

REPORT



임베디드시스템 과제2

수강과목: 임베디드 시스템
담당교수: 백윤주 교수님
학 과: 전기컴퓨터공학부 정보컴퓨터공학전공
이 름: 박윤형
학 번: 201624481
제출일자: 2021. 11. 21

1. Task정의

```
98 static void AppTask_LED1(void *p_arg);
99 static void AppTask_LED2(void *p_arg);
100 static void AppTask_LED3(void *p_arg);
101 static void AppTask_USART(void *p_arg);
102
```

Serial통신을 하는 task를 AppTask_USART로 정의하고 LED1, LED2, LED3를 조작하는 task를 각각 AppTask_LED1, AppTask_LED2, AppTask_LED3로 정의해 놓았다.

2. AppTask_LED1, AppTask_LED2, AppTask_LED3에 대한 설명

```
70 typedef enum{
71     LED_ON,
72     LED_OFF,
73     LED_BLINK
74 }led_status;
75
76 typedef enum{
77     LED1,
78     LED2,
79     LED3
80 }led_name;
81
82 typedef struct{
83     led_name name;
84     led_status status;
85     unsigned short blink_time;
86 }led_struct;
87
131 led_struct led_array[3] = {
132     {LED1, LED_OFF, 1u},
133     {LED2, LED_OFF, 1u},
134     {LED3, LED_OFF, 1u}
135 };
```

먼저 LED의 동작상태와 LED의 이름을 열거형으로 정의하고 Led의 이름, 상태, blink시간을 관리하기 쉽게 led_struct라는 구조체를 만들어 led_array라는 배열에 저장했다. 각각의 Led의 기본값은 꺼진 상태이다. led_array는 전역변수로 정의되어있다.

```
331 static void AppTask_LED1(void *p_arg)
332 {
333     OS_ERR err;
334
335     while (DEF_TRUE) { /* Task body, always written as an infinite
336
337         switch(led_array[LED1].status){
338             case LED_ON:
339                 BSP_LED_On(1);
340                 OSTimeDlyHMSM(0u, 0u, 0u, 500u, OS_OPT_TIME_HMSM_STRCT, &err);
341                 break;
342             case LED_OFF:
343                 BSP_LED_Off(1);
344                 OSTimeDlyHMSM(0u, 0u, 0u, 500u, OS_OPT_TIME_HMSM_STRCT, &err);
345                 break;
346             case LED_BLINK:
347                 BSP_LED_Toggle(1);
348                 OSTimeDlyHMSM(0u, 0u, led_array[LED1].blink_time, 0u, OS_OPT_TIME_HMSM_STRCT, &err);
349                 break;
350         }
351     }
352 }
353 }
```

LED를 제어하는 task의 전부 위와 동일하다. 0.5초를 주기로 led_array에 저장된 해당 LED의 동작 상태를 검사하며 동작상태가 LED_ON이면 LED를 켜고, LED_OFF면 LED를 끄고, LED_BLINK면 LED의 동작상태를 반전시키고 blink시간만큼 유지한다.

3. AppTask_USART에 대한 설명

```

234 static void AppTask_USART(void *p_arg)
235 {
236     uint16_t i;
237     char command[15] = "";
238     char temp[15] = "";
239     led_name led;
240     led_status status;
241     unsigned short blink_time;
242     int i = 0, k = 0;
243     int command_is_right;
244
245     OS_ERR err;
246
247     while (DEF_TRUE) {
248         i = 0; k = 0;
249         command_is_right = 0;
250
251         CPU_SR_ALLOC();
252
253         while(USART_ReceiveData(Nucleo_COM1) != '\0'){
254             while(USART_GetFlagStatus(Nucleo_COM1, USART_FLAG_RXNE) == RESET){
255                 OSTimeDlyHMSM(0u, 0u, 0u, 10u, OS_OPT_TIME_HMSM_STRICT, &err);
256             }
257
258             c = USART_ReceiveData(Nucleo_COM1);
259             USART_SendData(Nucleo_COM1, c);
260             command[i] = c;
261             i++;
262         }
263         if(i){ send_string("\n\r"); }
264
265         command[i-1] = '\0';

```

c는 serial 통신에서 데이터를 받기위한 변수, command[15]는 serial통신에서 받은 데이터를 문자열로 만들기 위한 변수, temp[15]는 command를 parsing할 때 쓰이는 변수이다. Command를 parsing한 후 led이름, led의 동작상태, blink시간이 각각 led, status, blink_time에 저장될 예정이며, command가 올바르게 입력되었는지 확인하기 위해 command_is_right라는 변수가 사용된다.

247줄의 while문은 serial통신에서 " ` " 가 입력되면 command를 parsing하고 입력된 command가 올바르면 전역변수로 정의된 led_array의 값을 바꾼다.

253줄의 while문이 command를 입력 받는 과정이며 이때 현재의 task의 priority가 led의 task보다 높게 되어 있기 때문에 현재의 task가 cpu를 독점하지 않게 하기위해 255줄에 0.01초씩 OS Time delay를 주도록 했다.

251줄의 코드는 task에서 critical section을 사용하기 위해 task의 상태를 저장하는 공간을 마련한다.

```

267     for(int j = 0; j < strlen(command); j++){
268         temp[k] = command[j];
269         temp[k+1] = '\0';
270         k++;
271
272         if(!strcmp(temp, "led1")){
273             led = LED1;
274             temp[0] = '\0';
275             k = 0;
276         }else if(!strcmp(temp, "led2")){
277             led = LED2;
278             temp[0] = '\0';
279             k = 0;
280         }else if(!strcmp(temp, "led3")){
281             led = LED3;
282             temp[0] = '\0';
283             k = 0;
284         }else if(!strcmp(temp, "on")){
285             status = LED_ON;
286             temp[0] = '\0';
287             k = 0;
288             command_is_right = 1;
289             break;
290         }else if(!strcmp(temp, "off")){
291             status = LED_OFF;
292             temp[0] = '\0';
293             k = 0;
294             command_is_right = 1;
295             break;
296         }else if(!strcmp(temp, "blink")){
297             status = LED_BLINK;
298             blink_time = command[j+1] - '0';
299             temp[0] = '\0';
300             k = 0;
301             command_is_right = 1;
302             break;
303         }else if(!strcmp(temp, "reset")){
304             led_array[LED1].status = LED_OFF;
305             led_array[LED2].status = LED_OFF;
306             led_array[LED3].status = LED_OFF;
307             status = LED_OFF;
308             temp[0] = '\0';
309             k = 0;
310             break;
311     }

```

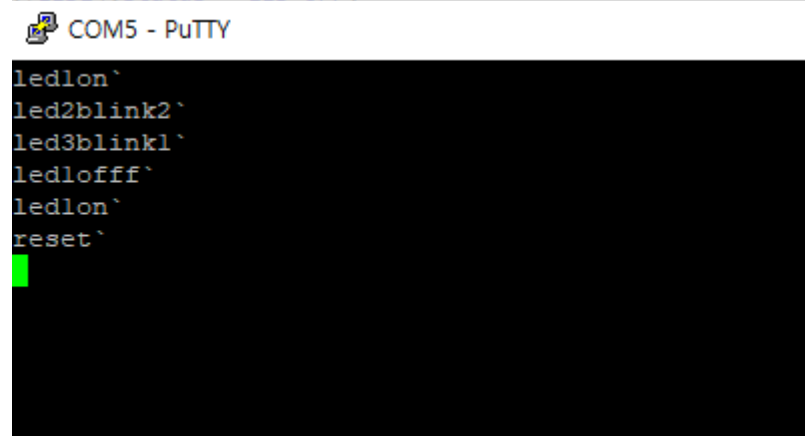
위의 코드는 입력된 command를 parsing하는 부분이다. Reset을 제외한 모든 command의 정보는 지역변수에 저장된다. Command가 reset일 경우에는 전역변수인 led_array를 강제로 변경시킨다. 지역변수에 저장된 command의 정보로는 LED의 이름, LED의 동작상태, blink 시간이 있으며 이 정보들은 이후에 command가 올바르다고 판단되면 critical section으로 보호된 상태에서 전역변수 led_array에 저장된다.

```

315     //critical section
316     if(command_is_right){
317         OS_CRITICAL_ENTER();
318         led_array[led].status = status;
319         led_array[led].blink_time = blink_time;
320         OS_CRITICAL_EXIT();
321     }
322
323     command[0] = '\0';
324
325     OSTimeDlyHMSM(0u, 0u, 0u, 50u, OS_OPT_TIME_HMSM_STRICT, &err);
326
327 }
328 BSP_LED_On(3);
329 }

```

4. 결과



```
COM5 - PuTTY
ledlon`
led2blink2`
led3blink1`
ledloff`
ledlon`
reset`
█
```

-putty에서 키보드 입력 시 보드에 전달되고 전달된 데이터가 다시 화면에 출력된다.

-동작영상에서 확인할 수 있듯이 command로 각 led의 제어가 가능하다. 다만 parsing 방법이 정교하지 않기 때문에 blink타임이 한자리수의 정수로 한정된다

.