STM32 블루투스 알람 시계

블루투스 알람시계 코드

실습 목표

실제 제품에 사용되는 수준의 코드를 분석하고 수정하는 실습

- 신입사원으로 연구소에 입사했더니 사수인 김대리가 1주일짜리 미션을 다음과 같이 던져주었다.
- 김대리왈 “UART 버전 알람시계 코드를 수정하여 LCD와 버튼을 사용한 블루투스 알람시계로 일주일안에 작성하시오.”

- 첨부된 코드(stm32_alarm_clock.zip)는 Select(=Set), left, right, up, down 버튼 과 LCD 대신 UART를 사용하여 작성된 알람시계를 작성한 코드
- 이 코드를 “UART 버전 알람시계”라 명명, 이 코드를 실제 버튼과 LCD를 사용하는 코드로 수정

UART 버전

버튼 대신 키보드 입력

알아야 할 키보드 정보

- 블루투스 알람 시계 프로젝트는 Select(=Set), left, right, up, down 버튼의 5개 버튼을 사용
- 키보드의 Enter를 Set 버튼으로 간주하고 키보드의 방향키를 left, right, up, down 버튼으로 간주
- 키보드 입력의 키 값과 버튼 입력 시간(길게 3초, 더블 클릭)을 체크할 수 있어야 함.

- 키보드 입력의 키 값은 key_value_main.c 파일로 설명
- 버튼 입력 시간(길게 3초, 더블 클릭) 은 key_interval_main.c 파일로 설명

UART 버전

버튼 대신 키보드 입력

key_value_main.c

```
121 }
122 /* Arrow Keys mapping
123  * UP = 'A' = 65
124  * DOWN = 'B' = 66
125  * RIGHT = 'C' = 67
126  * LEFT = 'D' = 68
127  * ENTER = 13
128  */
129 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
130 {
131   if (huart->Instance == USART3)
132   {
133       /* Transmit one byte with 100 ms timeout */
134 //    HAL_UART_Transmit(&huart3, &rcv_byte, 1, 100);
135
136     memset(uart_buf,0,sizeof(uart_buf));
137     sprintf(uart_buf,"key_value=%d\r\n",rcv_byte);
138     HAL_UART_Transmit_IT(&huart3,uart_buf,sizeof(uart_buf));
139
140     /* Receive one byte in interrupt mode */
141     HAL_UART_Receive_IT(&huart3, &rcv_byte, 1);
142   }
143 }
144 /* USER CODE END 0 */
145
146 /**
```

UART RX interrupt 콜백함수

키 입력 값을 저장하는 변수

Notepad++로 열었을 때 Line 번호

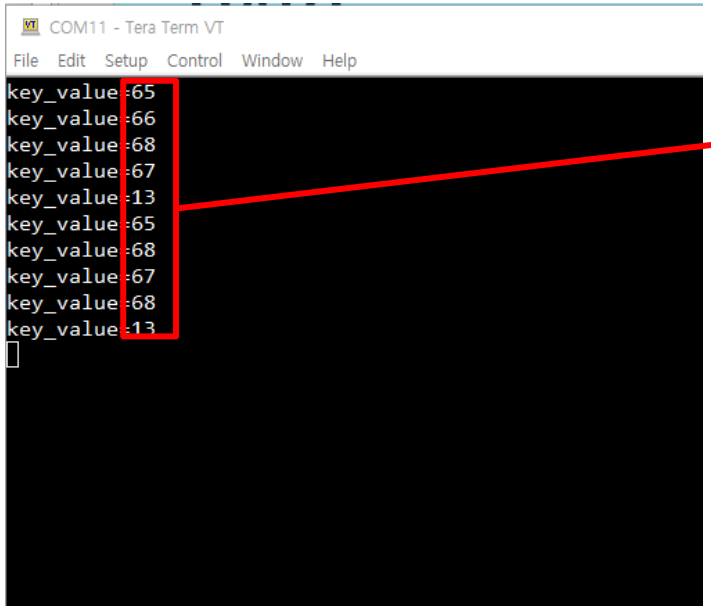
```
166   /* Configure the system clock */
167   SystemClock_Config();
168
169   /* USER CODE BEGIN SysInit */
170
171   /* USER CODE END SysInit */
172
173   /* Initialize all configured peripherals */
174   MX_GPIO_Init();
175   MX_RTC_Init();
176   MX_USART3_UART_Init();
177   MX_TIM2_Init();
178   /* USER CODE BEGIN 2 */
179   HAL_TIM_Base_Init(&htim2);
180   HAL_TIM_Base_Start_IT(&htim2);
181
182   HAL_UART_Receive_IT(&huart3,&rcv_byte,1);
183   /* USER CODE END 2 */
184
185   /* Infinite loop */
186   /* USER CODE BEGIN WHILE */
187   while (1)
188   {
189     /* USER CODE END WHILE */
190
191     /* USER CODE BEGIN 3 */
192   }
193   /* USER CODE END 3 */
194 }
195
```

UART RX interrupt를 시작하려면 main함수에 HAL_UART_Receive_IT를 넣어 줘야함.

UART 버전

버튼 대신 키보드 입력

key_value_main.c



방향키 UP을 누르면 65,
DOWN은 66, RIGHT는 67,
LEFT는 68, ENTER키는 13
인지 확인

- Teraterm을 열고 key 값을 확인

UART 버전

버튼 대신 키보드 입력

key_interval_main.c

```
126 /* Arrow Keys mapping
127  * UP = 'A' = 65
128  * DOWN = 'B' = 66
129  * RIGHT = 'C' = 67
130  * LEFT = 'D' = 68
131  * ENTER = 13
132  */
133
134 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
135 {
136   if (huart->Instance == USART3)
137   {
138       /* Transmit one byte with 100 ms timeout */
139 //    HAL_UART_Transmit(&huart3, &rcv_byte, 1, 100);
140
141     current_time = HAL_GetTick();
142     time_interval = current_time - last_time;
143     last_time = current_time;
144
145     /* Receive one byte in interrupt mode */
146     HAL_UART_Receive_IT(&huart3, &rcv_byte, 1);
147   }
148 }
149 /* USER CODE END 0 */
150
```

- HAL_GetTick()함수는 부팅후 1ms마다 1씩 증가하는 카운터 값
- 즉, 읽을 때마다 부팅후 지난 시간을 알 수 있음.
- 키를 누르는 순간의 시간을 current_time에 저장
- 지난 번 키를 눌렀을 때의 시간은 last_time에 저장
- 지난번 키와 현재 누른 키의 시간 간격은 time_interval에 저장

UART 버전

버튼 대신 키보드 입력

key_interval_main.c

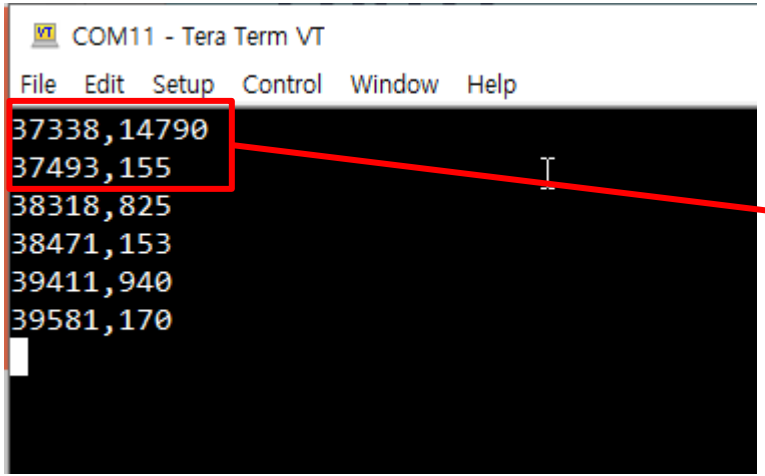
```
187   HAL_UART_Receive_IT(&huart3,&rcv_byte,1);
188   /* USER CODE END 2 */
189
190   /* Infinite loop */
191   /* USER CODE BEGIN WHILE */
192   while (1)
193   {
194     /* USER CODE END WHILE */
195     if(time_interval)
196     {
197       memset(uart_buf,0,sizeof(uart_buf));
198       sprintf(uart_buf,"%d,%d\r\n",current_time,time_interval);
199       HAL_UART_Transmit_IT(&huart3,uart_buf,sizeof(uart_buf));
200       time_interval = 0;
201
202
203     /* USER CODE BEGIN 3 */
204   }
205   /* USER CODE END 3 */
206 }
```

- 무한 while문에서 폴링방식으로 time_interval이 0이 아니면 UART로 current_time과 time_interval 출력

UART 버전

버튼 대신 키보드 입력

key_interval_main.c



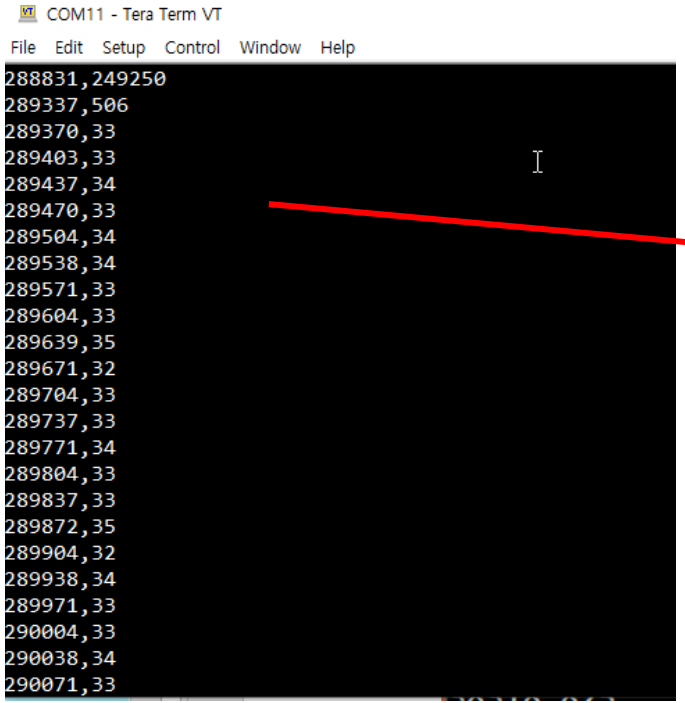
ENTER키를 더블 클릭하면 첫번재 클릭때는 큰값,
두번째 클릭때는 155정도로 작은 값

즉, 1값이 1ms 이므로 더블 클릭을 구분하려면
100~200ms정도의 값이 time_interval로 저장되면
더블 클릭으로 간주

UART 버전

버튼 대신 키보드 입력

key_interval_main.c



ENTER키를 계속 누르고 있으면 33~35정도의 값들이 쭉 출력됨.

즉, 길게 3초는 20~50ms의 값이 30개 이상 들어오면 3초이상 long key로 간주

Clock 설정 모드

Clock 설정 모드의 종류

설정모드 4가지

- 알람시계는 4가지 모드로 분류 가능
- 어떤 설정이 아닌 시간을 보여주는 모드는 NORMAL_STATE
- 시간을 설정하는 모드는 TIME_SETTING 모드
- 알람 시간을 설정하는 모드는 ALARM_TIME_SETTING 모드
- 알람 노래를 설정하는 모드는 MUSIC_SELECT모드
- 이 4가지 모드를 enum CLOCK_MODE로 표시.

```
enum CLOCK_MODE{
NORMAL_STATE,
TIME_SETTING,
ALARM_TIME_SETTING,
MUSIC_SELECT
};
```

Clock 설정 모드

Clock 설정 모드의 전환

모드간의 전환

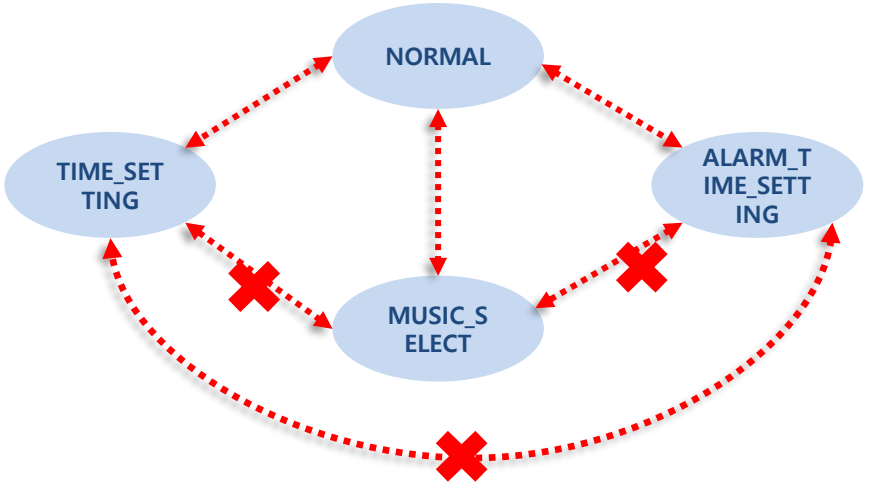
- 모드간의 전환은 Set 버튼으로 가능
- NORMAL_STATE에서 Set버튼을 누르면 TIME_SETTING 모드
- NORMAL_STATE에서 Set버튼을 3초 이상 누르면 ALARM_TIME_SETTING 모드
- NORMAL_STATE에서 Set버튼을 Double Click하면 MUSIC_SELECT모드
- NORMAL_STATE가 아닌 설정 모드에서 Set버튼을 누르면 다른 모드로 전환하지 않고 그냥 Set버튼
- NORMAL_STATE에서 up,down,right,left 버튼을 누르면 의미 없음.
- 설정 모드에서 Set, up,down,right,left 버튼은 의미 있음.

Clock 설정 모드

Clock 설정 모드의 전환

State machine

- 앞에서 설명한 내용을 도표로 나타내면 이해하기 쉬움.
- 도표로 나타낸 것을 state machine이라고 함.
- NORMAL mode에서 설정 모드로 진입 후 다시 NORMAL mode로 회귀
- 설정모드 끼리는 진입안됨.

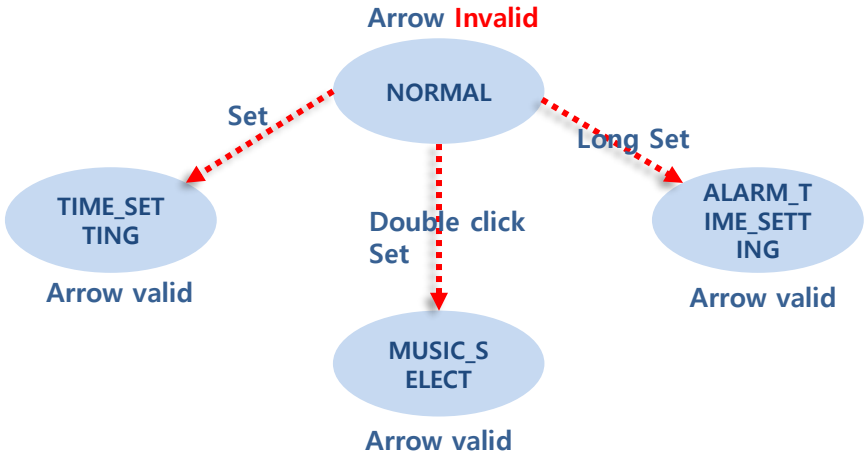


Clock 설정 모드

Clock 설정 모드의 버튼

버튼의 사용

- 모드마다 Set 버튼과 방향키(Arrow key)들의 사용을 정의할 수 있음.
- NORMAL mode에서 설정모드 진입은 Set 키로 가능
- NORMAL mode에서 방향키는 invalid
- 설정 모드에서 방향키는 valid



Clock 설정 모드

Clock 설정 모드의 버튼

모드와 버튼의 상태 저장

```
59  enum CLOCK_MODE{
60      NORMAL_STATE,
61      TIME_SETTING,
62      ALARM_TIME_SETTING,
63      MUSIC_SELECT
64  };
65
66  enum CLOCK_BUTTON{
67      NO_KEY,
68      UP,
69      DOWN,
70      RIGHT,
71      LEFT,
72      SEL
73  };
74
75  struct clock_state{
76      enum CLOCK_MODE mode;
77      enum CLOCK_BUTTON button;
78  };
79
80  struct clock_state current_state;
81
82
```

- 4가지 모드와 입력 버튼의 상태를 항상 저장해 두기 위해 current_state라는 구조체 변수를 생성
- current_state 구조체는 현재 mode 상태를 위한 enum CLOCK_MODE를 멤버로 가짐
- current_state 구조체는 현재 입력 버튼을 위한 enum CLOCK_BUTTON를 멤버로 가짐
- 이와 같이 관련있는 정보를 구조체로 묶어서 관리하면 코드가 이해하기 쉽고 향후 디버깅도 유리함.

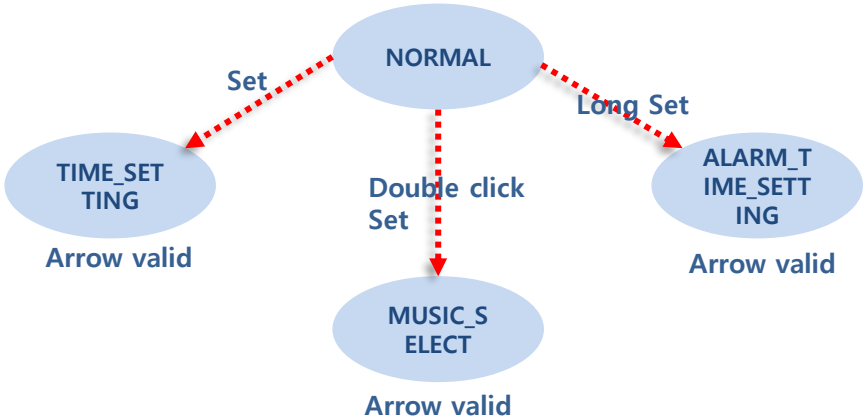
Clock 설정 모드

Clock 설정 모드의 버튼

모드 판단

```
181 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
182 {
183   if (huart->Instance == USART3)
184   {
185
186     current_time = HAL_GetTick();
187     time_interval = current_time - last_time;
188     last_time = current_time;
189
190     switch(current_state.mode)
191     {
192       case NORMAL_STATE:
193           switch(key_value)
194           {
195               case SEL_KEY:
196                   current_state.button = SEL;
197                   mode_analysis();
198                   break;
199               default:  // Arrow keys is Not valid in NORMAL state
200                   break;
201           }
202           break;
203       case TIME_SETTING:
204           current_state.mode  = NORMAL_STATE;
205           break;
206       case ALARM_TIME_SETTING:
207           current_state.mode  = NORMAL_STATE;
208           break;
209       case MUSIC_SELECT:
210           current_state.mode  = NORMAL_STATE;
211           break;
212       default:
213           break;
214     }
215
216
217     /* Receive one byte in interrupt mode */
218     HAL_UART_Receive_IT(&huart3, &key_value, 1);
219   }
220 }
```

- 시리얼 입력이 들어왔을 때 현재 모드와 키 값등을 판단
- Current_state.mode가 NORMAL_STATE이고 Set가 들어오면 모드 전환을 위해 mode_analysis함수 실행
- Mode_analysis()는 double click , Long키등을 판단
- Switch(key_value)에서 SEL_KEY만 판단하고 방향키는 무시



Clock 설정 모드

Clock 설정 모드의 버튼

Mode_anlaysis

```
void mode_analysis(void)
{
    if(time_interval>=LONG_CLICK_MIN && time_interval <=LONG_CLICK_MAX)
    {
        long_key_count++;
    }
    else if(time_interval>=DOUBLE_CLICK_MIN && time_interval <=DOUBLE_CLICK_MAX)
    {
      current_state.mode = MUSIC_SELECT;
      long_key_count = 0;

      memset(uart_buf,0,sizeof(uart_buf));
      sprintf(uart_buf,"MUSIC_SELECT %d\r\n",time_interval);
      HAL_UART_Transmit_IT(&huart3,uart_buf,sizeof(uart_buf));

    }
    else if(time_interval>= NORMAL_CLICK_MIN)
    {
      current_state.mode = TIME_SETTING;
      long_key_count = 0;

      memset(uart_buf,0,sizeof(uart_buf));
      sprintf(uart_buf,"TIME_SETTING  %d\r\n",time_interval);
      HAL_UART_Transmit_IT(&huart3,uart_buf,sizeof(uart_buf));

    }

    if(long_key_count>=LONG_CLICK_COUNT)
    {
      current_state.mode  = ALARM_TIME_SETTING;
      long_key_count = 0;

      memset(uart_buf,0,sizeof(uart_buf));
      sprintf(uart_buf,"ALARM_TIME_SETTING  %d\r\n",time_interval);
      HAL_UART_Transmit_IT(&huart3,uart_buf,sizeof(uart_buf));
    }
}
/* USER CODE END 0 */
```

```
#define LONG_CLICK_MIN 20
#define LONG_CLICK_MAX 50
#define LONG_CLICK_COUNT 30

#define DOUBLE_CLICK_MIN 100
#define DOUBLE_CLICK_MAX 200

#define NORMAL_CLICK_MIN 500
```

- Long 키는 time_interval이 20~50ms가 30가 이상 연속되었을 때 long키라고 판단
- Double click은 time_interval 이 100~200ms 일 때라고 판단
- 일반 키는 time_interval이 500ms 이상이라고 판단

Printf 추가

printf

STM32 용 printf

- HAL_UART_Transmit_IT()과 같은 코드는 복잡하니 익숙한 printf를 추가
- 두 군데 코드를 추가하면 됨.

```
113 #ifdef __GNUC__
114   /* With GCC, small printf (option LD Linker->Libraries->Small printf
115      set to 'Yes') calls __io_putchar() */
116   #define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
117 #else
118   #define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
119 #endif /* __GNUC__ */
120
```

```
616 /* USER CODE BEGIN 4 */
617 /**
618   * @brief  Retargets the C library printf function to the USART.
619   * @param  None
620   * @retval None
621   */
622 PUTCHAR_PROTOTYPE
623 {
624   /* Place your implementation of fputc here */
625   /* e.g. write a character to the EVAL_COM1 and Loop until the end of transmission */
626   HAL_UART_Transmit(&huart3, (uint8_t *)&ch, 1, 0xFFFF);
627
628   return ch;
629 }
630
631 /* USER CODE END 4 */
```

현재 시간, 알람 시간, 알람 음악 설정

```
254 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
255 {
256
257   if (huart->Instance == USART3)
258   {
259
260     current_time = HAL_GetTick();
261     time_interval = current_time - last_time;
262     last_time = current_time;
263
264     switch(current_state.mode)
265     {
266       case NORMAL_STATE:
267           switch(key_value)
268           {
269               case SEL_KEY:
270                   current_state.button = SEL;
271                   mode_analysis();
272                   break;
273               default:  // Arrow keys is Not valid in NORMAL state
274                   break;
275           }
276           break;
277       case TIME_SETTING:
278           stime_setting();
279           break;
280       case ALARM_TIME_SETTING:
281           atime_setting();
282           break;
283       case MUSIC_SELECT:
284           music_select();
285           break;
286       default:
287           break;
288     }
289     last_time_interval = time_interval;
```

- 현재 시간 설정은 stime_setting()함수
- 알람 시간 설정은 atime_setting()함수
- 알람 음악 설정은 music_select()함수

현재 시간, 알람 시간, 알람 음악 설정

시간 관련 변수

시간 관련 3가지 변수

```
83  typedef struct {
84    int8_t hours;
85    int8_t minutes;
86    int8_t seconds;
87  }TimeTypeDef;
88
89  TimeTypeDef ctime;  // current time
90  TimeTypeDef stime; // setting time
91  TimeTypeDef atime;  // alarm time
92
```

- 시,분,초를 3개의 멤버 변수를 가진 TimeTypeDef 변수로 지정
- TimeTypeDef 형식의 3개의 변수 선언
- Ctime은 현재 시간. 즉, timer 인터럽트에서 업데이트 하는 시간
- Stime은 사용자가 설정하는 시간. 즉, stime_setting()에서 설정되는 시간
- Atime은 알람 설정 시간. 즉, atime_setting()에서 설정되는 시간

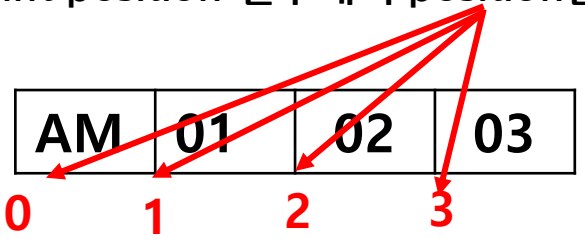
현재 시간, 알람 시간, 알람 음악 설정

현재 시간 설정

stime_setting

```
343 void stime_setting(void)
344 {
345 static  int position = 0;
346
347     if(position==0)
348     {
349         switch(key_value)
350         {
351           case SEL_KEY:
352           case RIGHT_KEY:
353               position =1;
354               break;
355           case UP_KEY:
356           case DOWN_KEY:
357               if(stime.hours>=12)
358                   stime.hours -=12;
359               else
360                   stime.hours +=12;
361               break;
362           default:
363               break;
364         }
365     }
366     else if(position==1)
367     {
368         switch(key_value)
369         {
370           case SEL_KEY:
371           case RIGHT_KEY:
372               position =2;
373               break;
374           case LEFT_KEY:
375               position =0;
376               break;
377           case UP_KEY:
378               stime.hours++;
```

- Static int position 변수에서 position은 셋팅 할 값의 위치



- AM이나 PM을 선택하는 위치는 position 이 0
- 시는 position이 1, 분은 position이 2, 초는 position이 3
- 각 위치에서 입력되는 key_value에 따라 stime변수를 조정.

현재 시간, 알람 시간, 알람 음악 설정

현재 시간 설정

stime_setting

```
else if(position==3)
{
    switch(key_value)
    {
        case SEL_KEY:
        case RIGHT_KEY:
            position = 0;
            ctime.hours = stime.hours;
            ctime.minutes = stime.minutes;
            ctime.seconds = stime.seconds;
            current_state.mode = NORMAL_STATE;
            break;
        case LEFT_KEY:
            position =2;
            break;
        case UP_KEY:
            stime.seconds++;
            if(stime.seconds>=60)
                stime.seconds = 0;
            break;
        case DOWN_KEY:
            stime.seconds--;
            if(stime.seconds<0)
                stime.seconds = 59;
            break;
        default:
            break;
    }
}
```

- Position이 3일 때는 초단위를 셋팅하는 것으로 SEL 이나 RIGHT 키가 들어오면 설정한 시간값인 stime을 현재 시간인 ctime으로 복사하고 모드를 NORMAL_STATE로 변환.

현재 시간, 알람 시간, 알람 음악 설정

알람 시간 설정

atime_setting

```
void atime_setting(void)
{
static  int position = 0;

    if(position==0)
    {
        switch(key_value)
        {
//          case SEL_KEY:
            case RIGHT_KEY:
                position =1;
                break;
            case UP_KEY:
            case DOWN_KEY:
                if(atime.hours>=12)
                    atime.hours -=12;
                else
                    atime.hours +=12;
                break;
            default:
                break;
        }
    }
```

- 앞의 stime_setting과 거의 유사함
- SEL_KEY를 주석처리 한 것은 알람 설정이 Long키 이다보니 ALARM모드 선택 후 입력되는 SEL키를 계속 입력 받게 되어 position이동에 사용되므로 주석처리하고 RIGHT키만 이용하였음. 향후 실제 LCD모듈을 사용하면 해결될 문제

현재 시간, 알람 시간, 알람 음악 설정

알람 음악 설정

알람 음악 관련 변수

```
92
93  typedef struct {
94    int8_t music_num;
95    char music_title[16];
96  }MusicTypeDef;
97
98  MusicTypeDef alarm_music[] =
99  {
100   {0,"Three Bears"},
101   {1,"Spring Water"},
102   {2,"Bicycle"},
103   {3,"Home town"},
104   {4,"Mom"},
105
106 };
107
```

- 원래는 2개의 알람 음악이지만 여러 개의 선택을 연습하기 위해 5개로 설정
- 음악 번호, 음악 제목을 멤버 변수를 가진 MusicTypeDef 변수로 지정
- 총 5개의 알람을 alarm_music[] 배열로 지정

현재 시간, 알람 시간, 알람 음악 설정

알람 음악 선택

music_select

```
296 void music_select(void)
297 {
298   int pos, count;
299
300   pos = current_state.music_num;
301   count = sizeof(alarm_music)/sizeof(alarm_music[0]);  //total music count
302
303   switch(key_value)
304   {
305     case UP_KEY:
306         pos++;
307         if(pos==count)
308         {
309           pos =0;
310         }
311         break;
312     case DOWN_KEY:
313         pos--;
314         if(pos < 0)
315         {
316           pos =count-1;
317         }
318         break;
319       case SEL_KEY:
320       case RIGHT_KEY:
321           current_state.mode = NORMAL_STATE;
322           break;
323
324   }
325   current_state.music_num =  pos;
```

- Music_select()함수로 노래 선택
- 변수 pos는 현재 선택된 곡의 번호
- 변수 count는 총 노래의 개수. 배열의 원소 개수를 알려면 sizeof(배열이름)/sizeof(배열원소)
- 최대 번호인 count를 넘어가면 다시 pos를 0으로 셋팅하고 최소 번호인 0보다 작아지면 pos를 4로 셋팅
- 현재 선택된 곡 번호를 current_state.music_num에 저장. 향후 Flash메모리등에 저장

현재 시간, 알람 시간, 알람 음악 설정

알람 음악 선택

music_select

```
    current_state.music_num =  pos;

    memset(line,0,sizeof(line));
    if(pos <= count-2)
    {
      sprintf(line[0],"%s <\r\n",alarm_music[pos].music_title);  // "<" mark is selected music title
      sprintf(line[1],"%s \r\n",alarm_music[pos+1].music_title);
    }
    else
    {
      sprintf(line[0],"%s <\r\n",alarm_music[pos].music_title);
      sprintf(line[1],"%s \r\n",alarm_music[0].music_title);
    }

    HAL_UART_Transmit_IT(&huart3,(uint8_t *)line,sizeof(line));

}
```

- LCD가 2개의 라인이므로 5개중 선택된 2개의 곡만 display함.
- 곡중 < 로 표시되면 현재 선택된 곡

Flash 메모리에 parameter 관리하기

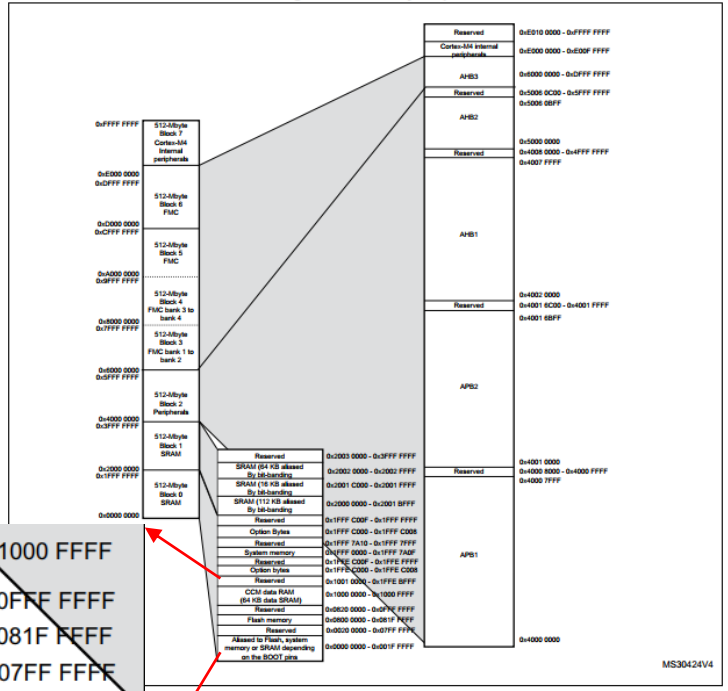
STM32F429의 플래시 메모리 소개

STM32F429의 메모리 맵

어드레스 범위 0x08000000~0x081FFFFF

2Mbyte의 범위 0~0x1FFFFF

Figure 19. Memory map



CCM data RAM (64 KB data SRAM)	0x1000 0000 - 0x1000 FFFF
Reserved	0x0820 0000 - 0x0FFF FFFF
Flash memory	0x0800 0000 - 0x081F FFFF
Reserved	0x0020 0000 - 0x07FF FFFF
Aliased to Flash, system memory or SRAM depending on the BOOT pins	0x0000 0000 - 0x001F FFFF

Flash 메모리에 parameter 관리하기

STM32F429의 플래시 메모리 소개

STM32F429의 메모리 모듈 구조

Table 6. Flash module - 2 Mbyte dual bank organization (STM32F42xxx and STM32F43xxx)

Block	Bank	Name	Block base addresses	Size
Main memory	Bank 1	Sector 0	0x0800 0000 - 0x0800 3FFF	16 Kbytes
		Sector 1	0x0800 4000 - 0x0800 7FFF	16 Kbytes
		Sector 2	0x0800 8000 - 0x0800 BFFF	16 Kbytes
		Sector 3	0x0800 C000 - 0x0800 FFFF	16 Kbyte
		Sector 4	0x0801 0000 - 0x0801 FFFF	64 Kbytes
		Sector 5	0x0802 0000 - 0x0803 FFFF	128 Kbytes
		Sector 6	0x0804 0000 - 0x0805 FFFF	128 Kbytes
		-	-	-
		-	-	-
		-	-	-
		Sector 11	0x080E 0000 - 0x080F FFFF	128 Kbytes
	Bank 2	Sector 12	0x0810 0000 - 0x0810 3FFF	16 Kbytes
		Sector 13	0x0810 4000 - 0x0810 7FFF	16 Kbytes
		Sector 14	0x0810 8000 - 0x0810 BFFF	16 Kbytes
		Sector 15	0x0810 C000 - 0x0810 FFFF	16 Kbytes
		Sector 16	0x0811 0000 - 0x0811 FFFF	64 Kbytes
		Sector 17	0x0812 0000 - 0x0813 FFFF	128 Kbytes
		Sector 18	0x0814 0000 - 0x0815 FFFF	128 Kbytes
			-	-
			-	-
			-	-
		Sector 23	0x081E 0000 - 0x081F FFFF	128 Kbytes
System memory			0x1FFF 0000 - 0x1FFF 77FF	30 Kbytes
OTP			0x1FFF 7800 - 0x1FFF 7A0F	528 bytes
Option bytes	Bank 1		0x1FFF C000 - 0x1FFF C00F	16 bytes
	Bank 2		0x1FFE C000 - 0x1FFE C00F	16 bytes

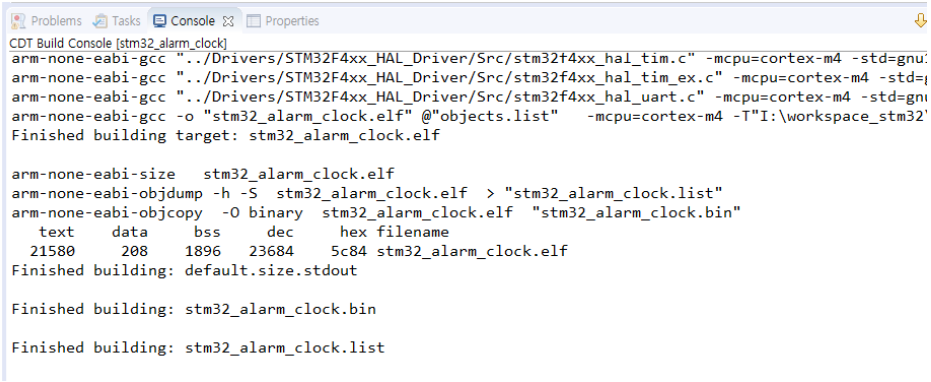
Main memory

- 주요 플래시 메모리 공간으로 2개의 Bank로 나뉨
- 각각은 16Kbytes 또는 64Kbytes, 128Kbytes 크기의 섹터로 나뉨
- 총 24개의 섹터로 구성(bank당 12개의 섹터)
- 플래시 메모리는 쓰기 전에 먼저 메모리를 지워야 하는데 지울 때 사용하는 단위가 sector단위
- 메모리를 지우는 방법은 한번에 칩 전체를 지우거나 sector 하나를 지정해서 지울 수 있음.

Flash 메모리에 parameter 관리하기

여기서 잠깐

Flash 메모리에 저장되는 코드와 데이터

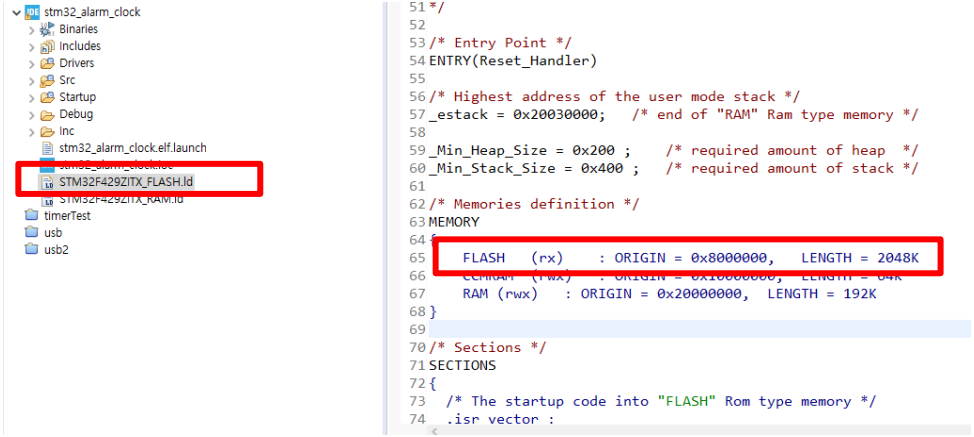


- 우리가 작성한 SW가 CPU의 어디에 저장될까? → ROM인 Flash 메모리
- Flash 메모리에는 C언어로 작성한 코드와 변수 등이 바이너리 값 형태로 저장됨.
- 그러면 Flash 메모리의 어느 위치에 저장될까?
- Build project 후 Console창에 위와 같은 화면이 출력되는데 여기서 text, data, bss, dec, hex 는 각각 무엇일까?

Flash 메모리에 parameter 관리하기

여기서 잠깐

Flash 메모리에 저장되는 코드와 데이터



- 코드와 데이터와 같은 바이너리가 저장될 위치를 지정하는 파일을 Linker script 파일이라고 하며 CubeIDE 환경에서는 STM32F429ZITX_FLASH.ld와 STM32F429ZITX_RAM.ld
- STM32F429ZITX_FLASH.ld파일에 바이너리가 Flash 메모리에 저장될 자세한 위치가 나옴.
- Linker script 와 같은 컴파일 환경의 문법은 우리가 사용하는 컴파일러가 GNU GCC이기 때문에 GCC의 문법을 따름 → 자세한 설명은 인터넷에 GNU Linker 설명을 찾아보면 많이 나옴.
- korea.gnu.org/manual/release/ld/ld-sjp/ld-ko_3.html

Flash 메모리에 parameter 관리하기

여기서 잠깐

Flash 메모리에 저장되는 코드와 데이터

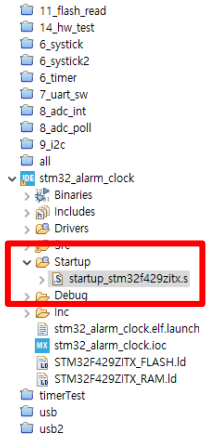
```
/* Memories definition */
MEMORY
{
    FLASH    (rx)    : ORIGIN = 0x8000000,   LENGTH = 2048K
    CCMRAM   (rwx)   : ORIGIN = 0x10000000,  LENGTH = 64K
    RAM (rwx)    : ORIGIN = 0x20000000,  LENGTH = 192K
}

/* Sections */
SECTIONS
{
  /* The startup code into "FLASH" Rom type memory */
  .isr_vector :
  {
    . = ALIGN(4);
    KEEP(*(.isr_vector)) /* Startup code */
    . = ALIGN(4);
  } >FLASH

  /* The program code and other data into "FLASH" Rom type memory */
  .text :
  {
    . = ALIGN(4);
    *(.text)           /* .text sections (code) */
    *(.text*)          /* .text* sections (code) */
    *(.glue_7)         /* glue arm to thumb code */
    *(.glue_7t)        /* glue thumb to arm code */
    *(.eh_frame)
```

- GNU GCC 컴파일 환경의 문법을 자세하게 알 필요는 없으나 중요한 부분 몇 개만 살펴보자
- MEMORY라는 정의부분은 전체 메모리의 정보를 담고 있고 그 중 FLASH는 rx (읽기와 실행하기 가능)속성을 가지고 시작 주소(ORIGIN) 는 0x08000000 이며 크기 (LENGTH)는 2048K 인 2MB
- SECTIONS는 컴파일된 바이너리 덩어리들 이라고 할 수 있음
- .isr_vector 섹션은 인터럽트 벡터들이 모여 있는 곳이고 FLASH 의 제일 처음 부분인 0x08000000에 저장하도록 지정됨.
- .text는 코드를 말하며 Build project 후 Console창에 출력 되는 text와 같은 의미임.
- 코드는 어셈블리 기계어(=instruction set)를 말함.
- 데이터는 변수와 같이 메모리 저장되는 값을 말함.

Flash 메모리에 parameter 관리하기

여기서 잠깐

Flash 메모리에 저장되는 코드와 데이터




```
129 /*******************************************************************
130 *
131 * The minimal vector table for a Cortex M3. Note that the prope
132 * must be placed on this to ensure that it ends up at physical
133 * 0x0000.0000.
134 *
135 *******************************************************************
136     .section  .isr_vector,"a",%progbits
137   .type  g_pfnVectors, %object
138   .size  g_pfnVectors, .-g_pfnVectors
139
140 g_pfnVectors:
141   .word  _estack
142   .word  Reset_Handler
143
144   .word  NMI_Handler
145   .word  HardFault_Handler
146   .word  MemManage_Handler
147   .word  BusFault_Handler
148   .word  UsageFault_Handler
149   .word  0
150   .word  0
```

```
74   .section  .text.Reset_Handler
75   .weak  Reset_Handler
76   .type  Reset_Handler, %function
77 Reset_Handler:
78   ldr   sp, =_estack      /* set stack pointer */
79
80 /* Copy the data segment initializers from flash to SRAM */
81   movs  r1, #0
82   b  LoopCopyDataInit
83
84 CopyDataInit:
85   ldr  r3, =_sidata
86   ldr  r3, [r3, r1]
87   str  r3, [r0, r1]
88   adds  r1, r1, #4
```

- Startup/startup_stm32f429zitx.s 파일이 최초 실행되는 어셈블리어 파일임
- 이 파일에 .isr_vector라는 이름의 섹션이 있으며 이 섹션은 Reset_Handler, NMI_Handler등의 인터럽트 벡터들이 있음.
- 이 코드들이 플래시메모리의 제일 처음 부분인 0x08000000 을 채운다고 할 수 있음.
- Reset_Handler의 내용은 startup_stm32f429zitx.s 의 첫 부분에 있음
- NMI_Handler등의 내용은 stm32f4xx_it.c 파일에 있음

```
68 /**
69   * @brief This function handles Non maskable interrupt.
70   */
71 void NMI_Handler(void)
72 {
73   /* USER CODE BEGIN NonMaskableInt_IRQn 0 */
74
75   /* USER CODE END NonMaskableInt_IRQn 0 */
76   /* USER CODE BEGIN NonMaskableInt_IRQn 1 */
77
78   /* USER CODE END NonMaskableInt_IRQn 1 */
79 }
```

Flash 메모리에 parameter 관리하기

여기서 잠깐

Flash 메모리에 저장되는 코드와 데이터

구분	내용	저장 위치	사용
text	함수, 제어문, 상수등 (=code)	ROM	컴파일 시에 크기가 결정되고 이후로 변하지 않음
data	초기값이 있는 전역 변수	ROM	
bss	초기값이 없는 전역변수	RAM	
Heap	동적 할당(malloc)된 지역 변수	RAM	동작 시(=run time)에 사용되며 heap은 아래로 어드레스로 증가하고 stack은 위로 어드레스 증가
Stack	정적 할당된 지역 변수	RAM	

- 컴파일 콘솔창의 text, data, bss는 위의 내용이고 dec는 text와 data와 bss의 크기를 합친 값을 10진수로 hex 는 16진수로 보여줌
- Flash

Flash 메모리에 parameter 관리하기

여기서 잠깐

Flash 메모리에 저장되는 코드와 데이터

전역 변수 ctime,stime, atime은 초기화되지 않았으므로 bss에 속함

```
89  TimeTypeDef ctime;  // current time
90  TimeTypeDef stime;  // setting time
91  TimeTypeDef atime;  // alarm time
92
93  typedef struct {
94    int8_t music_num;
95    char music_title[16];
96  }MusicTypeDef;
97
98  MusicTypeDef alarm_music[] =
99  {
100   {0,"Three Bears"},
101   {1,"Spring Water"},
102   {2,"Bicycle"},
103   {3,"Home town"},
104   {4,"Mom"},
105
106 };
```

전역 변수 alarm_music 은 초기화되어 있으므로 data에 속함

```
189 void time_display(void)
190 {
191   uint8_t hours;
192   uint8_t minutes;
193   uint8_t seconds;
194
195   memset(line,0,sizeof(line));
196
197   if(current_state.mode == NORMAL_STATE)
198   {
199       sprintf(line[0],"Korea Polytech   \r\n");
200       hours = ctime.hours;
201       minutes = ctime.minutes;
202       seconds = ctime.seconds;
203   }
204   else if(current_state.mode == TIME_SETTING)
205   {
206       sprintf(line[0],"Time Setting     \r\n");
207       hours = stime.hours;
208       minutes = stime.minutes;
209       seconds = stime.seconds;
210   }
```

지역 변수 hours, minutes, seconds는 정적할당이므로 stack에 속함

모든 함수, 실행문들은 text에 속함

전역 변수인 last_time등은 초기화되지 않았으므로 bss에 속함

```
241 uint32_t last_time,current_time,time_interval, last_time_interval;
242 uint32_t long_key_count;
243 /* Arrow Keys mapping
244  * UP = 'A' = 65
245  * DOWN = 'B' = 66
246  * RIGHT = 'C' = 67
247  * LEFT = 'D' = 68
248  * ENTER = 13
249  */
250 /*
251  * Double click : time_interval = 100~200ms
252  * Long Key : time_interval = 20~50ms in 30 consecutive times
253  */
254 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
255 {
256
257   if (huart->Instance == USART3)
258   {
259
260     current_time = HAL_GetTick();
261     time_interval = current_time - last_time;
262     last_time = current_time;
```

STM 라이브러리 함수들도 모두 text에 속함

Flash 메모리에 parameter 관리하기

여기서 잠깐

Flash 메모리에 저장되는 코드와 데이터

Problems Tasks Console Properties

CDT Build Console [stm32_alarm_clock]

```
arm-none-eabi-gcc -o "stm32_alarm_clock.elf" @"objects.list"   -mcpu=cortex-m4 -T"I:\workspac
Finished building target: stm32_alarm_clock.elf

arm-none-eabi-objdump -h -S  stm32_alarm_clock.elf  > "stm32_alarm_clock.list"
arm-none-eabi-objcopy  -O binary  stm32_alarm_clock.elf  "stm32_alarm_clock.bin"
arm-none-eabi-size   stm32_alarm_clock.elf
   text    data     bss     dec     hex filename
  21580     208    1896   23684    5c84 stm32_alarm_clock.elf
Finished building: default.size.stdout

Finished building: stm32_alarm_clock.bin

Finished building: stm32_alarm_clock.list
```

```
I:₩workspace_stm32₩stm32_alarm_clock₩Debug>dir
 I 드라이브의 볼륨: ext
 볼륨 일련 번호: 7ED3-2234

 I:₩workspace_stm32₩stm32_alarm_clock₩Debug 디렉터리

2020-03-12  오후 01:21    <DIR>          .
2020-03-12  오후 01:21    <DIR>          ..
2020-03-11  오후 05:00    <DIR>          Drivers
2020-03-12  오후 01:21             2,087 makefile
2020-03-12  오후 01:21             1,175 objects.list
2020-03-11  오후 05:00               239 objects.mk
2020-03-12  오후 01:21               516 sources.mk
2020-03-12  오후 01:21    <DIR>          Src
2020-03-12  오후 01:21    <DIR>          Startup
2020-03-12  오후 01:21            21,792 stm32_alarm_clock.bin
2020-03-12  오후 01:21         1,531,172 stm32_alarm_clock.elf
2020-03-12  오후 01:21           503,270 stm32_alarm_clock.list
2020-03-12  오후 01:21           434,503 stm32_alarm_clock.map
               8개 파일           2,314,754 바이트
               5개 디렉터리  545,447,571,456 바이트 남음
```

- ROM에 저장되는 영역은 text와 data영역임
- Text의 크기가 21580 bytes
- Data의 크기가 208 bytes
- 두 영역을 합하면 21788 bytes
- Stm32_alarm_clock.bin파일의 크기가 21792 bytes인데 앞의 값보다 4 bytes 차이가 남
- 첫번째 4 bytes stack의 끝 위치인 _estack = 0x20030000 의 값이 저장되어 있음

Flash 메모리에 parameter 관리하기

여기서 잠깐

Flash 메모리에 저장되는 코드와 데이터

HxD - [I:₩workspace_stm32₩stm32_alarm_clock₩Debug₩stm32_alarm_clock.bin]

파일(F) 편집(E) 찾기(S) 보기(V) 분석(A) 도구(T) 창 설정(W) 도움말(H)

16 | Windows (ANSI) | 16진수

stm32_alarm_clock.bin

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	00	00	03	20	91	40	00	08	6D	3E	00	08	7B	3E	00	08	... '@..m>..{>..
00000010	81	3E	00	08	87	3E	00	08	8D	3E	00	08	00	00	00	00	.>..‡>...>......
00000020	00	00	00	00	00	00	00	00	00	00	00	00	93	3E	00	08	..........."..>..
00000030	A1	3E	00	08	00	00	00	00	AF	3E	00	08	BD	3E	00	08	¡>.....¯>..½>..
00000040	E1	40	00	08	E1	40	00	08	E1	40	00	08	E1	40	00	08	á@..á@..á@..á@..
00000050	E1	40	00	08	E1	40	00	08	E1	40	00	08	E1	40	00	08	á@..á@..á@..á@..
00000060	E1	40	00	08	E1	40	00	08	E1	40	00	08	E1	40	00	08	á@..á@..á@..á@..
00000070	E1	40	00	08	E1	40	00	08	E1	40	00	08	E1	40	00	08	á@..á@..á@..á@..
00000080	E1	40	00	08	E1	40	00	08	E1	40	00	08	E1	40	00	08	á@..á@..á@..á@..
00000090	E1	40	00	08	E1	40	00	08	E1	40	00	08	E1	40	00	08	á@..á@..á@..á@..
000000A0	E1	40	00	08	E1	40	00	08	E1	40	00	08	E1	40	00	08	á@..á@..á@..á@..
000000B0	C9	3E	00	08	E1	40	00	08	E1	40	00	08	E1	40	00	08	É>..á@..á@..á@..
000000C0	E1	40	00	08	E1	40	00	08	E1	40	00	08	E1	40	00	08	á@..á@..á@..á@..
000000D0	E1	40	00	08	E1	40	00	08	E1	40	00	08	DD	3E	00	08	á@..á@..á@..Ý>..
000000E0	E1	40	00	08	E1	40	00	08	E1	40	00	08	E1	40	00	08	á@..á@..á@..á@..
000000F0	E1	40	00	08	E1	40	00	08	E1	40	00	08	E1	40	00	08	á@..á@..á@..á@..
00000100	E1	40	00	08	E1	40	00	08	E1	40	00	08	E1	40	00	08	á@..á@..á@..á@..
00000110	E1	40	00	08	E1	40	00	08	E1	40	00	08	E1	40	00	08	á@..á@..á@..á@..
00000120	E1	40	00	08	E1	40	00	08	E1	40	00	08	E1	40	00	08	á@..á@..á@..á@..
00000130	E1	40	00	08	E1	40	00	08	E1	40	00	08	E1	40	00	08	á@..á@..á@..á@..
00000140	E1	40	00	08	E1	40	00	08	E1	40	00	08	E1	40	00	08	á@..á@..á@..á@..
00000150	E1	40	00	08	E1	40	00	08	E1	40	00	08	E1	40	00	08	á@..á@..á@..á@..
00000160	E1	40	00	08	E1	40	00	08	E1	40	00	08	E1	40	00	08	á@..á@..á@..á@..
00000170	E1	40	00	08	E1	40	00	08	E1	40	00	08	00	00	00	00	á@..á@..á@.....
00000180	E1	40	00	08	E1	40	00	08	E1	40	00	08	E1	40	00	08	á@..á@..á@..á@..
00000190	E1	40	00	08	E1	40	00	08	E1	40	00	08	E1	40	00	08	á@..á@..á@..á@..
000001A0	E1	40	00	08	E1	40	00	08	E1	40	00	08	00	00	00	00	á@..á@..á@.....
000001B0	10	B5	05	4C	23	78	33	B9	04	4B	13	B1	04	48	AF	F3	.µ.L#x3¹.K.±.H¯ó
000001C0	00	80	01	23	23	70	10	BD	C8	00	00	20	00	00	00	00	.€.##p.½È.. ....
000001D0	B8	52	00	08	08	B5	03	4B	1B	B1	03	49	03	48	AF	F3	¸R...µ.K.±.I.H¯ó
000001E0	00	80	08	BD	00	00	00	00	CC	00	00	20	B8	52	00	08	.€.½....Ì.. ¸R..
000001F0	03	46	13	F8	01	2B	00	2A	FB	D1	18	1A	01	38	70	47	.F.ø.+.*ûÑ...8pG
00000200	01	F0	FF	01	10	2A	2B	DB	10	F0	07	0F	08	D0	10	F8	.ðÿ..*+Û.ð...Ð.ø
00000210	01	3B	01	3A	8B	42	2D	D0	10	F0	07	0F	42	B3	F6	D1	.;.:‹B-Ð.ð..B³öÑ
00000220	F0	B4	41	EA	01	21	41	EA	01	41	22	F0	07	04	7F	F0	ð´Aê.!Aê.A"ð...ð
00000230	00	07	00	23	F0	F8	02	56	08	3C	85	EA	01	05	86	EA	...#ðø.V.<…ê..†ê

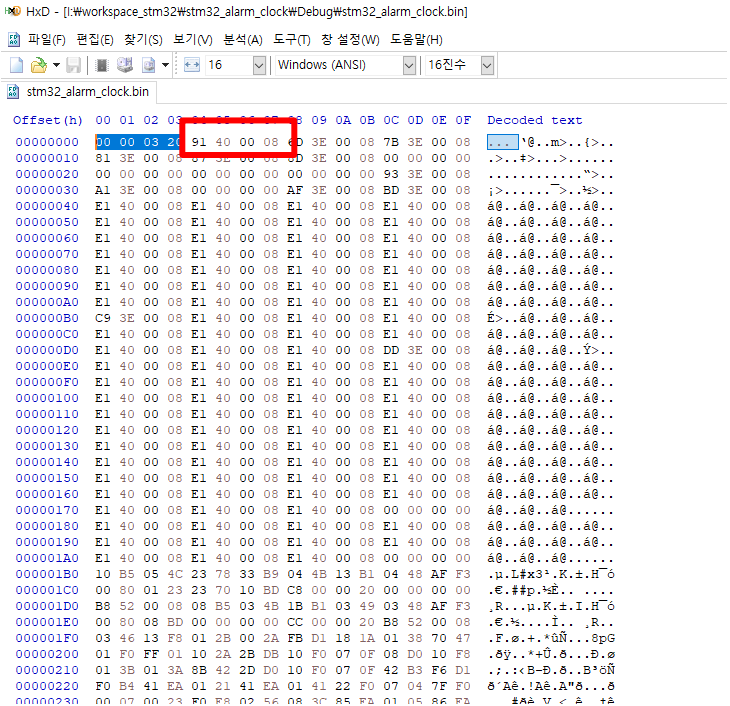
- 그림은 HxD hex editor라는 툴(https://mh-nexus.de/en/hxd/) 로 stm32_alarm_clock.bin 파일을 열어 본 내용
- Edian문제로 바이트 순서가 뒤집힌 상태
- 처음 4바이트는 00 00 03 20 인데 순서를 바꾸면 20030000 임
- 20030000은 _estack의 값
- _estack의 값은 STM32F429ZITX_RAM.ld 에 지정되어 있음

```
50 ******************************************************************
51 */
52
53 /* Entry Point */
54 ENTRY(Reset_Handler)
55
56 /* Highest address of the user mode stack */
57 _estack = 0x20030000;    /* end of "RAM" Ram type memory */
58
59 _Min_Heap_Size = 0x200; /* required amount of heap  */
60 _Min_Stack_Size = 0x400;    /* required amount of stack */
61
62 /* Memories definition */
63 MEMORY
64 {
65     FLASH    (rx)    : ORIGIN = 0x8000000,   LENGTH = 2048K
66     CCMRAM   (rwx)   : ORIGIN = 0x10000000,  LENGTH = 64K
67     RAM (rwx)    : ORIGIN = 0x20000000,  LENGTH = 192K
68 }
69
```

Flash 메모리에 parameter 관리하기

여기서 잠깐

Flash 메모리에 저장되는 코드와 데이터



- 다음은 91 40 00 08 로 뒤집으면 08004091
- Stm32_alarm_clock.list은 심볼(함수명, 변수명, 상수명)의 위치를 알려줌
- 아래 그림과 같이 Reset_Handler는 0x08004090의 위치에 있음

```
10659  800407e:   f85d 7b04   ldr.w   r7, [sp], #4
10660  8004082:   4770        bx  lr
10661  8004084:   e000ed00    .word   0xe000ed00
10662  8004088:   40023800    .word   0x40023800
10663  800408c:   24003010    .word   0x24003010
10664
10665 08004090 <Reset_Handler>:
10666
10667     .section  .text.Reset_Handler
10668   .weak  Reset_Handler
10669   .type  Reset_Handler, %function
10670 Reset_Handler:
10671   ldr   sp, =_estack       /* set stack pointer */
10672  8004090:   f8df d034   ldr.w   sp, [pc, #52]   ; 80040c8 <LoopF
10673
10674 /* Copy the data segment initializers from flash to SRAM */
10675   movs  r1, #0
10676  8004094:   2100        movs    r1, #0
```

Flash 메모리에 parameter 관리하기

여기서 잠깐

Flash 메모리에 저장되는 코드와 데이터

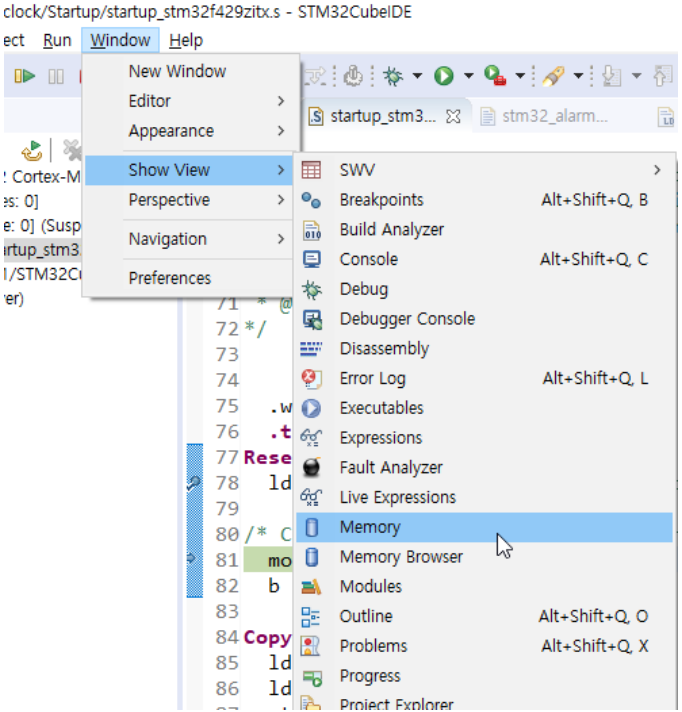
```
1
2 stm32_alarm_clock.elf:     file format elf32-littlearm
3
4 Sections:
5 Idx Name          Size      VMA       LMA       File off  Algn
6   0 .isr_vector   000001ac  08000000  08000000  00010000  2**0
7                   CONTENTS, ALLOC, LOAD, READONLY, DATA
8   1 .text         00005120  080001b0  080001b0  000101b0  2**4
9                   CONTENTS, ALLOC, LOAD, READONLY, CODE
10  2 .rodata       00000178  080052d0  080052d0  000152d0  2**2
11                  CONTENTS, ALLOC, LOAD, READONLY, DATA
12  3 .ARM.extab    00000000  08005448  08005448  000200c8  2**0
13                  CONTENTS
14  4 .ARM          00000008  08005448  08005448  00015448  2**2
15                  CONTENTS, ALLOC, LOAD, READONLY, DATA
```

- 그림은 Stm32_alarm_clock.list 파일의 앞부분
- .isr_vector 의 크기는 0x1ac, 시작주소는 0x08000000
- .text의 크기가 0x5120 이고 시작주소는 0x080001b0

Flash 메모리에 parameter 관리하기

여기서 잠깐

Flash 메모리에 저장되는 코드와 데이터



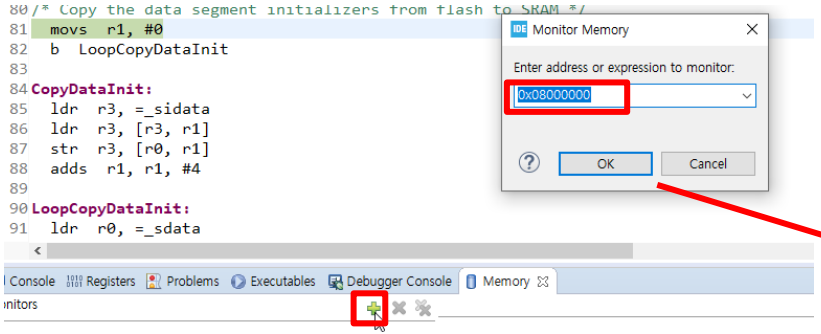
- flash 메모리에 우리가 작성한 코드 결과물인 Stm32_alarm_clock.bin 파일이 적혀 있는지 확인해보자.
- St-link 디버그 모드를 사용하면 메모리를 바로 읽어 볼 수 있음.
- 디버그 모드로 진입한 후 Window → Show View → Memory를 선택

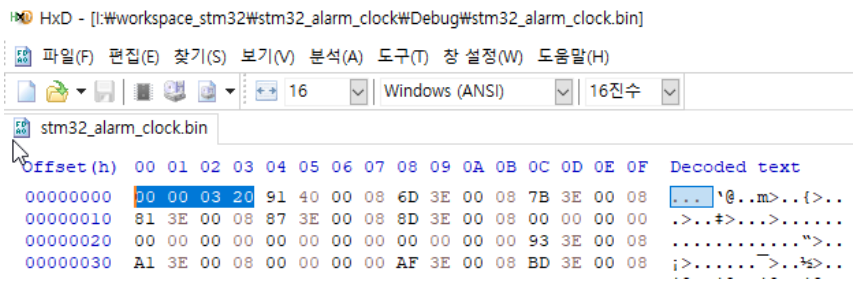
Flash 메모리에 parameter 관리하기

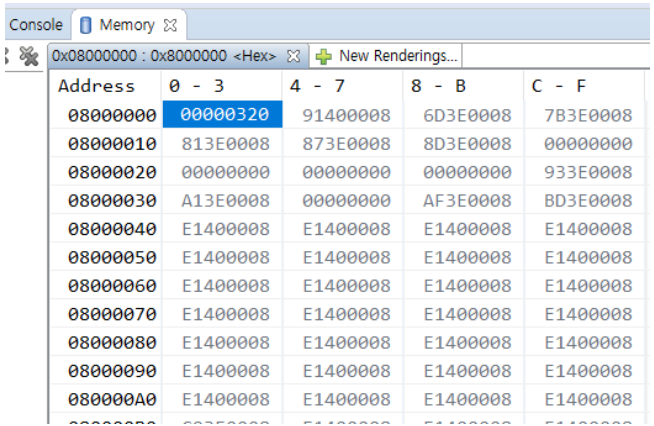
여기서 잠깐

Flash 메모리에 저장되는 코드와 데이터

- + 를 누르고 어드레스 창에 0x08000000을 입력 후 OK선택
- 앞에서 확인했던 stm32_alarm_clock.bin 의 내용과 동일함을 확인.







Flash 메모리에 parameter 관리하기

Parameter를 저장할 공간 선정

코드와 데이터가 저장된 공간

Table 6. Flash module - 2 Mbyte dual bank organization (STM32F42xxx and STM32F43xxx)

Block	Bank	Name	Block base addresses	Size
Main memory	Bank 1	Sector 0	0x0800 0000 - 0x0800 3FFF	16 Kbytes
		Sector 1	0x0800 4000 - 0x0800 7FFF	16 Kbytes
		Sector 2	0x0800 8000 - 0x0800 BFFF	16 Kbytes
		Sector 3	0x0800 C000 - 0x0800 FFFF	16 Kbyte
		Sector 4	0x0801 0000 - 0x0801 FFFF	64 Kbytes
		Sector 5	0x0802 0000 - 0x0803 FFFF	128 Kbytes
		Sector 6	0x0804 0000 - 0x0805 FFFF	128 Kbytes
		-	-	-
		-	-	-
		-	-	-
		Sector 11	0x080E 0000 - 0x080F FFFF	128 Kbytes
	Bank 2	Sector 12	0x0810 0000 - 0x0810 3FFF	16 Kbytes
		Sector 13	0x0810 4000 - 0x0810 7FFF	16 Kbytes
		Sector 14	0x0810 8000 - 0x0810 BFFF	16 Kbytes
		Sector 15	0x0810 C000 - 0x0810 FFFF	16 Kbytes
		Sector 16	0x0811 0000 - 0x0811 FFFF	64 Kbytes
		Sector 17	0x0812 0000 - 0x0813 FFFF	128 Kbytes
		Sector 18	0x0814 0000 - 0x0815 FFFF	128 Kbytes
			-	-
			-	-
			-	-
		Sector 23	0x081E 0000 - 0x081F FFFF	128 Kbytes
System memory			0x1FFF 0000 - 0x1FFF 77FF	30 Kbytes
OTP			0x1FFF 7800 - 0x1FFF 7A0F	528 bytes
Option bytes	Bank 1		0x1FFF C000 - 0x1FFF C00F	16 bytes
	Bank 2		0x1FFE C000 - 0x1FFE C00F	16 bytes

- stm32_alarm_clock.bin 의 크기가 21792(=0x5520) bytes 이 므로 약 22KB임
- Flash 메모리 시작 주소인 0x08000000부터 22KB는 코드와 데이터가 사용. 정확하게는 0x08000000~0x08005520 사용
- 그러면 sector 0번과 1번을 사용하고 있음.
- Sector 2번 부터 parameter 저장할 공간으로 사용할 수 있음
- 하지만 향후 코드가 늘어나 저장 공간이 커지면 Sector 2번도 사용하여 parameter저장 공간과 겹칠 수 있으므로 넉넉하게 뒤로 미루는 것이 바람직.
- 본 학습에서는 sector 23번(=0x081E0000)을 사용하여 parameter를 저장할 예정

Flash 메모리에 parameter 관리하기

Flash 메모리 다루기

Flash 메모리를 읽기

- Flash 메모리를 읽는 것은 일반 메모리 읽는 것과 동일
- C언어의 포인터를 사용하면 쉽게 Flash 메모리의 내용을 읽을 수 있음.
- 포인터를 사용하여 *(읽고자 하는 주소) 와 같은 형태로 읽음.
- 예를 들어 0x08000000 번지에 저장된 값을 4byte 읽는 코드

```
uint32_t value;

value  =  *((uint32_t *)0x08000000);
```

Flash 메모리에 parameter 관리하기

Flash 메모리 다루기

Flash 메모리에 쓰기

- Flash 메모리는 바로 쓸 수 없고 먼저 지운 다음에 쓸 수 있음.
- 공장 출하시 Flash 메모리는 모두 지워진 상태며 모든 비트가 1로 채워져 있음.
- 지우고 나면 Flash 메모리에는 값이 0xFFFFFFFF으로 채워 짐 (지우면 값이 0이 아님)
- 평소에는 지워지지 않기 위해 lock을 걸어두고 쓰고 싶을 때만 unlock 하여 사용
- 일반적인 Flash 메모리 쓰기 과정
  1. Unlock : HAL_FLASH_Unlock()
  2. Erase : HAL_FLASHEx_Erase() , 지우는 단위는 sector단위 또는 칩 전체
  3. Program : HAL_FLASH_Program() , 쓰는 단위는 1byte, 2,4,8 bytes 가능
  4. Lock : HAL_FLASH_Lock()

Flash 메모리에 parameter 관리하기

Flash 메모리 다루기

Flash 메모리에 쓰기

```
HAL_FLASH_Unlock();

FirstSector = FLASH_SECTOR_23;  // 지우고자 하는 섹터 번호
NbOfSectors = 1; // 지우고자 하는 섹터 개수
 /* Fill EraseInit structure*/
 EraseInitStruct.TypeErase     = FLASH_TYPEERASE_SECTORS;
 EraseInitStruct.VoltageRange  = FLASH_VOLTAGE_RANGE_3;
 EraseInitStruct.Sector        = FirstSector;
 EraseInitStruct.NbSectors     = NbOfSectors;

 if(HAL_FLASHEx_Erase(&EraseInitStruct, &SECTORError) != HAL_OK)
 {
                return -1;
 }
 HAL_FLASH_Program(FLASH_TYPEPROGRAM_WORD, Address, 0x12345678); // 4bytes 크기로 쓰고자하는
Address 에 0x12345678를 씀
 HAL_FLASH_Lock();  // 다 쓰고난 다음에 lock 시켜줌
```

- 23번째 sector를 지우고 시작주소인 0x081E0000에 0x12345678을 쓰는 예제

Flash 메모리에 parameter 관리하기

NV item

저장할 Parameter 선정

- STM32 블루투스 알람 시계 프로젝트 알고리즘에서 flash에 저장할 항목은 다음과 같음.
- 설정한 시간, 알람 시간, 선택한 알람 노래
- 이 항목들을 효율적으로 관리하기 위해 개별적으로 처리하지 않고 하나의 구조체로 묶어서 처리할 예정

- 일반적으로 이렇게 계속 저장하면서 관리하는 parameter 를 NV item이라고 함.
- NV item은 Non Volatile item의 약자
- 실제 제품상의 예로는 휴대폰의 전화번호부, 텔레비전의 채널번호나 볼륨 크기등 다양함.
- 대부분의 전자제품에는 NV item이 필요하며 저장공간으로 EEPROM이나 Flash 메모리를 사용함.
- NV item 공간이 대용량으로 필요하거나 동적으로 사용해야 한다면 저장 공간에 파일시스템을 올려 NV item들을 파일로 처리함. 마치 하드디스크와 흡사하게 사용
- 안드로이드 스마트폰의 대부분은 Flash 메모리에 EXT4라는 파일시스템을 사용하여 NV item들을 관리함.

Flash 메모리에 parameter 관리하기

NV item

저장할 Parameter 선정

```
typedef struct {
  uint32_t magic_num;
  TimeTypeDef setting_time;
  TimeTypeDef alarm_time;
  int8_t alarm_music_num;
}NVitemTypeDef;
```

- 설정 시간, 알람 시간, 선택한 알람 노래를 NV item으로 선정
- 첫번째 멤버인 magic_num의 역할
  - ✓ Flash 메모리는 초기값이 0xFFFFFFFF임.
  - ✓ 초기값인 0xFF 값을 그대로 사용하면 안되기 때문에 한번이라도 Nvitem을 쓴적이 있는지 검사하는 값
- 설정 시간은 setting_time
- 알람 시간은 alarm_time
- 선택한 알람 노래는 music_num에 저장

Flash 메모리에 parameter 관리하기

NV item

NV item 구조

```
#define MAGIC_NUM 0xdeadbeef
#define nv_items  ((NVitemTypeDef *) ADDR_FLASH_SECTOR_23)

NVitemTypeDef default_nvitem =
{
  MAGIC_NUM,
  {0,0,0},
  {0,0,0},
  0
};
```

- MAGIC_NUM은 magic_num 자리에 적을 상수 값
- nv_items 는 ADDR_FLASH_SECTOR_23 = 0x081E0000을 시작 주소로 하는 NVitemTypeDef 구조체 구조로 관리되는 값
- 이와 흡사한 구조가 #define GPIOB ((GPIO_TypeDef *) GPIOB_BASE)
- nv_items->magic_num 는 첫번째 멤버로 0x081E00000를 주소로 하는 4byte값
- nv_items->setting_time 는 0x081E00004를 주소로 하는 설정 시간이 저장된 3byte 값 (TimeTypeDef가 3byte)
- nv_items->alarm_time 는 0x081E00007를 주소로 하는 알람 시간이 저장된 3byte 값 (TimeTypeDef가 3byte)
- nv_items->alarm_music_num 는 0x081E0000A를 주소로 하는 알람 노래 번호가 저장된 1byte 값
- 부팅시 Flash ROM에 저장된 nv_items 에 값들을 RAM에 위치하는 default_nvitem로 복사하여 사용.

Flash 메모리에 parameter 관리하기

NV item

NV item 사용

```
if(nv_items->magic_num == MAGIC_NUM)
 {
  memcpy(&default_nvitem,nv_items,sizeof(NVitemTypeDef));
  ctime.hours = default_nvitem.setting_time.hours;
  ctime.minutes = default_nvitem.setting_time.minutes;
  ctime.seconds = default_nvitem.setting_time.seconds;
 }
 else
 {
   update_nvitems();
 }
```

Flash 메모리 초기값인 0xFFFFFFFF가 아니고 상수값인 0xdeadbeef이면 → 즉, 한번이라도 NV item을 flash 메모리에 업데이트 한 적이 있다면

Flash 메모리에 저장된 nv_items값을 RAM에 저장된 변수인 default_nvitem으로 복사

복사된 default_nvitem값중 설정 시간값을 현재 시간 변수인 ctime으로 복사하여 현재 시간을 display

Flash 메모리가 초기값으로 되어 있으면 default_nvitem 값으로 flash 메모리를 초기화

Flash 메모리에 parameter 관리하기

NV item

NV item 업데이트

```
HAL_StatusTypeDef update_nvitems(void)
{
....
  uint8_t *ptr;
  HAL_FLASH_Unlock();
  FirstSector = FLASH_SECTOR_23;
  NbOfSectors = 1;

  EraseInitStruct.TypeErase     = FLASH_TYPEERASE_SECTORS;
  EraseInitStruct.Sector        = FirstSector;
  EraseInitStruct.NbSectors     = NbOfSectors;
  error = HAL_FLASHEx_Erase(&EraseInitStruct, &SECTORError);

  ptr = (uint8_t*)&default_nvitem;
  for(i=0;i<sizeof(NVitemTypeDef);i++)
  {
    Address = (uint8_t*)nv_items+i;
    Data = *((uint8_t*)ptr+ i);
    HAL_FLASH_Program(FLASH_TYPEPROGRAM_BYTE,Address,Data);
  }
   HAL_FLASH_Lock();
```

23번째 섹터에 nvitem들을 저장하기 위해 23번 섹터를 erase

Default_nvitems의 시작 주소를 ptr변수에 복사

1바이트씩 Default_nvitems의 값을 23번 섹터에 write

Flash 메모리에 parameter 관리하기

NV item

NV item 업데이트

```
void stime_setting(void)
{
.........
  case SEL_KEY:
  case RIGHT_KEY:
    default_nvitem.setting_time.hours = stime.hours;
    default_nvitem.setting_time.minutes = stime.minutes;
    default_nvitem.setting_time.seconds = stime.seconds;
    update_nvitems();
    current_state.mode = NORMAL_STATE;
```

현재 시간 설정후 stime값을 flash에 저장

```
void atime_setting(void)
{
.........
  case SEL_KEY:
  case RIGHT_KEY:
    default_nvitem.alarm_time.hours = atime.hours;
    default_nvitem.alarm_time.minutes = atime.minutes;
    default_nvitem.alarm_time.seconds = atime.seconds;
    update_nvitems();
    current_state.mode = NORMAL_STATE;
```

알람 시간 설정후 atime값을 flash에 저장

```
void music_select (void)
{
.........
  case SEL_KEY:
  case RIGHT_KEY:
    default_nvitem.alarm_music_num = pos;
    update_nvitems();
    current_state.mode = NORMAL_STATE;
```

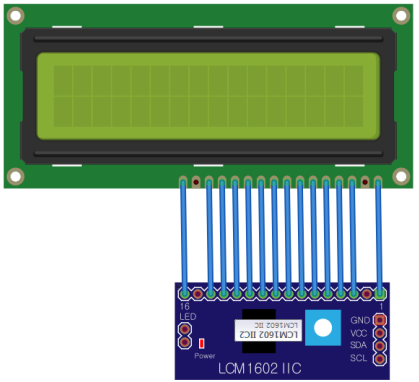
알람 노래 설정후 노래 번호를 flash에 저장

버튼 다루기

I2C 버스용 LCD 모듈

아두이노 LCM1602 IIC 쉴드

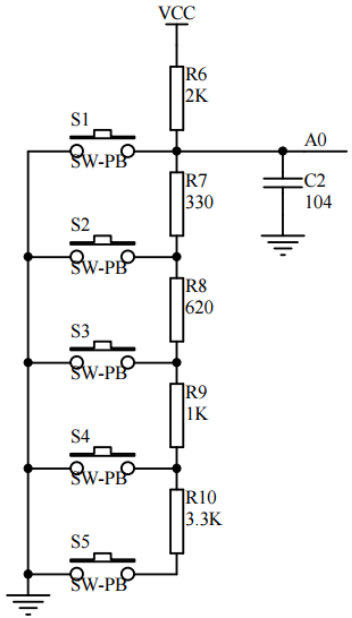
- 원래16개 핀의 인터페이스 필요
- 하지만 I2C 인터페이스를 가지는 컨트롤러를 중간에 사용하여 VCC, GND, I2C SDA, I2C SCL의 4개의 핀만을 가지고 제어 가능
- http://www.devicemart.co.kr/goods/view?no=1279486



버튼 다루기

I2C 버스용 LCD 모듈

아두이노 LCM1602 IIC 쉴드 회로도중 switch 회로

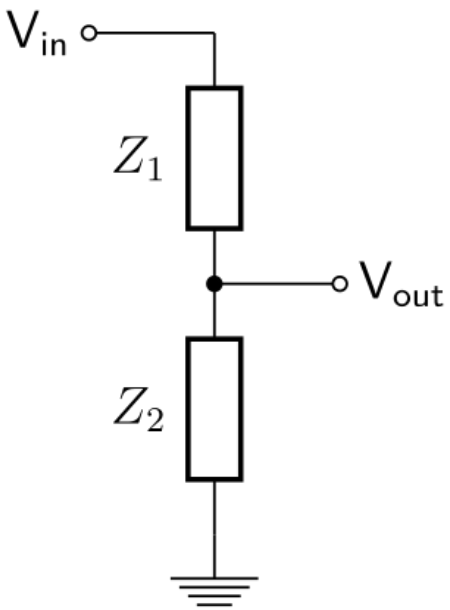


- Reset 버튼을 제외한 UP,DOWN, LEFT, RIGHT, SELECT의 5개의 버튼 회로도
- A0라는 아날로그 핀의 전압 레벨로 5개 버튼의 눌림 상태를 알 수 있음.
- 이 회로를 이해하려면 voltage divider를 이해해야 함.

버튼 다루기

I2C 버스용 LCD 모듈

Voltage divider



- V_{out}의 값은 아래 공식과 같이 Z_1과 Z_2 저항의 값으로 정해진다.

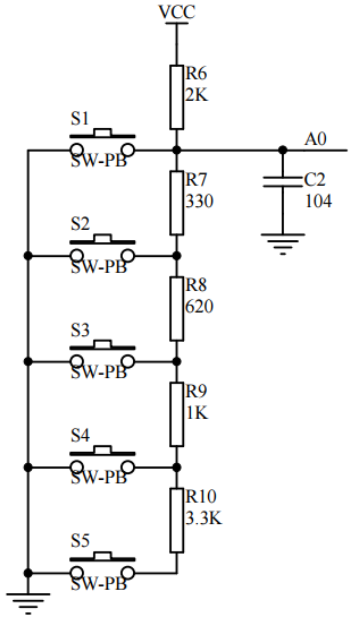
$$V_{\text{out}} = \frac{Z_2}{Z_1 + Z_2} \cdot V_{\text{in}}$$

- V_{in}= 5V, Z_1=100Ω, Z_2 = 200 Ω
- V_{out}= $\frac{200}{100+200}$*5=3.33V

버튼 다루기

I2C 버스용 LCD 모듈

아두이노 LCM1602 IIC 쉴드 회로도중 switch 회로



- 어떤 스위치도 누르지 않으면 R6은 pull-up저항역할이 되어 A0에 걸리는 전압은 5V
- S1을 누르면 접지와 연결되므로 0V
- S2를 누르면 R6과 R7의 voltage divider 에 의해 5V*(300/(2000+300)) = 0.7V
- S3을 누르면 R6과 R7+R8의 voltage divider 에 의해 5V*((300+620)/(2000+(300+620)) = 1.6V
- S4 는 5V*(/ +) = 2.46V
- S5 는 5V*(/ +) = 3.6V

I2C 버스용 LCD 모듈

LCD 모듈의 사양과 구조

LCD 디스플레이 뒷면 보드 설명

I2C Interface

SDA, SCL, VCC, GND로
이루어진 I2C 버스용 커넥터

Contrast ADJ

LCD의 Contrast를 조절하는 파란색 가변 저항

I2C Address Setting Jumper

- I2C Slave Address를 셋팅할 수 있는 헤더 핀
- 여러 개의 LCD 모듈을 같이 연결할 때 각각 구분하기 위함
- A0, A1, A2의 3개 점퍼로 모두 연결된 상태는 I2C Slave Address는 0x20

I2C 버스용 LCD 모듈

LCD 모듈의 사양과 구조

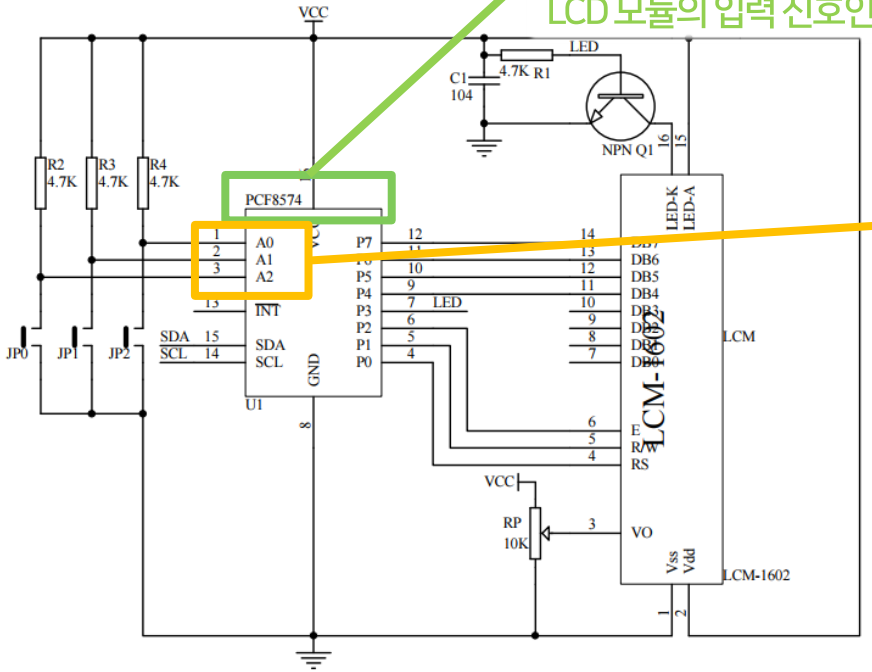
LCD 디스플레이 모듈의 사양

사양	
I2C 어드레스	0x20~0x27 (Default 0x27)
디스플레이	Blue 색상의 Back Light
공급 전원	5V
인터페이스	I2C/TWI 1개
특이사항	Contrast 조절 가능
5개 버튼	Analog port A0
사이즈	81x57mm

I2C 버스용 LCD 모듈

LCD 모듈의 사양과 구조

LCD 디스플레이 회로도



PCF8574 : 8bit I/O expander for I2C bus, I2C버스를 통해 8비트 핀을 제어

LCD 모듈의 입력 신호인 E, R/W, RS와 DB4~7 번 핀 제어

동일한 칩 여러 개 를 동일한 i2C 버스에 연결하려면 서로 I2C slave address가 달라야 함. PCF8574는 A0~A2의 값에 따라 address를 0x20~0x027까지 8개 달 수 있음.

A0~A2핀은 R2,R3,R4 의 pull-up 저항이 있으므로 JP0, JP2, JP3를 연결하지 않으면 모두 1의 값을 가짐

LCM1602 보드에 JP0, JP2, JP3가 연결되어 있지 않으므로 default address는 0x27