EXTI_GetITStatus(EXTI_Line12) != RESET) {
        if (GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_12) == Bit_SET) {
            TIM_SetCounter(TIM1, 0);
            echoState = 1;
        } else {
            pulseWidth = TIM_GetCounter(TIM1);
            echoState = 0;
        }
        EXTI_ClearITPendingBit(EXTI_Line12);
    }
```

Timer_Configuration()에서 72MHz 시스템 클럭을 72로 나누어 1μs 단위의 시간 측정이 가능하도록 설정한다. **TriggerPulse() & personCheck()**에서 초음파 센서는 trigger pin을 가동해 10μs 동안 펄스를 전송하고 echo pin을 통해 물체에 반사되어 돌아오는 데 걸리는 시간을 측정한다. 이때 TIM1 타이머를 사용해 echo pin의 HIGH 상태 지속시간을 정확히 측정하도록 했다. 초음파 센서는 물체까지의 거리를 '거리 = 시간 x 속도 / 2'의 식을 사용해 계산한다. **personCheck()** 함수는 초음파 송신을 위해 trigger pin에 펄스를 보낸 후, 초음파 센서로부터 10cm 이내에 물체가 감지되면 사용자가 입장한 것으로 간주하도록 설계됐다. Echo IRQ에서 Echo가 HIGH로 변했을 때, 타이머를 리셋하여 시간을 측정하며 Echo가 LOW로 변했을 때, 타이머 값으로 시간 측정을 완료한다.

• LCD 센서

1. 역할

사용자가 입장해 초음파 센서로 인식되면 LCD가 켜진다. LCD에는 두 개의 조도 센서로 읽은 조도값과 그 둘을 배합한 mood_value 값이 실시간으로 표시된다.

2. 구현

- LCD_DisplayStatus()

```
void LCD_DisplayStatus(void) {
    /* 생략 */
    char buffer3[64];
    memset(buffer3, 0, sizeof(buffer3));
    sprintf(buffer3, "mood_value : ");
    LCD_ShowString(10, 130, buffer3, BLACK, WHITE);
    LCD_ShowNum(140, 130, mood_value, 4, BLACK, WHITE);
}
```

- main()

```
    LCD_Init();
    Touch_Configuration();
    //Touch_Adjust();
    LCD_Clear(WHITE);
```

memset을 이용해 항상 배열을 초기화 해주어 쓰레기 값이 들어가지 않도록 유지한다.
LCD_ShowString과 LCD_ShowNum을 이용해 LCD에 각각 문자열과 정수를 출력한다.
main 함수에서는 LCD_Init 및 LCD_Clear 함수를 호출하여 매 사이클마다 LCD를 초기화한다.

• 조도 센서

1. 역할

2개의 CDS 조도 센서를 활용하여, 하나는 외부의 광량을. 다른 하나는 실내의 밝기를 측정한다.

2. 구현

- DMA_Configure() & ADC_Configure()

```
void DMA_Configure(void) {
    DMA_InitTypeDef DMA_InitStructure;
    DMA_DeInit(DMA1_Channel1);
```

```
    DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)&ADC1->DR;
    DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)&adc_values[0];
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
    DMA_InitStructure.DMA_BufferSize = sizeof(adc_values)/sizeof(adc_values[0]);
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
    DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
    DMA_InitStructure.DMA_Priority = DMA_Priority_High;
    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
    DMA_Init(DMA1_Channel1, &DMA_InitStructure);
    DMA_Cmd(DMA1_Channel1, ENABLE);
}

void ADC_Configure(void) {
    ADC_InitTypeDef ADC_InitStructure;
    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_InitStructure.ADC_NbrOfChannel = 2;
    ADC_InitStructure.ADC_ScanConvMode = ENABLE;
    // 조도센서1 채널 설정 (PB0 - ADC_Channel_8)
    ADC_RegularChannelConfig(ADC1, ADC_Channel_8, 1, ADC_SampleTime_239Cycles5);
    // 조도센서2 채널 설정 (PB1 - ADC_Channel_9)
    ADC_RegularChannelConfig(ADC1, ADC_Channel_9, 2, ADC_SampleTime_239Cycles5);
    ADC_Init(ADC1, &ADC_InitStructure); //ADC_ITConfig 대신 사용
    ADC_DMACmd(ADC1, ENABLE);
    ADC_Cmd(ADC1,ENABLE);
    ADC_ResetCalibration(ADC1);
    while(ADC_GetResetCalibrationStatus(ADC1) != RESET) ;
    ADC_StartCalibration(ADC1);
    while(ADC_GetCalibrationStatus(ADC1)) ;
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}
```

조도 센서를 읽는 방식으로 처음에는 인터럽트 방식을 사용했다. 그러나 이 방식은 각 조도 센서 핀마다 별도의 인터럽트를 설정해야 했으며, 센서 간 번갈아 가며 인터럽트가 발생해야 했기 때문에 지연시간이 증가하고 코드 복잡성도 높아졌다. 이를 개선하기 위해 DMA 방식을 채택하여 CPU의 개입 없이 주변 장치와 메모리 간 데이터 전송이 가능하도록 설정했다. 레퍼

런스를 참고하여 DMA를 ADC의 두 채널(채널 8과 채널 9)에 각각 조도 센서 1과 2를 매핑해 아날로그 값을 읽도록 구성했습니다. 이 환경을 구성하기 위해 DMA는 배열의 길이(조도 센서 개수)인 버퍼 크기 2만큼을 Circular 모드로 읽도록 설정했으며, ADC의 continuous 및 scan conversion 모드를 활성화하여 여러 채널의 데이터를 지속적으로 샘플링할 수 있도록 했다.

• LED 센서

1. 역할

스마트 홈 IoT 시스템의 자동 밝기 무드등 기능을 담당한다. 두 조도값을 적절하게 계산하여 스마트 홈 내부에 최적화된 밝기를 제공한다. 즉 주변 밝기에 따라 LED의 밝기를 조정하여 사용자에게 최적화된 밝기를 제공하는 역할을 한다. 이를 통해 불필요한 에너지 낭비를 줄이면서도 효율적인 밝기를 제공할 수 있다.

2. 구현

- Timer_Configuration()

```
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    TIM_OCInitTypeDef OutputChannel;
    TIM_TimeBaseStructure.TIM_Period = 100 - 1;  // 100kHz
    TIM_TimeBaseStructure.TIM_Prescaler = 24 - 1;
    TIM_TimeBaseStructure.TIM_ClockDivision = 0;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
    OutputChannel.TIM_OCMode = TIM_OCMode_PWM1;
    OutputChannel.TIM_OutputState = TIM_OutputState_Enable;
    OutputChannel.TIM_OutputNState = TIM_OutputNState_Enable;
    OutputChannel.TIM_Pulse = 50 - 1;  // 50% duty ratio
    OutputChannel.TIM_OCPolarity = TIM_OCPolarity_Low;
    OutputChannel.TIM_OCNPolarity = TIM_OCNPolarity_High;
    OutputChannel.TIM_OCIdleState = TIM_OCIdleState_Set;
    OutputChannel.TIM_OCNIdleState = TIM_OCIdleState_Reset;
    TIM_OC2Init(TIM2, &OutputChannel);
    TIM_Cmd(TIM2, ENABLE);
    TIM_ITConfig(TIM2, TIM_IT_Update | TIM_IT_CC2, ENABLE);
```

- TIM2_IRQHandler()

```
void TIM2_IRQHandler(void) {
    if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET) {
```

```
        TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
        GPIOC->BRR = GPIO_Pin_12;
    }
    if (TIM_GetITStatus(TIM2, TIM_IT_CC2) != RESET) {
        TIM_ClearITPendingBit(TIM2, TIM_IT_CC2);
        GPIOC->BSRR = GPIO_Pin_12;
    }
}
```

Timer_Configuration() 함수에서 Prescaler 설정으로 타이머 클럭은 72MHz에서 24로 나누어 3MHz로 설정되며, Period 설정으로 PWM 주파수는 3MHz를 100으로 나누어 30kHz로 구성된다. LED는 TIM2의 CCR2 채널을 통해 PWM 신호를 출력하여 밝기를 조절하도록 설계되었다. 자동화 시스템에서는 조도 센서로 읽어들인 두 개의 값을 연산하고 매핑하여, 일조량이 낮아지면 LED 밝기가 증가하고, 일조량이 높아지면 LED 밝기가 감소하도록 설정했다. **TIM2_IRQHandler()** 함수는 타이머2의 오버플로 이벤트 발생 시 호출되며, TIM2의 주기(Period)와 비교 이벤트(CC2)에 따라 PC12 핀이 점등 및 소등되어 LED가 깜빡이는 동작을 수행한다.

• 모터 센서

1. 역할

모터는 선풍기의 역할로써 스마트 홈 IoT 시스템 내 풍량을 조정하여 사용자에게 쾌적한 환경을 제공해주는 역할을 한다. cds 조도센서의 센서값을 이용하여 구한 mood_value값을 바탕으로 바람의 세기를 조절한다. 이때 두 개의 모터를 각각 제어하여 가동할 모터 수를 결정한다. 일조량이 크면 체감 온도가 올라가기에 모터 2개가 가동되고, 햇빛이 약하면 가동하는 모터 수가 줄어든다.

2. 구현

- Motor_Control(int motor, int enable)

```
void Motor_Control(int motor, int enable) {
    if (motor == 1) {  // 모터 1 제어
        if (enable) {
            GPIO_WriteBit(GPIOB, GPIO_Pin_6, Bit_SET);
            GPIO_WriteBit(GPIOB, GPIO_Pin_7, Bit_SET);
        } else {
            GPIO_WriteBit(GPIOB, GPIO_Pin_6, Bit_RESET);
        }
```

```
    } else if (motor == 2) {  // 모터 2 제어
        if (enable) {
            GPIO_WriteBit(GPIOC, GPIO_Pin_0, Bit_SET);
            GPIO_WriteBit(GPIOC, GPIO_Pin_1, Bit_SET);
        } else {
            GPIO_WriteBit(GPIOC, GPIO_Pin_0, Bit_RESET);
        }
    }
}
```

모터 제어를 위해 EN 및 IN 핀(PB6, PB7, PC0, PC1)을 GPIO 출력 모드로 설정한다. Motor_Control 함수는 enable의 상태에 따라 모터 1과 2를 활성화 및 비활성화 시킨다. EN 핀과 IN 핀이 모두 HIGH인 경우 모터가 활성화 되며 EN 핀이 LOW이면 모터는 비활성화 된다.

• 버튼 인터럽트

1. 역할

버튼 1, 3, 4는 각각 인터럽트 핸들러 4, 15_10, 0에 등록되며 버튼 1은 모터를 on/off, 버튼 3은 LED 무드등을 on/off 해 사용자 모드로 진입하게 한다. 버튼 4를 누르면 다시 시스템 제어 모드로 전환된다.

2. 구현

- EXTI4_IRQHandler()

```
// button 1
void EXTI4_IRQHandler(void) {
    // motor1, 2 on/off
    if (EXTI_GetITStatus(EXTI_Line4) != RESET) {
        if (GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_4) == Bit_RESET) {
            user_mode = true;
            BitAction motor1_state = GPIO_ReadOutputDataBit(GPIOB, GPIO_Pin_6);
            if (motor1_state == Bit_RESET) {
                Motor_Control(1, 1);
                Motor_Control(2, 1);
            } else {
                Motor_Control(1, 0);
                Motor_Control(2, 0);
            }
```

```
        }
        EXTI_ClearITPendingBit(EXTI_Line4);
    }
}
```

- EXTI15_10_IRQHandler()

```
// button 3 - LED on/off
    if (EXTI_GetITStatus(EXTI_Line13) != RESET) {
        if (GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_13) == Bit_RESET) {
            user_mode = true;
            if(onoff) {TIM2->CCR2 = 0; onoff = false;}
            else {TIM2->CCR2 = 100; onoff = true;}
        }
        EXTI_ClearITPendingBit(EXTI_Line13);
    }
```

- EXTI10_IRQHandler()

```
// button 4 - System Mode
void EXTI0_IRQHandler(void) {
    if (EXTI_GetITStatus(EXTI_Line0) != RESET) {
        if (GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0) == Bit_RESET) {
            user_mode = false;
        }
        EXTI_ClearITPendingBit(EXTI_Line0);
    }
}
```

EXTI15_10_IRQHandler()는 GPIOC의 Pin 13 버튼(버튼 3)을 눌러 인터럽트가 발생하면, LED 밝기를 TIM2->CCR2 값을 토글하여 제어한다.
모든 인터럽트 핸들러는 EXTI_ClearITPendingBit를 호출하여 인터럽트 플래그를 클리어한다.

• 블루투스 통신

1. 역할

USART1은 Putty와 연결되며 USART2는 Bluetooth와 연결하여 두 USART 간의 양방향 데이터 통신을 가능하게 되며, 각각의 데이터 수신에 대해 다음의 작업을 수행할 수 있다.

2. 구현

- USART1_Init() & USART2_Init()

```
void USART1_Init(void) {
    USART_InitTypeDef USART1_InitStructure;
    USART_Cmd(USART1, ENABLE);
    USART1_InitStructure.USART_BaudRate = 9600;
    USART1_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART1_InitStructure.USART_StopBits = USART_StopBits_1;
    USART1_InitStructure.USART_Parity = USART_Parity_No;
    USART1_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART1_InitStructure.USART_HardwareFlowControl
    = USART_HardwareFlowControl_None;
    USART_Init(USART1, &USART1_InitStructure);
    USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
}

void USART2_Init(void) {
    USART_InitTypeDef USART2_InitStructure;
    USART_Cmd(USART2, ENABLE);
    USART2_InitStructure.USART_BaudRate = 9600;
    USART2_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART2_InitStructure.USART_StopBits = USART_StopBits_1;
    USART2_InitStructure.USART_Parity = USART_Parity_No;
    USART2_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART2_InitStructure.USART_HardwareFlowControl
    = USART_HardwareFlowControl_None;
    USART_Init(USART2, &USART2_InitStructure);

    GPIO_PinRemapConfig(GPIO_Remap_USART2, ENABLE); // Remap 활성화
    USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);
}
```

- sendDataUART2()

```
void sendDataUART2(void) {
    char msg[] = "User Entered\r\n";
    int i = 0;
    while(msg[i] != '\0') {
        while ((USART2->SR & USART_SR_TC) == 0);
        USART_SendData(USART2, msg[i]);
        i++;
    }
}
```

- USART1_IRQHandler() & USART2_IRQHandler()

```
void USART1_IRQHandler() {
    uint16_t word;
    if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET) {
        word = USART_ReceiveData(USART1);
        USART_SendData(USART2, word);
        USART_ClearITPendingBit(USART1, USART_IT_RXNE);
    }
}

void USART2_IRQHandler(void) {
    uint16_t word;
    if (USART_GetITStatus(USART2, USART_IT_RXNE) != RESET) {
        word = USART_ReceiveData(USART2);
        USART_SendData(USART1, word);
        switch (word) {
            case 'a':
                Motor_Control(1, 1);
                Motor_Control(2, 1);
                break;
            case 'b':
                Motor_Control(1, 1);
                Motor_Control(2, 0);
                user_mode = true;
                break;
            case 'c':
                Motor_Control(1, 0);
                Motor_Control(2, 0);
                user_mode = true;
                break;
            case 'r':
                user_mode = false; // 사용자 모드 해제
                break;
            case 'q':
                quitFlag = true;
                break;
            default:
                break;
        }
        USART_ClearITPendingBit(USART2, USART_IT_RXNE);
```

```
    }
}
```

사용자가 블루투스 터미널로 'a'를 입력하면 두 개의 모터가 동작하고, 'b'를 입력하면 한 개의 모터가 동작하며, 'c'를 입력하면 모든 모터가 꺼지도록 구성했다. 이 동작들은 모두 사용자 모드로 진입하는 인터럽트에 해당하고, 반대로 'r'을 입력하면 사용자 모드를 해제하고 시스템 모드로 돌아가도록 구현했다. 마지막으로 'q'를 입력하면 사용자가 퇴실했다는 신호로 간주하며, 모든 센서를 초기화하고 초음파 센서만 동작하도록 설정했다. 이후 사용자가 다시 입실하면 초음파 센서에 의해 동일한 기능들이 자동으로 수행된다. 이런 동작은 IoT 시스템의 자동화를 통해 사용자의 편의성을 높이는 데 기여한다. **sendDataUART2()** 함수는 초음파 센서가 사람을 감지하면 블루투스를 통해 "User Entered"라는 알림 메시지를 전송한다.

• 메인 함수

1. 역할

시스템 셋팅과 프로퍼티들을 초기화 시킨 후, 초음파 센서를 이용해 사용자가 입실할 때까지 대기한다. 사용자가 접근해 초음파 센서가 인식하면 이후의 시스템 모드들이 활성화된다. 외/내부의 조도 값을 읽은 후 평균을 낸 뒤 TIM2 타이머의 CCR2 채널의 포맷에 맞게 0에서 100으로 맵핑을 해 LED 무드등을 제어한다. mood_value가 70 이상이면 모터 2개, 50 이상이면 1개, 50 이하이면 0개가 동작하도록 자동화한다.

2. 구현

- **main()**

```
    /* 시스템 셋팅 생략 */

    personCheck(); // 초음파 대기
    sendDataUART2();
    while (1) {
        if(quitFlag) {
            quitFlag = false;
            goto Start;
        }
        while (DMA_GetFlagStatus(DMA1_FLAG_TC1) == RESET) ;
        DMA_ClearFlag(DMA1_FLAG_TC1);
        LCD_DisplayStatus();
        Delay();
        if(!user_mode) {
```

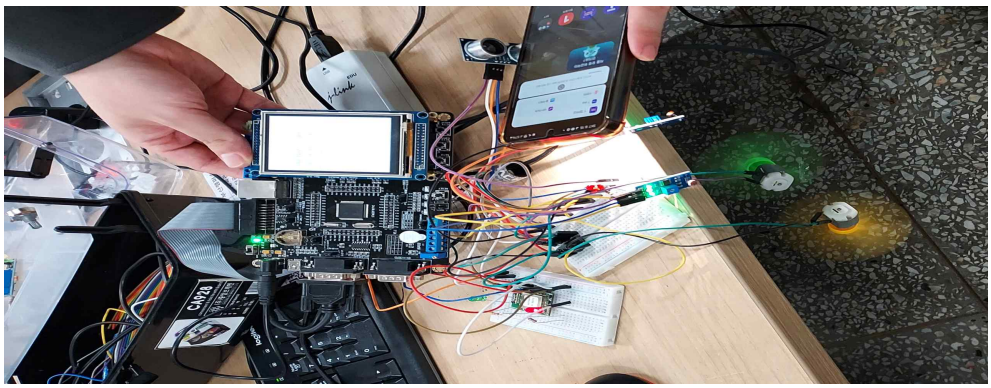
```
            __disable_irq();
            jodo = (adc_values[0] + adc_values[1]) / 2;
            mood_value = ((jodo - ADC_MIN) * (OUTPUT_MAX – OUTPUT_MIN))
                         / (ADC_MAX - ADC_MIN) + OUTPUT_MIN;
            mood_value = 100 - mood_value;
            TIM2->CCR2 = mood_value;
            if(mood_value >= 70) {
                Motor_Control(1, 1);
                Motor_Control(2, 1);
            }
            else if(mood_value >= 50) {
                Motor_Control(1, 1);
                Motor_Control(2, 0);
            }
            else if(mood_value < 50) {
                Motor_Control(1, 0);
                Motor_Control(2, 0);
            }
            __enable_irq();
        }
        // 사용자 모드
        else {

        }
    }
```

조도 센서는 저항 값에 반비례하므로 어두워질수록 ADC 값이 증가한다. TIM2->CCR2는 값이 작아질수록 LED는 더 밝게 점등된다. 이를 기반으로 100 - mode_value를 사용해 무드등의 밝기를 제어했고, 최종 무드 값이 낮아지면 LED는 더 밝아지도록 설정했다. 이와 함께, 무드 값이 낮아지면 모터의 동작이 줄어들고, 일조량이 증가해 밝아질수록 더 많은 모터가 동작하도록 구현했다.

시스템 제어 모드의 코드 섹션은 **__disable_irq()**와 **__enable_irq()**를 사용하여 인터럽트를 비활성화하고 다시 활성화하는 방식으로 임계 구역(critical section)을 설정했다. 만약 위와 같은 방식으로 임계구역을 설정하지 않는다면, 시스템 제어 모드 안의 조건 절이 수행되고 있을 때 인터럽트가 발생해 사용자 모드로 전환되어 특정 센서값을 변경을 한 뒤, 다시 원래 코드의 실행 흐름으로 돌아왔을 때 간헐적으로 센서값을 재변경 해버릴 위험이 있다. 즉 인터럽트 핸들러가 실행되었지만 반영이 되지 않는 코드가 된다. 따라서 안전한 데이터 처리와 원치 않는 동작을 방지하기 위해 시스템 제어 모드의 분기절은 원자성이 보장되도록 설정했다.

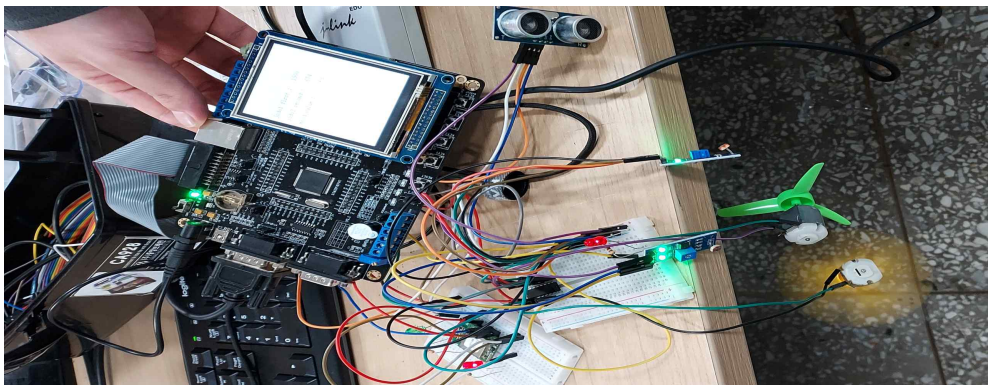
III 동작 결과

Fig. 1 - System Mode (mood_value >= 70)



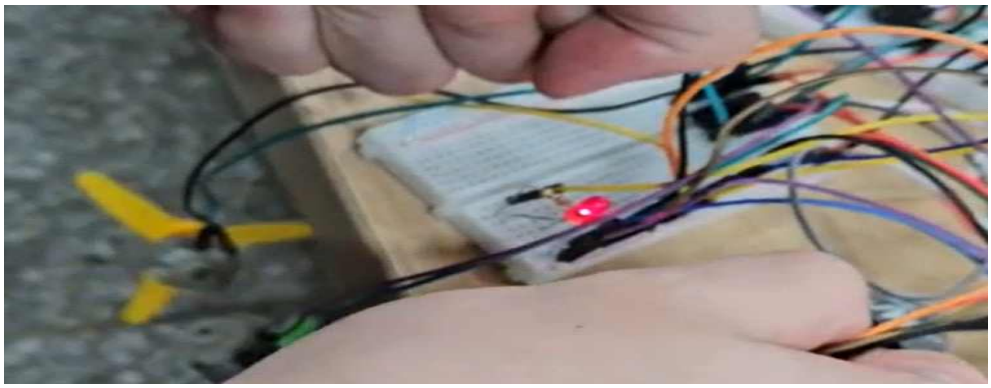
mood_value가 70을 넘었을 때 2개의 모터가 모두 작동하고 LED 무드등은 가장 낮은 밝기를 디스플레이 했다. 70을 넘기기 위해 플래시 라이트를 이용해 의도적으로 광량을 증가시킨 모습이다.

Fig. 2 - System Mode (50 <= mood_value < 70)



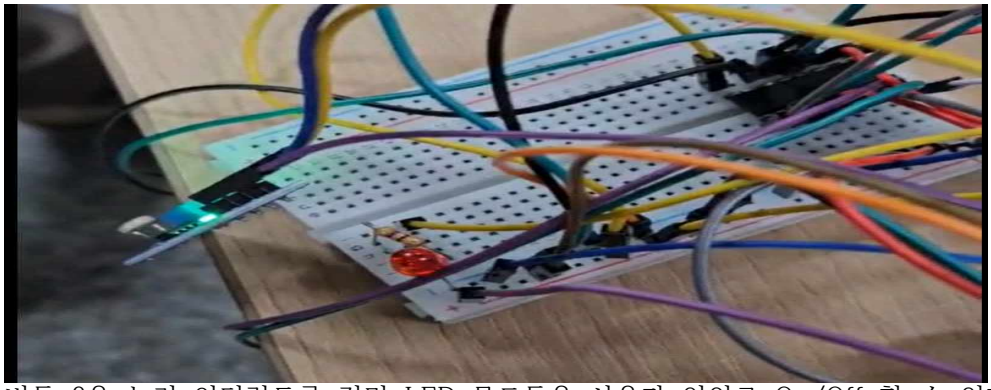
mood_value가 70 이하, 50 이상일 때 모터는 1개만 동작하며 LED 무드등은 적절한 밝기를 표시하고 있다.

Fig. 3 - System Mode (mood_value < 50)



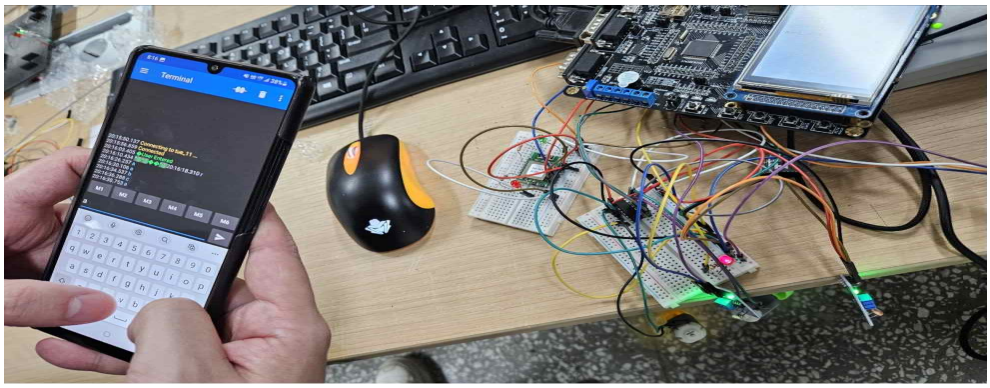
mood_value가 50미만 일 때 2개의 모터는 모두 꺼지며 LED 무드등은 가장 밝은 빛을 표시한다. Fig. 3은 테스트를 위해 조도 센서를 손으로 완전히 가려서 mood_value를 낮춘 상황이다.

Fig. 4 - Button Interrupt 3



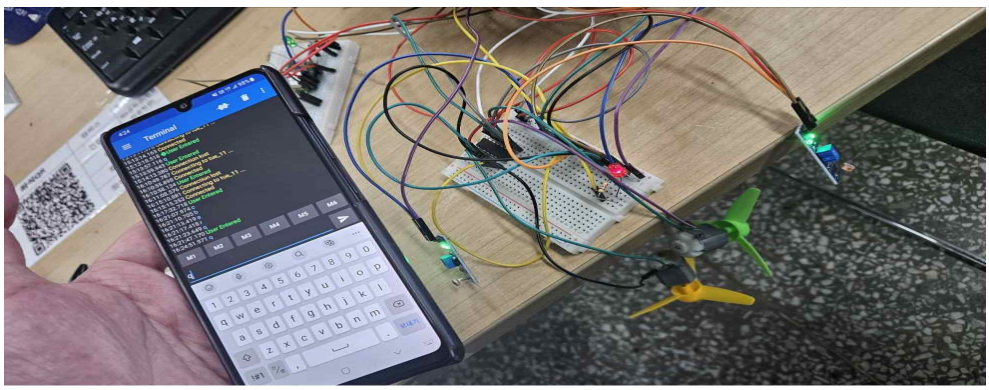
버튼 3을 눌러 인터럽트를 걸면 LED 무드등을 사용자 임의로 On/Off 할 수 있다. 추가적으로, 버튼 1은 모터를 On/Off 하며 버튼 4는 시스템 모드로 되돌림을 확인했다.

Fig. 5 - Bluetooth - user mode



휴대폰과 STM32 보드가 Bluetooth 연결을 한 상태이다. 초음파 센서에 의해 사람이 인식되어 화면에 “User Entered”라는 문자열이 출력된 모습을 확인할 수 있다. 그 뒤 ‘a’, ‘b’, ‘c’, ‘r’ 등의 문자를 입력하면, 사용자 모드 인터럽트가 수행됨을 확인했다.

Fig. 6 - Bluetooth - quit & re-entry



휴대폰에서 ‘q’를 입력하면, UART 통신을 통해 모터, LED, LCD 디스플레이 등의 초음파 센서를 제외한 모든 센서들이 리셋 되는 모습을 확인할 수 있다. 초음파 센서가 다시 사람을 인식하면 Fig. 4 상황을 반복할 수 있음을 확인했다.

IV 실험 결론 및 제언

• 실험 결론

본 프로젝트를 통해 스마트 홈 IoT 시스템 설계와 구현의 기초를 성공적으로 수행하였다.
주요 결과는 다음과 같다.

1. 센서와 장치 간의 통합 제어
- 초음파 센서를 이용해 사용자 출입을 정확히 감지하고, 이를 기반으로 시스템 초기화를 수행하였다.
- 조도 센서를 통해 실내외 밝기를 측정하고, LED 밝기와 모터 동작을 환경에 따라 동적으로 제어하였다.

2. 효율적인 데이터 처리
- DMA(Direct Memory Access)를 사용해 ADC 데이터를 CPU 개입 없이 처리함으로써 성능을 최적화하였다.
- TIM 및 PWM(Pulse Width Modulation) 신호를 활용해 LED 밝기를 효율적으로 조절하였다.

3. 사용자 친화적인 인터페이스
- UART 블루투스 통신 및 버튼 인터럽트를 사용해 사용자 모드에서 장치를 직접 제어할 수 있도록 구현하였다.
- ‘a’, ‘b’, ‘c’ 등의 간단한 명령어를 통해 사용자 모드에서 모터를 제어하고, ‘r’ 키를 통해 시스템 제어 모드로 전환 가능하게 하여 사용 편의성을 높였음.
- ‘q’ 키를 통해 스마트 홈 IoT 시스템을 중단할 수 있고, 이후 초음파 센서가 동작해 사용자가 다시 집에 들어오는지를 실시간으로 확인할 수 있었다.

4. 확장 가능성 확보
- 설계 단계에서 Wi-Fi 및 음성 인식 기능을 추가할 수 있도록 시스템의 확장 가능성을 열어두었다.
- 추가적인 센서와 장치를 통합할 수 있는 구조로 설계하였으며, 향후 다양한 인터페이스를 연결할 기반을 마련하였다. 예를 들면 DMA의 채널을 늘려서 추가적인 센서들을 결합할 수 있다.

• 직면했던 문제점

1. 초음파 센서 간섭 문제
초음파 센서와 모터의 동작이 간섭하여 간헐적으로 거리 측정이 부정확한 결과를 보였었다.
이를 해결하기 위해 타이머 설정과 간격 조정을 통해 간섭을 최소화하는 방법을 적용하였다.

2. UART 데이터 처리 지연
블루투스를 통한 명령 처리 중 일부 데이터 전송이 지연되는 문제가 발생하였다.
USART 인터럽트 우선순위를 조정하고 데이터를 실시간으로 처리하도록 개선하였다.

3. 조도 센서 데이터 정확도
조도 센서의 값이 특정 환경에서 불규칙적으로 변화하는 현상이 발생하였다.
ADC의 샘플링 속도와 해상도를 조정하고 필터링 알고리즘을 추가하여 문제를 완화하였다.

4. 인터럽트 방식의 비효율성
조도 센서를 읽는 인터럽트 방식은 센서 핀마다 별도의 설정이 필요하고, 센서 간 번갈아 발생하는 인터럽트로 인해 지연시간 증가와 코드 복잡성을 초래했다.
DMA의 다중 채널 방식을 채택해 CPU 개입 없이 주변 장치와 메모리 간 데이터를 전송하도록 구성해 센서 안정성과 신뢰성을 보장했다.

• 제언 및 개선 방안

1. 시스템 안정성 강화
센서와 모터 간의 간섭 문제를 근본적으로 해결하기 위해 하드웨어 레벨의 노이즈 필터를 추가할 수 있다.

2. 인터페이스 개선
사용자 경험(UX)을 향상시키기 위해 LCD 화면에 메뉴와 설정 옵션을 추가하는 것을 고려할 수 있다.
추후 블루투스 외에도 Wi-Fi를 통합하여 원격으로 제어 및 모니터링 기능을 제공할 수 있을 것으로 기대한다.

3. 확장된 테스트 환경 구성
다양한 조명 환경(실내, 야외)과 사용자 행동 시나리오를 테스트하여 시스템의 성능과 신뢰성을 검증하는 것이 좋을 것 같음.

V 참고문헌

[1] STM32F101xx, STM32F102xx, and STM32F105xx STM32F103xx. "STM32F107xx advanced ARM-based 32-bit MCUs." Reference manual. STMicroelectronics (2010). (https://www.st.com/resource/en/reference_manual/cd00171190-stm32f101xx-stm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf)