Embedded Term Project Report

Use Sensor and Processing on Real Time Operating System(uC/OS)

201524582 정희석

201724628 KHAN OSAMA

2019.11.25 ~ 2019.12.23

---

1. 프로젝트 진행 목적

2019학년도 2학기 임베디드 시스템에서 배운 내용 중 RTOS를 바탕으로 실제 ARM CORTEX-M3 Processor 기반 보드(STM32DISCOVERY)를 이용하여 여러 센서와 모듈을 관리해 봄으로써 임베디드 시스템을 이해하는데 그 목적이 있다.

=> 저희 팀의 목적은 어떤 주제를 정하고 그 기능을 구현하는 것이 아니라, 가지고 있는 센서와 모듈들을 uC/OS(RTOS)상의 Task로 어떻게 구현하며 이를 관리해 봄으로써 임베디드 시스템을 이해하고 사용한 센서를 이용하여 어떤 방향으로 응용이 가능할 지를 고찰해 보는 것에 의의를 두었습니다.

2. 프로젝트에서 사용한 RTOS와 보드, 센서 소개.

2.1. uC/OS-II

개요) 이번 프로젝트에서 사용한 운영체제로 실시간 운영체제(Real Time Operating System의 한 종류이며 캐나다 출신 소프트웨어 개발자 Jean J. Labrosse가 uC/OS-I을 1991년 출시하였고 uC/OS-II는 1998년 출시된 ANSI C 언어로 작성된 운영체제.

특징)

1. 간단한 구조: 예제 코드 만으로 OS의 대부분의 기능을 확인하고 이해할 수 있다.

2. 작은 크기: 사용하지 않는 OS기능을 뺄 수 있고 OS에 필요한 모든 기능을 다 넣어도 30kbyte를 초과하지 않는다.

3. 실시간 커널 사용: 이벤트와 우선순위를 통하여 Task를 처리하여 Multi-Tasking을 지원하며, 외부 장치 확장성을 높인다.

4. 우선순위 스케줄링: 타 플랫폼 OS의 Round-Robin 방식과 다르게 Preemptive Priority Scheduling을 채택한 Real-Time Kernel 구조를 사용, Priority Table을 통해 최대 64개의 Task의 우선순위를 설정 가능 이를 통해 각 Task의 Multi-Tasking을 지원 및 외부 장치 확장성을 높인다.

5. 내부 task간 통신을 지원: Semaphore, Mutual Execution, Mailbox등 각 Task사이의 데이터 전달을 지원한다.

6. 정적 메모리 관리: OS가 동작하기 전 필요한 메모리 자원을 설정한 대로 할당하고 OS가 동작할 때는 메모리를 추가로 할당하지 않는다.

2.2. STM32DISCOVERY(STM32F100RBT6B)

SPEC) [i]

Microcontroller: STM32F100RBT6B (ARM Cortex-M3 Based)

Flash: 128 Kbyte, RAM: 8 Kbyte, 64-pin LQFP[1] On-board ST-Link with selection mode switch, stand-alone ST-link

Red LED 2EA (LED1,2), Green LED 1EA (LED3), Blue LED 1EA (LED4)

Push button 2EA (USER & RESET)

---

[1] LQFP: Low-profile Quad Flat Package: 부품의 핀이 4면으로 돌출된 형태의 집적회로 패키지

2.3. Obstacle Detection Sensor(장애물 감지 센서)(WAT-S006)

Feature)[ii]

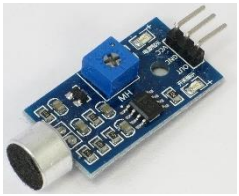
Detection Range: 0~20cm (Black)/0~40cm (White)(위의 흰색 가변저항으로 조절 가능)

Voltage Range: 3-6V(STM32보드의 3.3V, 5V 모두 사용가능)

Output: Detect->0, Not Detect->1 (Reverse)

Potentiometer: 202 Potentiometer => Carrier Frequency 조절 (up to 38Khz, related to 민감도), 103 Potentiometer => Detection Distance 조절

2.4. Sound Detection Sensor(소리 감지 센서)(LM393)



Feature)

Voltage Range: 4-6V(STM32보드의 5V 사용가능)

Output: Analog data (Detect 하면 값이 증가, 소리 크기에 따라 차이가 있음)

Potentiometer: 민감도 조절 가능(조절하면 Output 값이 증가, 본 프로젝트에서는 200을 감지 기준으로 잡음)

2.5. Pizio Buzzer(passive)

Feature)

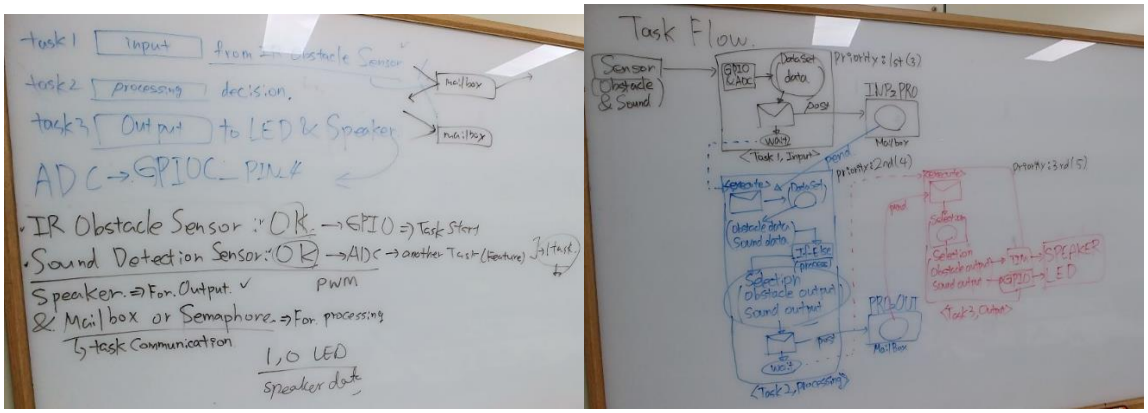
이번 프로젝트에서 사용하는 buzzer는 수동 buzzer로 주파수에 맞게 출력을 해줘야 소리가 출력되는 buzzer.

3. 역할분담

정희석: uC/OS 구조 분석, Task생성 및 구현, 보고서 작성

KHAN OSAMA: 센서 모듈 분석, 회로 구성, 보고서 수정

4. Inter-task Communication Diagram



Task를 3개로 나눠서 Input Task – Processing Task – Output Task로 나누었다.

Input에서는 외부 센서 들로부터 값을 읽는 과정을, Processing은 일정 기준에 따라 Output 여부를 결정하는 과정, Output은 외부 모듈에 값을 전달하는 과정을 담당한다. 각 Task는 Mailbox를 통해 값을 전달하고 전달받는다.

5. 구현 내용

5.1. uC/OS 설정 및 Task 생성

```c
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 Task와 Statistics Task 생성                       */
    OSInit();

    /* Create the start task.                                */
    /* OSTaskCreatExt()                                      */
    /* OSTaskCreate()와 다르게 Stack을 검사할수 있는 기능을 가짐 */
    //1st. input task
    os_err = OSTaskCreateExt((void (*)(void *))App_TaskStart, // Task가 수행할 함수
                    (void* )0,                  // Task로 넘겨줄 인자
                    (OS_STK* )&App_TaskStartStk[APP_TASK_START_STK_SIZE - 1],    // Task가 할당될 Stack의 Top을 가리키는 주소
                    (INT8U      )APP_TASK_START_PRIO,// Task의 우선 순위(3으로 최우선순위 -> 이유 : 제일 처음으로 실행)
                    (INT16U     )APP_TASK_START_PRIO,// Task를 지칭하는 유일한 식별자, Task 갯수의 극복을 위해서 사용할 예정, 현재는 우선 순위와 같게끔 설정
                    (OS_STK* )&App_TaskStartStk[0],     // Task가 할당될 Stack의 마지막을 가리키는 주소, Stack 검사용으로 사용
                    (INT32U     )APP_TASK_START_STK_SIZE,// Task Stack의 크기를 의미
                    (void* )0,      // Task Control Block 활용시 사용
                    (INT16U     )(OS_TASK_OPT_STK_CLR | OS_TASK_OPT_STK_CHK));// Task 생성 옵션 - 초기화 시 Stack을 0으로 채울 것인지, 부동 소수점 연산 장치 사용할 것인지 등 설정
    //Add task2
    //2nd. processing task
    os_err = OSTaskCreateExt((void (*)(void *))App_Process_TaskStart, // Task가 수행할 함수
                    (void* )0,                  // Task로 넘겨줄 인자
                    (OS_STK* )&App_Process_TaskStartStk[APP_TASK_START_STK_SIZE - 1],    // Task가 할당될 Stack의 Top을 가리키는 주소
                    (INT8U      )APP_PRO_TASK_START_PRIO,// Task의 우선 순위(4로 차선순위 -> 이유 : 두번째로 실행)
                    (INT16U     )APP_PRO_TASK_START_PRIO,// Task를 지칭하는 유일한 식별자, Task 갯수의 극복을 위해서 사용할 예정, 현재는 우선 순위와 같게끔 설정
                    (OS_STK* )&App_Process_TaskStartStk[0],     // Task가 할당될 Stack의 마지막을 가리키는 주소, Stack 검사용으로 사용
                    (INT32U     )APP_PRO_TASK_START_STK_SIZE,// Task Stack의 크기를 의미
                    (void* )0,      // Task Control Block 활용시 사용
                    (INT16U     )(OS_TASK_OPT_STK_CLR | OS_TASK_OPT_STK_CHK));// Task 생성 옵션 - 초기화 시 Stack을 0으로 채울 것인지, 부동 소수점 연산 장치 사용할 것인지 등 설정

    //Add task3
    ///3rd. output task
    os_err = OSTaskCreateExt((void (*)(void *))App_Output_TaskStart, // Task가 수행할 함수
                    (void* )0,                  // Task로 넘겨줄 인자
                    (OS_STK* )&App_Output_TaskStartStk[APP_OUT_TASK_START_STK_SIZE - 1],    // Task가 할당될 Stack의 Top을 가리키는 주소
                    (INT8U      )APP_OUT_TASK_START_PRIO,// Task의 우선 순위(5로 Task중에서는 최하순위 -> 이유 : 3번째로 실행)
                    (INT16U     )APP_OUT_TASK_START_PRIO,// Task를 지칭하는 유일한 식별자, Task 갯수의 극복을 위해서 사용할 예정, 현재는 우선 순위와 같게끔 설정
                    (OS_STK* )&App_Output_TaskStartStk[0],     // Task가 할당될 Stack의 마지막을 가리키는 주소, Stack 검사용으로 사용
                    (INT32U     )APP_OUT_TASK_START_STK_SIZE,// Task Stack의 크기를 의미
                    (void* )0,      // Task Control Block 활용시 사용
                    (INT16U     )(OS_TASK_OPT_STK_CLR | OS_TASK_OPT_STK_CHK));// Task 생성 옵션 - 초기화 시 Stack을 0으로 채울 것인지, 부동 소수점 연산 장치 사용할 것인지 등 설정

#if (OS_TASK_NAME_SIZE >= 11)
    OSTaskNameSet(APP_TASK_START_PRIO, (CPU_INT08U*)"Start Task", &os_err);
#endif

    OSStart();                              /* Start multitasking (i.e. give control to uC/OS-II).  */

    return(0);
}
```

-> OSInit()을 통해 OS 기본 설정(메모리 할당, 가장 기본이 되는 Task생성)

-> OSTaskCreateExt를 이용해 Task를 3개 만든다. 각각 Stack를 가지고 있으며 Size는 128byte이다. 각 Tasks, TaskStart(Input Task)는 3의 우선순위를 Process_TaskStart(Processing Task)은 4의 우선순위를, Output_TaskStart는 5의 우선순위를 갖는다. 이유는 Input -> Processing-> Output의 순서를 유지하기 위해 설정하였다.

5.2. Task 구현

1) Input Task

```c
static void  App_TaskStart(void *p_arg)
{
    DataSet input;
    (void)p_arg;

    BSP_Init();                               /* Initialize BSP functions.            */
    OS_CPU_SysTickInit();                       /* Initialize the SysTick.              */

#if (OS_TASK_STAT_EN > 0)
    OSStatInit();                             /* Determine CPU capacity.              */
#endif

#if ((APP_PROBE_COM_EN == DEF_ENABLED) || ₩
    (APP_OS_PROBE_EN == DEF_ENABLED))
    App_InitProbe();
#endif
   Init_For_Obst_Sensor();
    /* Create application events.           */
    /* Task간 통신을 위한 MailBox 생성        */
    App_EventCreate();

    /* Create application tasks.            */
    /* LCD 갱신 Task, 키보드 입력 Task 생성    */
    App_TaskCreate();

    /* Task body, always written as an infinite loop.    */
    while (DEF_TRUE) {
        //
        input.obstacleData = GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_2);
        input.soundData = ADC_GetConversionValue(ADC1);
        OSMboxPost(App_INP_PRO_Mbox, &input); //받은 값을 process와의 mail box에 전달
        OSTimeDlyHMSM(0, 0, 0,20);//20ms 대기
    }
}
```

-> 이 Task는 무한 loop를 돌며 GPIOD_Pin2를 통해서 장애물 센서 값을, ADC1을 통해서 소리 센서 값을 받고 MailBox를 이용하여 다음 Task에 데이터를 넘겨준다. 한 작업마다 20ms를 대기한다.

2) Processing Task

```
static void  App_Process_TaskStart(void *p_arg)
{
      DataSet *input;
      Selection output;
      CPU_INT08U err;
       (void)p_arg;

       BSP_Init();                                  /* Initialize BSP functions.                            */
       OS_CPU_SysTickInit();                           /* Initialize the SysTick.                              */

#if (OS_TASK_STAT_EN > 0)
       OSStatInit();                                /* Determine CPU capacity.                              */
#endif

#if ((APP_PROBE_COM_EN == DEF_ENABLED) || ₩
       (APP_OS_PROBE_EN == DEF_ENABLED))
       App_InitProbe();
#endif
       /* Task body, always written as an infinite loop.       */

       while (DEF_TRUE) {
            input = (DataSet*)OSMboxPend(App_INP_PRO_Mbox, OS_TICKS_PER_SEC / 10, &err);//input과의 mailbox에서부터 데이터를 받음
            if(!(input->obstacleData)){//데이터에 따라 output여부 설정
               output.obstacleOutput = DEF_TRUE;
            }
            else{
               output.obstacleOutput = DEF_FALSE;
            }
            if((input->soundData>200)&&(input->soundData<1000)){//센서 데이터가 200에서 1000사이인 경우 output설정
               output.soundOutput = DEF_TRUE;
            }
            else{
               output.soundOutput = DEF_FALSE;
            }
            printf("%d₩n%d₩n%d,%d₩n", input->obstacleData, input->soundData, output.obstacleOutput, output.soundOutput);
            //값 test code
            OSMboxPost(App_PRO_OUT_Mbox, &output);//output명령을 output과의 mail box에 전달
            OSTimeDlyHMSM(0, 0, 0,20);//20ms 대기
       }
}
```

-> Mailbox로부터 input task에서 보내준 값을 읽어서 인식 여부를 확인하여Output에게 출력 여부를 Mailbox를 통해 전달, 20ms 대기한다. Printf를 통해 제대로 값이 생성되었는지 확인.

3) Output Task

```
// OUTPUT을 위한 Task
static void  App_Output_TaskStart(void *p_arg)
{
       CPU_INT08U    err;
       Selection* selection;
       BSP_Init();                                  /* Initialize BSP functions.                            */
       OS_CPU_SysTickInit();                           /* Initialize the SysTick.                              */
       TIMER_Configure();

#if (OS_TASK_STAT_EN > 0)
       OSStatInit();                                /* Determine CPU capacity.                              */
#endif

#if ((APP_PROBE_COM_EN == DEF_ENABLED) || ₩
       (APP_OS_PROBE_EN == DEF_ENABLED))
       App_InitProbe();
#endif

       /* Task body, always written as an infinite loop.       */
       while (DEF_TRUE) {
        //to do implement
          temp = 0;
          selection = (Selection*)OSMboxPend(App_PRO_OUT_Mbox, OS_TICKS_PER_SEC / 10, &err );
          if(selection->obstacleOutput){
            while(selection->obstacleOutput){//Check!
              TIM_Cmd(TIM4, ENABLE);//piezo 동작 //틱틱소리 밖에 안남
            }
          }
          else{
            TIM_Cmd(TIM4, DISABLE); //piezo 정지
          }
          if(selection->soundOutput){
             BSP_LED_On(0);            //LED 동작
          }
          else{
             BSP_LED_Off(0); // LED 정지
          }
          OSTimeDlyHMSM(0, 0, 0, 20);//20ms 대기
       }
}
```

-> Mailbox로부터 동작 여부(1: 동작, 0: 미 동작)를 받아서 piezo와 LED를 동작시킨다. 그리고 각 Task간 순서 유지를 위해 한번 출력 후 20ms를 대기한다.

5.3 Mailbox 생성

```
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 TaskU Interrupt Service Routine  */
    /* 에서 다른 Task 전달할 때 사용함                      */
    App_UserIFMbox = OSMboxCreate((void*)0);
   App_INP_PRO_Mbox = OSMboxCreate((void*)0);//input과 process와의 mailbox
   App_PRO_OUT_Mbox = OSMboxCreate((void*)0);//process와 output과의 mailbox
#if (OS_EVENT_NAME_SIZE > 12)
    OSEventNameSet(App_UserIFMbox, "User IF Mbox", &os_err);
   OSEventNameSet(App_INP_PRO_Mbox, "Bt ItoP Mbox", &os_err);
   OSEventNameSet(App_PRO_OUT_Mbox, "Bt PtoO Mbox", &os_err);
#endif
}
```

-> input과 processing사이의 mailbox와 process와 output사이의 mailbox를 사용한다.

6. 구현 결과

<디버깅 과정>

```
1
172
0 0
0
177
1 0
```

1 => 장애물 센서 인식 여부(1:X, 0:O),

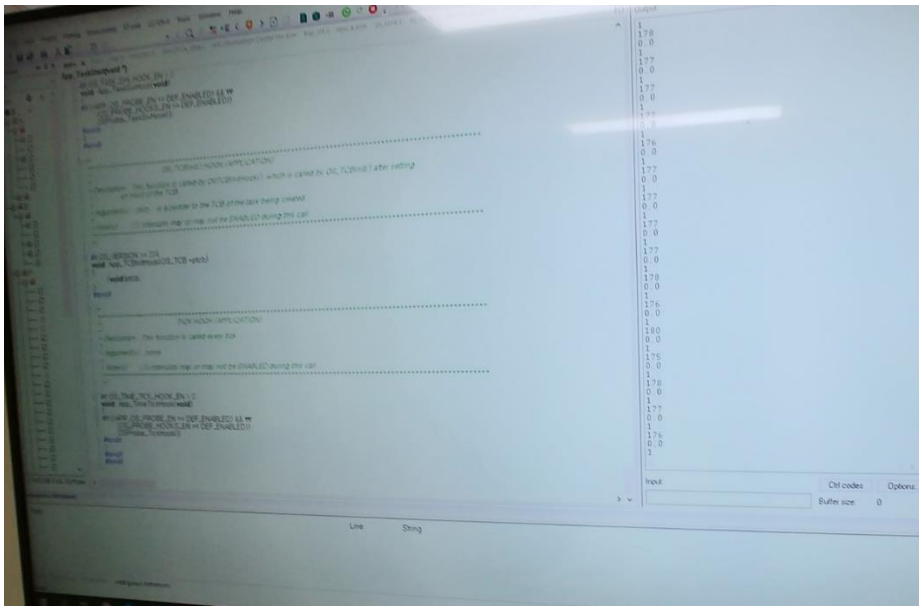
172 => 소리 센서 값(인식 기준: 200)

0 => 스피커 동작신호(0:X,1:O) | 0 => LED 동작신호(0:X,1:O)

0 => 장애물 센서 인식 여부(1:X, 0:O)

177 => 소리 센서 값(인식 기준: 200)

1 => 스피커 동작신호 | 0 => LED 동작신호



<동작 영상>

https://youtu.be/crQtFjUK8fc

7. 향후 응용 방안

-> 장애물 센서: 자동차에서 초근거리에서 특히 주차나 차선 유지 부분에서 차선인식에 주로 사용할 수 있을 것 같고 초음파 센서로 물체와의 거리를 측정하고 이를 보조하는 역할로써 사용 가능하다고 예상한다.

-> 소리 감지 센서: 자동차 실내에 설치하여 시동 꺼진 상태에서 문이 잠기고 차 내부에 동물이나 어린 아기들이 방치되어 있을 때 소리를 인식하여 경적을 울리도록 하여 사용 가능하다고 예상한다.

8. 한계 점

이 프로젝트를 통해서 OS의 동작과 외부 모듈을 사용하는 법을 알 수 있었다. 하지만 임베디드의 중요점은 동작(계산)의 최적화, 시간의 최적화가 중요한 점인데 이 프로젝트에서는 각 Task뒤에 20ms라는 정적인 시간을 집어넣어서 일정한 시간마다 동작하도록 하여서 속도가 느리게 동작하여 최적화가 되지 않은 점, 그리고 Piezo 스피커를 동작 시키기 위해서는 특정 주파수의 신호를 보드의 clock를 이용하여 만들어야 하는데 신호를 만드는 것에 대한 이해도가 낮아서 제대로 된 소리가 출력할 수 없었던 것이 너무 아쉬운 점이었다.

9. 느낀 점

정희석: 이번 프로젝트를 통하여 STM32보드를 만져보고 uC/OS-II라는 RTOS를 이용하여 3학년 1학기에 배웠던 운영체제의 동작과 이번 학기 배웠던 임베디드 시스템, 그 중 Real-Time Operating System에 대해 알아볼 수 있었다. 현재 같이하고 있는 임베디드 시스템 설계 및 실험에서의 RTOS 없이 진행하는 프로젝트와는 달라서 새로운 경험이 되었다. 이런 기회를 주신 교수님과 조교님께 감사드리며 프로젝트 동안 조원(Mr. Khan)이 잘 도와줘서 좋았다. 다음에 다른 프로젝트를 하게 되면 이번 보다 더 많이 공부해서 보다 더 완벽한 결과물을 낼 수 있으면 좋겠다.

KHAN OSAMA: Through this project, we learn how to control the input and output. The most important thing was, we understood how to use the real time operating system. We came to know about the task creation, deletion and modification. We managed to use different sensors. I would like to thank my teammate, Mr. 희석 for his contribution in this project. Without his contribution, it was very difficult for me to finish the project. I would also like to thank our Professor for giving us such an opportunity to learn and implement the knowledge we learn during the semester.

참고자료

[i] https://www.st.com/content/ccc/resource/technical/document/user_manual/f3/16/fb/63/d6/3d/45/aa/CD00267113.pdf/files/CD00267113.pdf/jcr:content/translations/en.CD00267113.pdf

[ii] https://www.alibaba.com/product-detail/OEM-ODM-Vehicle-intelligent-obstacle-detection_60045432618.html