텀 프로젝트 보고서

화분관리시스템

임베디드 시스템
060 분반

부산대학교 정보컴퓨터공학부

201524530 이승준

2019 년 12 월 24 일

요 약

본 문서는 임베디드 시스템 과목의 텀 프로젝트로 조도 센서, 온도센서, 토양수분측정센서, led, 서보 모터를 활용한 화분 관리 시스템 설계 내용을 담고 있다. 시스템의 목표, 기능, 작동 원리 및 작성한 코드에 대하여 설명한다.,

1. 프로젝트 소개

1.1 개발 목표

본 프로젝트에서는 임베디드 시스템 강의에서 배운 ARM 프로세서의 구조와 원리, uC/OS-II 의 구성과 동작원리를 파악하고, 코드를 작성하여 원하는 결과물을 만들어내는 것을 목표로 한다.

이번 텀 프로젝트에서는 조도 센서, 온도센서, 토양수분측정센서, led, 서보 모터를 활용한 화분 관리 시스템을 통해 사람의 손을 거치는 일 없이 자동으로 화분을 관리 할 수 있다.

2. 개요

2.1 개발 시스템

STM32VL-DISCOVERY

ARM의 Cortex-M3 CPU를 기반으로, 128KB Flash memory와 8KB RAM을 탑재한 개발 보드이다. PA[0,15], PB[0,15], PC[0,15], PD[0,2]의 GPIO 포트를 지원하며 3개의 USART 채널, 2개의 SPI 채널, I2C 채널과 15개의 ADC 채널 및 4채널씩 5개의 타이머 포트를 지원한다. STM32VL-DISCOVERY 보드는 ST-Link 디버거를 내장하여 단순한 USB 연결을 통한 프로그래밍을 지원한다.

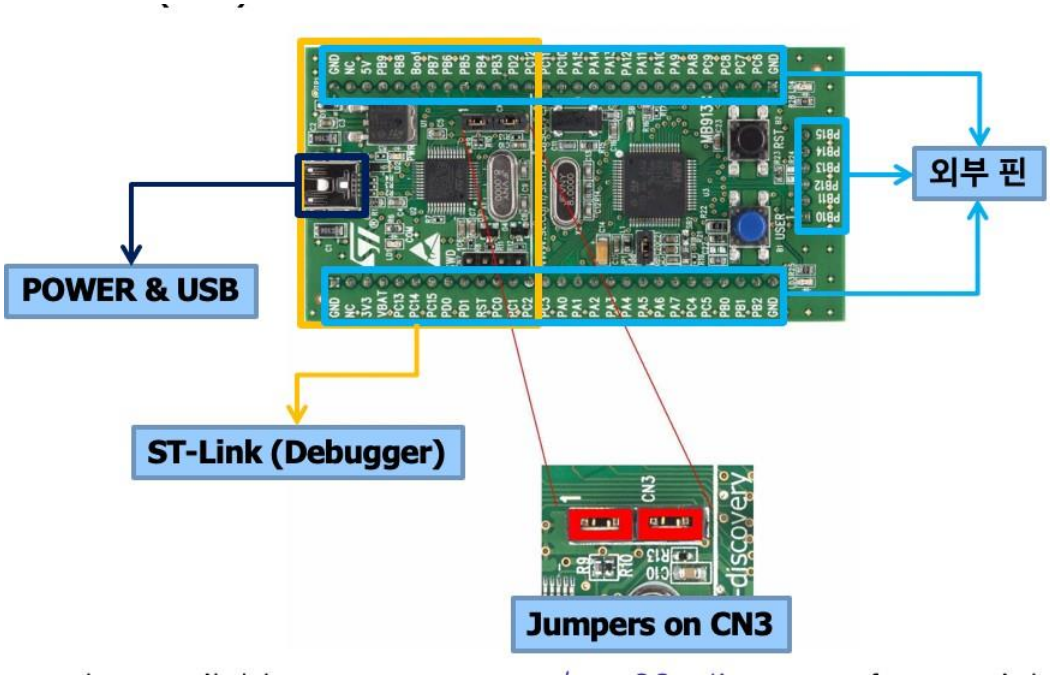


그림 1 STM32VL-DISCOVERY 구성도

2.2 사용하는 센서 및 기능

-입력 센서

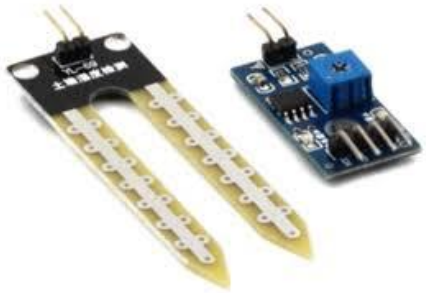
1) 조도 센서

빛의 양을 측정 한다. 본 프로젝트에서는 화분의 햇빛 량에 따라 led를 on/off 한다.

2) 온도 센서(LM35D)

온도를 측정 한다. 화분 주위의 온도를 측정하여 DC 쿨링 팬을 on/off 한다.

3) 토양 수분 측정 센서



센서 저항의 변화를 통해 수분을 측정. 화분 토양의 수분을 측정하여 서보 모터를 on/off 한다.

-출력 센서

1) LED

빛을 제공하는 센서. 조도 센서의 값에 따라 ON/OFF 된다.

2) DC 쿨링 팬

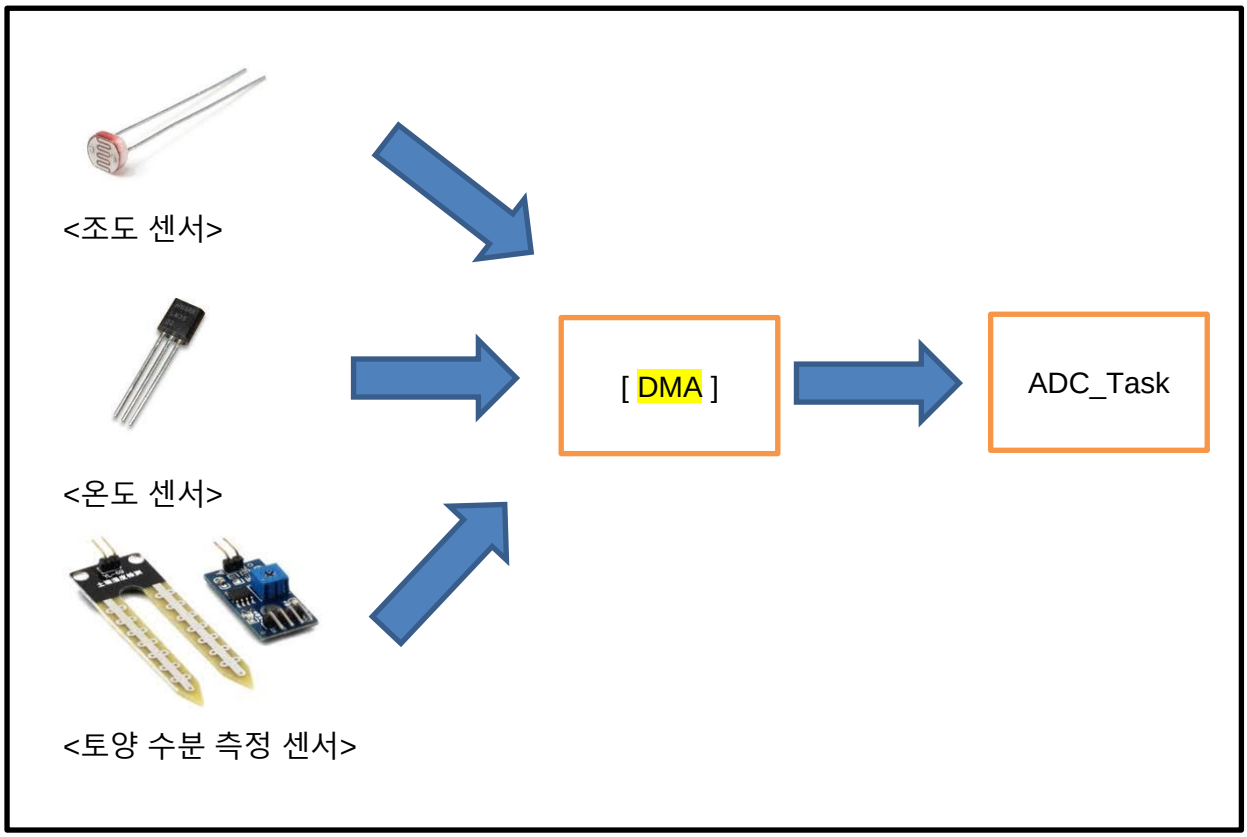
온도 센서의 값에 따라 ON/OFF 된다.

3) 서보모터(MG90S)

-90도 ~ 90도 까지 회전하는 모터. 토양 수분 측정 센서의 값에 따라

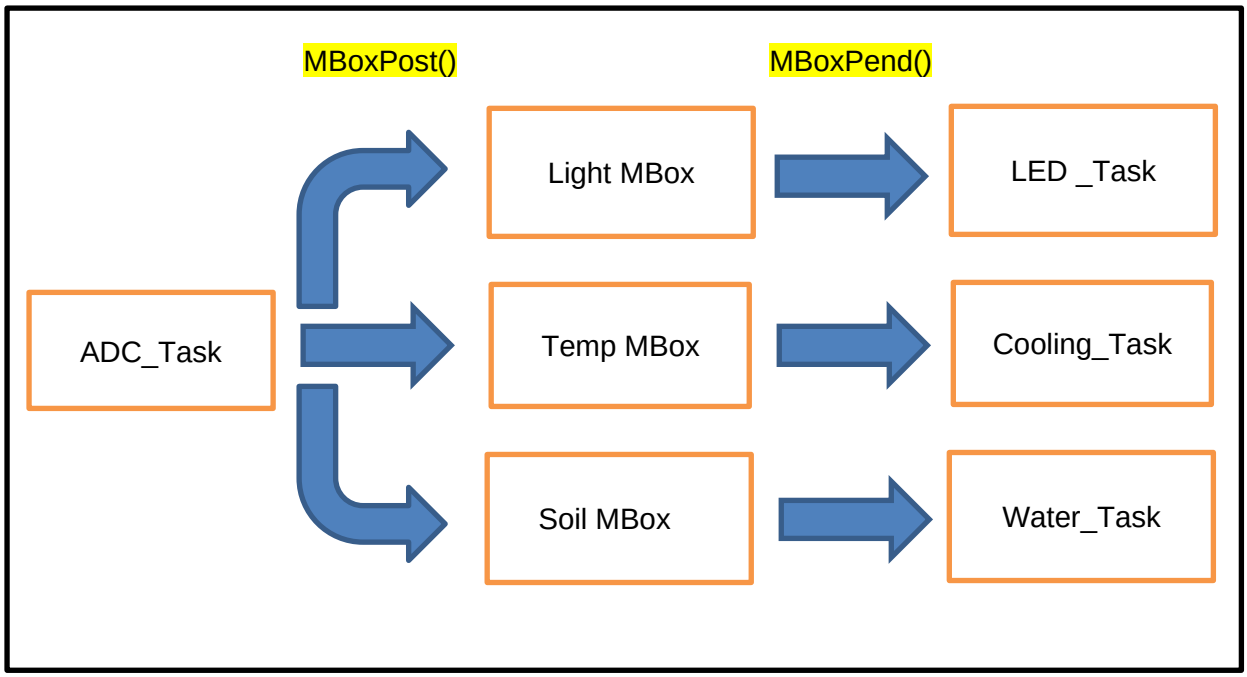
90도 또는 -90도로 전환하여 토양 수분 공급 장치의 물을 화분에 제공한다.

3. 작동 원리



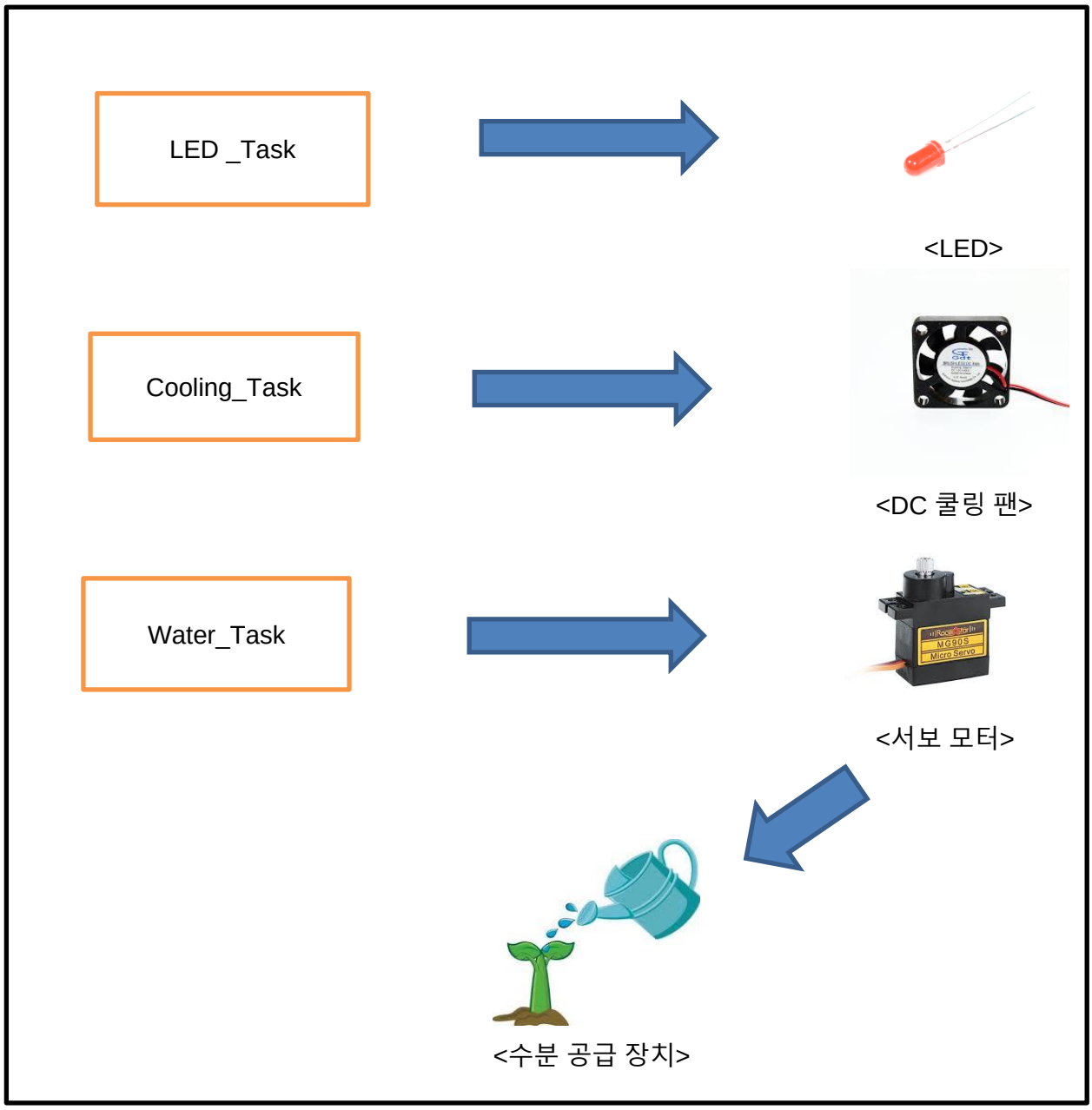
<그림 1. DMA를 통해 센서의 아날로그 값을 받는 입력 Task>

조도 센서, 온도 센서, 토양 수분 측정 센서로부터 얻은 아날로그 값을 ADC-Task에서 0.1초 간격으로 입력 받는다. 보드의 ADC converter는 2개인데 사용해야하는 ADC는 3개 이므로 DMA를 이용해 Multi ADC Channel을 구현 했다.



<그림 2. MailBox를 사용한 Task 간의 inter-networking >

아날로그 값을 입력받는 ADC_Task는 요구 되는 센서 값에 알맞게 RTOS에서 제공하는 MailBox 를 통해 MBoxPost() 함수로 값을 송신하고, 출력 Task 들은 MBoxPend() 함수를 통해 원하는 센서 값을 수신한다.



<그림 3. 출력 Task 내부 알고리즘에 따라 외부 센서 작동 >

출력 Task들은 MailBox 로부터 받은 값을 토대로 내부 알고리즘의 조건에 따라 외부 센서를 작동시킨다. 서보 모터의 경우 조건에 따라 수분 공급 장치와 연계되어 작동한다.

4. 구현 코드 설명(포트 설정 및 초기화 하는 부분은 생략)

1) Main()

```
int  main(void)
{
        CPU_INT08U os_err;
        /* Disable all ints until we are ready to accept them.  */
        BSP_IntDisAll();
        /* Initialize "uC/OS-II, The Real-Time Kernel".         */
        /* IDLE Task 와 Statistics Task 생성                     */
        OSInit();
        /* Create the start task.                               */
        /* OSTaskCreatExt()                                     */
        /* OSTaskCreate()와 다르게 Stack 을 검사할수 있는 기능을 가짐 */
        os_err = OSTaskCreateExt((void (*)(void *))App_TaskStart,
                        (void* )0,                      // Task 로 넘겨줄 인자
                        (OS_STK* )&App_TaskStartStk[APP_TASK_START_STK_SIZE -
1],
                        (INT8U           )APP_TASK_START_PRIO,
                         (INT16U          )APP_TASK_START_PRIO,
                        (OS_STK* )&App_TaskStartStk[0],
                         (INT32U          )APP_TASK_START_STK_SIZE,
                         (void* )0,
                        (INT16U          )(OS_TASK_OPT_STK_CLR |
OS_TASK_OPT_STK_CHK));
#if (OS_TASK_NAME_SIZE >= 11)
        OSTaskNameSet(APP_TASK_START_PRIO, (CPU_INT08U*)"Start Task", &os_err);
#endif
        OSStart();
        return(0);
}
```

main에서 uc/os api를 초기화하고 전체 task를 수행하는 함수 App_TaskStart() 를 OSTaskCreateExt() 함수를 통해 생성한다. Main 끝에는 OSStart 함수를 통해 task들을 running 한다.

2) APP_TaskStart()

```
static void  App_TaskStart(void *p_arg)
{
        ...
        App_EventCreate();
        RCC_Configure();
        GPIO_Configure();
        DMA_Configure();
        ADC_Configure();
        TIM_Init();
        os_err = OSTaskCreateExt((void (*)(void *)) ADC_Task, // Task 가 수행할
함수
                        (void*) 0, // Task 로 넘겨줄 인자
                        (OS_STK*) &ADC_TaskStartStk[APP_TASK_START_STK_SIZE -
1],
                         // Task 가 할당될 Stack 의 Top 을 가리키는 주소
                        (INT8U)5,// Task 의 우선 순위
                        (INT16U) 5,
                        // Task 를 지칭하는 유일한 식별자, Task 갯수의 극복을
위해서 사용할 예정, 현재는 우선 순위와 같게끔 설정
        os_err = OSTaskCreateExt((void (*)(void *)) LED_Task,
                        (void*) 0,
                        (OS_STK*) &LED_TaskStartStk[APP_TASK_START_STK_SIZE -
1],
                        (INT8U) 6,
                        (INT16U) 6,

        os_err = OSTaskCreateExt((void (*)(void *)) Cooling_Task,
                        (void*) 0,
                        (OS_STK*) &Cooling_TaskStartStk[APP_TASK_START_STK_SIZE
- 1],
                        (INT8U) 7,
                        (INT16U) 7,

        os_err = OSTaskCreateExt((void (*)(void *)) Water_Task,
                        (void*) 0,
                        (OS_STK*) &Water_TaskStartStk[APP_TASK_START_STK_SIZE -
1],
                        (INT8U) 8,
                        (INT16U) 8,

        while (DEF_TRUE) {
                OSTimeDlyHMSM(0, 0, 0, 100);
        }
}
```

포트에 클럭을 인가하는 RCC_Configure(), 포트 설정하는 GPIO_Configure, 아날로그 값을 받아내기 위한 ADC_Configure(), DMA를 설정하는 DMA_Configure() 함수를 호출한다. 사용할 모든 Task(ADC _Task ,LED Task, Cooling Task, Water_Task) 를 OSTaskCreateExt() 함수를 통해 실질적으로 Create 한다. 아날로그 값을 입력 받는 ADC_Task의 우선순위를 제일 높게 설정하고, 나머지 Task는 순서대로 우선순위를 다르게 설정하고 Stack도 개별적으로

할당 하였다.

3) App_EventCreate()

```
static void  App_EventCreate(void)
{
#if (OS_EVENT_NAME_SIZE > 12)
         CPU_INT08U os_err;
#endif
         /* Create MBOX for communication between Kbd and UserIF.*/
         /* Mail Box 생성                                        */
         /* 포인터 크기의 변수를 Task 나 Interrupt Service Routine   */
         /* 에서 다른 Task 전달할 때 사용함                        */
         LightMbox = OSMboxCreate((void*) 0);
         TempMbox = OSMboxCreate((void*) 0);
         SoilMbox = OSMboxCreate((void*) 0);
#if (OS_EVENT_NAME_SIZE > 12)
         OSEventNameSet(LightMbox, "LightMbox", &os_err);
         OSEventNameSet(TempMbox, "TempMbox", &os_err);
         OSEventNameSet(SoilMbox, "SoilMbox", &os_err);
#endif
}
```

Task 간의 inter-networking을 위해 RTOS에서 제공하는 MailBox 생성 함수이다. 빛의 양을 포함하는 LightMbox, 온도를 갖고 있는 TempMbox, 토양 수분 값을 갖고 있는 SoilBox를 선언했다.

4) 입력 Task - ADC_Task()

```
static void ADC_Task(void* parg){ /*  입력받은 ADC 를 다른 task 에 post 해주는 함수
*/
         while(DEF_TRUE){
           ADC_Value[0] = ADC_ValueTab[0]; // Light value
           ADC_Value[1] = ADC_ValueTab[1] * 500/4095; // Temperature value
           ADC_Value[2] = ADC_ValueTab[2]; // Soil moisture value
           OSMboxPost(LightMbox,(void*)(ADC_Value[0]));
           OSMboxPost(TempMbox,(void*)(ADC_Value[1]));
           OSMboxPost(SoilMbox,(void*)(ADC_Value[2]));
           OSTimeDlyHMSM(0,0,0,100);
         }
}
```

DMA를 통해 아날로그 값이 저장된 ADC_ValueTab 배열의 값은 가공 되지 않았으므로, 변수(빛, 온도, 토양수분)에 알맞게 변환하고 MailBox를 통해 MBoxPost() 함수로 출력 Task들에게 송신한다.

5) 출력 Task

- LED Task()
- Cooling Task()
- Water Task()

```
static void LED_Task(void* parg){
    while(DEF_TRUE){
            CPU_INT08U os_err;
            int ADC_value = (int) OSMboxPend(LightMbox,10,&os_err);
            if(ADC_value >30){  /*  빛이 밝은 경우  */
                 GPIO_ResetBits(GPIOC,GPIO_Pin_8);
            }
            else{
              GPIO_SetBits(GPIOC,GPIO_Pin_8);
            }
              OSTimeDlyHMSM(0,0,0,100);
    }
}

static void Cooling_Task(void* parg){
   while(DEF_TRUE){
           CPU_INT08U os_err;
           int ADC_value = (int) OSMboxPend(TempMbox,10,&os_err);
           if( ADC_value > 60){  /*  온도가 높은 경우 */
                Stop();
                delay();
          }
          else{
                Cooling();
                delay();
          }
             OSTimeDlyHMSM(0,0,0,200);
   }
}

static void Water_Task(void* parg){
   while(DEF_TRUE){
            CPU_INT08U os_err;
            int ADC_value = (int) OSMboxPend(SoilMbox,10,&os_err);
            if( ADC_value > 1000){  /* 수분 부족 */
                openDoor();
                delay();
          }
          else{
                closeDoor();
                delay();
          }
             OSTimeDlyHMSM(0,0,0,300);
   }
}
```

출력 Task(LED_Task, Cooling_Task, Water_Task) 는 요구되는 변수 값에 따라 MailBox로 부터 MBoxPend() 함수를 통해 값을 가져온다. Task 들은 얻은 값의 조건에 따라 센서의 동작이 바뀌는데, LED Task 에서는 빛이 밝은 경우 LED를 Off 하고 빛이 부족한 경우 LED를 On 한다. Cooling Task는 온도가 높은 경우 DC 쿨링팬을 On하고 온도가 낮은 경우 Off 한다.
Water Task는 토양에 수분이 부족한 경우 서보 모터를 동작하여 수분공급장치로부터 물을 제공하고, 수분이 충분한 경우 서보 모터를 원래 위치로 움직이게 다시 동작하여 수분의 공급을 끊는다. Task 들은 Pend 하여 얻은 값의 조건에 따라 센서의 동작이 변한다.

5. 결과

ADC_Task 를 조도 센서를 통한 LED On/Off, 토양수분측정센서를 통한 수분공급장치(서보 모터) 작동, 온도센서를 통한 온도 측정은 완성 하였으나, 온도에 따른 DC 쿨링 팬 작동은 DC 쿨링 팬이 없는데다가 시간이 부족함에 따라 이 부분을 코드로만 구현하였고 실제 테스팅은 하지 못했다.

동영상 : https://drive.google.com/file/d/1ozgcKzULs3Iz3h2W2XD2tdYH2glF7p-2/view?usp=drivesdk