제목 : FM Radio 설계

목 차

1. 전체 시스템의 하드웨어 구성

1-1. 전체 시스템의 하드웨어 구성도

- 전체 시스템의 하드웨어의 구성도는 그림 1-1과 같다. 사용한 MCU는 STM32F407VGT6이고 편의를 위해 해당 MCU가 탑재된 STM32F407G-DISC1이라는 보드를 사용하였다.

- MCU와 FM 모듈은 I2C 통신으로 연결되어, MCU가 FM모듈의 주파수를 선국하기도 하고 FM 모듈의 상태를 읽기도 한다.
- MCU와 LCD 모듈은 I2C통신으로 연결이 되고, FM모듈의 상태 정보를 LCD에 실시간으로 표시한다.
- MCU와 PC는 RS232모듈을 통해 UART통신으로 연결이 되고, MCU가 FM 모듈을 제어하기 위한 정보를 PC가 MCU에게 송신한다.
- 프로그램 다운로드는 Mini usb cable을 통해 MCU가 PC로부터 프로그램을 다운로드 한다.

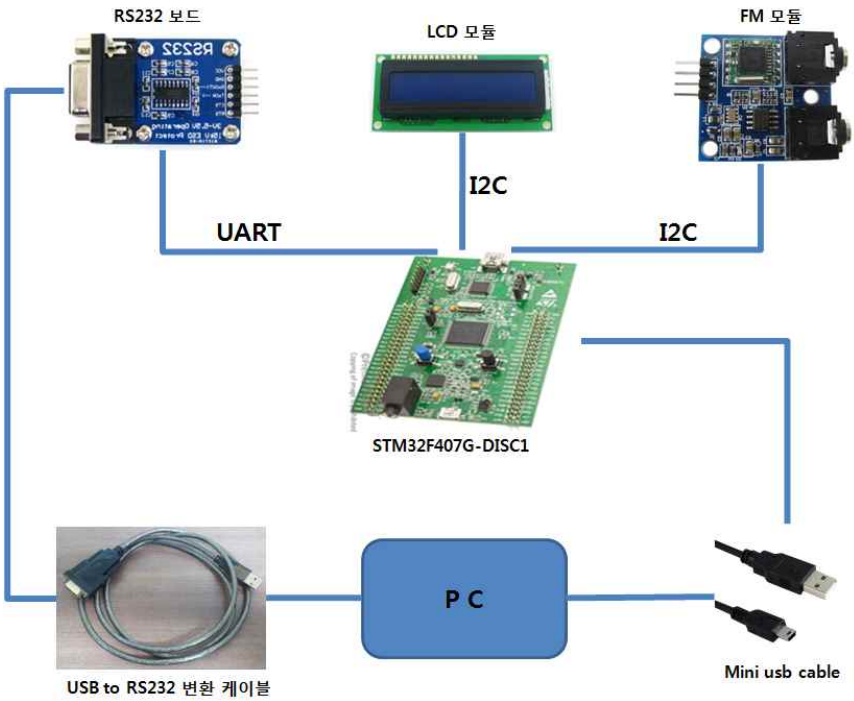

그림 1-1. 전체 시스템의 구성도

1-2. 전체 시스템의 동작 방식

- 프로그램이 시작되면 MCU는 FM 모듈의 주파수를 89.1Mhz로 선국하고, LCD에 표시한다.
- STM32F407G-DISC1 보드의 파란색 스위치를 한 번씩 누를 때마다 FM모듈의 주파수가 0.1Mhz씩 증가한다. FM모듈의 주파수 변화는 실시간으로 LCD에 표시된다.
- STM32F407G-DISC1 보드의 파란색 스위치를 누르면, 파란색 스위치 바로 옆의 LED에 불이 들어온다.
- 2초에 한 번씩 MCU가 FM 모듈로부터 데이터를 읽어서, FM 모듈이 모노 상태인지 스테레오 상태인지 검사 후 LCD에 표시한다.
- PC에서 MCU로 데이터를 송신해서 FM 모듈을 원하는 주파수로 선국할 수 있다. 예를 들어, PC에서 “fm w freq 100.0” 이라는 메시지를 송신하면 FM 모듈의 주파수는 100.0Mhz의 주파수로 선국된다.

2. 전체 시스템의 소프트웨어 구성

2-1. 20msec 루틴

- 본 설계의 핵심은 20msec루틴이다. 20msec 루틴이 실행되는 함수는 MainFunction이라는 함수이고, 코드 2-1에서와 같이 main함수의 while(1)문 안에서 수행된다. MainFunction함수는 코드 2-2와 같다.

- SysTickBuffer는 코드 2-3과 같이 TIM3_IRQHandler(인터럽트 서비스 루틴)에 들어갈 때마다 1로 변경된다. TIM3_IRQHandler는 4ms마다 수행된다.
- MainFunction함수에서 SysTickBuffer가 1이면 다시 0으로 초기화시키고, TaskSwitchBuffer변수를 하나 증가시킨다. TIM3_IRQHandler가 4ms마다 수행되기 때문에 TaskSwitchBuffer는 4ms마다 하나씩 증가한다. TaskSwitchBuffer는 0x05가 되면 0으로 초기화되고 TaskSwitchBuffer의 값에 따라 case를 나누어 각 case에 해당되는 루틴을 실행한다.
- 각 case의 루틴은 TaskSwitchBuffer가 한 주기(5회 증가)를 돌고와야 실행이된다. 따라서 각 case의 루틴은 4msec × 5 = 20msec 마다 한 번씩 실행된다. 그러므로 MainFunction함수를 20msec 루틴 함수라고 이름을 지었다.

```
void main()
{
        while (1)
        {
                MainFunction();
        }
{
```

코드 2-1. main 루틴

```
void MainFunction(void)
{
        if(SysTickBuffer == 0x01)
        {
                SysTickBuffer = 0x00;
                TaskSwitchBuffer++;

                if(TaskSwitchBuffer > 0x04)
                {
                        TaskSwitchBuffer=  0x00;
                }

                switch(TaskSwitchBuffer)
                {
                        case 0:
                                CheckRadioPower();
                                break;
                        case 1:
                                ReadKey();
                                break;
                        case 2:
                                ChangeChannel();
                                break;
                        case 3:
                                Usart2Recv();
                                break;
                        case 4:
                                break;
                        default
                                TaskSwitchBuffer=  0x00;
                                break;
                }
        }
}
```

코드 2-2. 20msec 루틴

```
void TIM3_IRQHandler(void)
{
  HAL_TIM_IRQHandler(&TimHandle);

  if(SysTickBuffer == 0)
  {
      SysTickBuffer= 1;
  }
}
```

코드 2-3. TIM3의 IRQHandler

2-1-1. 20msec 루틴 실행을 위한 타이머 초기화

- TIM3은 84Mhz로 동작하는 타이머이고, 초기화 설정은 코드 2-4와 같이 해주었다. 코드 2-4의 초기화 코드로 인해 TIM3은 4msec 마다 인터럽트를 발동하고, TIM3으로 인해 20msec 루틴을 동작하게 된다.

- __HAL_RCC_TIM3_CLK_ENABLE(); => TIM3에 클럭을 공급한다.
- TimHandle.Instance = TIM3; => 사용할 타이머를 TIM3으로 지정
- TimHandle.Init.Period = 80; => 주기를 80으로 지정
- TimHandle.Init.Prescaler = 4200; => 분주를 4200으로 지정
- TimHandle.Init.CounterMode = TIM_COUNTERMODE_DOWN; => 다운 카운터로 동작
- HAL_TIM_Base_Init(&TimHandle); => TIM을 초기화한다.
- HAL_NVIC_SetPriority(TIM3_IRQn, 4, 0); => TIM3의 우선순위를 4로 지정
- HAL_NVIC_EnableIRQ(TIM3_IRQn); => TIM3의 인터럽트 허용

```
void Init_Timer3(void)
{
	__HAL_RCC_TIM3_CLK_ENABLE();

	TimHandle.Instance = TIM3;
	TimHandle.Init.Period = 80;
	TimHandle.Init.Prescaler = 4200;
	TimHandle.Init.ClockDivision = 0;
	TimHandle.Init.CounterMode = TIM_COUNTERMODE_DOWN;

	HAL_TIM_Base_Init(&TimHandle);

	if(HAL_TIM_Base_Start_IT(&TimHandle) != HAL_OK)
	{
		Error_Handler();
	}

	HAL_NVIC_SetPriority(TIM3_IRQn, 4, 0);
	HAL_NVIC_EnableIRQ(TIM3_IRQn);
}
```

코드 2-4. Timer3의 초기설정

2-2. ReadKey 함수

- ReadKey 함수는 STM32F407G-DISC1 보드의 스위치가 눌렸는지 확인하는 함수이다. 스위치는 그림 2-1과 같고, 회로도는 그림2-2와 같다. 코드 2-5를 보면, 2-1장에서 설명했듯이 ReadKey 함수는 20msec 마다 실행이 된다. 즉, 20msec 마다 스위치의 상태를 확인하는 폴링 방식이다. 스위치 입력 부분을 외부 인터럽트 방식으로 구현하지 않고 폴링 방식으로 구현한 이유는 외부 인터럽트 방식으로 구현하면 채터링 현상이 발생하기 때문이다.

- ReadKey함수가 실행되면 포트 A의 0번 핀을 읽어서 ReadKey 변수에 저장한다. ReadKey = 1이면 스위치가 눌린 상태이다. 스위치가 눌리면 포트D의 12번 핀을 SET 시킨다. (스위치 옆의 LED를 ON시킴)
- 스위치가 눌리면 KeyOn = 1 로 만든다. KeyOn 가 1이면 스위치가 눌린상태이므로 FM 모듈의 채널을 변경할 때 이용한다.(스위치가 눌릴 때마다 FM 모듈의 주파수 채널을 0.1Mhz씩 증가시킴)
- 스위치가 눌리지 않으면 포트D의 12번 핀을 RESET 시킨다. (스위치 옆의 LED를 OFF시킴)

그림 2-1.
STM32F407G-DISC1
보드의 스위치

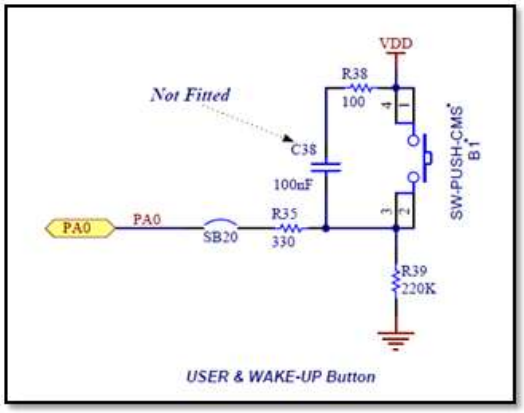

그림 2-2. MCU와 스위치의 결선도

```
void MainFunction(void)
{
	if(SysTickBuffer == 0x01)
	{
		SysTickBuffer = 0x00;
		TaskSwitchBuffer++;

		if(TaskSwitchBuffer > 0x04)
		{
			TaskSwitchBuffer= 0x00;
		}

		switch(TaskSwitchBuffer)
		{
			case 0:
				CheckRadioPower();
				break;
			case 1:
				ReadKey();
				break;
			case 2:
				ChangeChannel();
				break;
			case 3:
				Usart2Recv();
				break;
			case 4:
				break;
			default
				TaskSwitchBuffer= 0x00;
				break;
		}
	}
}
```

코드 2-5. 전체 프로그램에서 스위치 관련 함수 부분

```
void ReadKey(void)
{
	uint8_t ReadKey;

	ReadKey= HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0);
	if(ReadKey == 1)
	{
		HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);
		KeyOn= 1;
	}
	else
	{
		HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);

	}
}
```

코드 2-6. ReadKey 함수

3. MCU와 디바이스의 인터페이스

3-1. MCU와 FM 모듈의 인터페이스

3-1-1. 하드웨어 구성도

- MCU와 FM 모듈의 인터페이스 방식은 I2C통신을 사용하였다. MCU와 FM 모듈의 인터페이스 부분은 그림 3-1의 빨간색 사각형으로 표시된 부분에 해당되고, MCU와 FM 모듈의 결선도는 그림 3-2와 같다.

- MCU의 PD10 핀은 SCL로 동기식 통신을 위한 시리얼 클럭선이다.
- MCU의 PD11 핀은 SDA로 데이터를 직렬로 전송하기 위한 핀이다.
- VDD는 데이터 시트의 허용범위가 2.5V~5V 이므로 3.3V를 사용하였다.

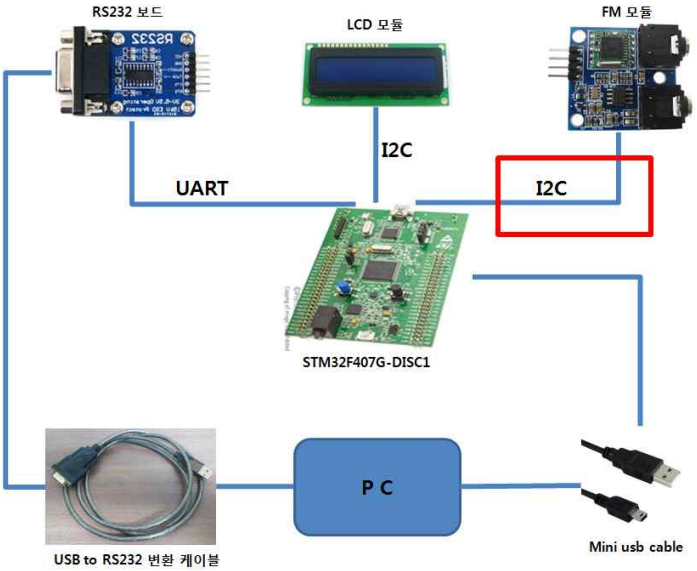

그림 3-1. 전체 시스템 구성도에서
MCU와 FM 모듈의 인터페이스 부분

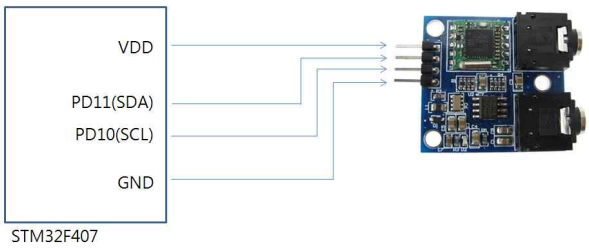

그림 3-2. MCU와 FM 모듈의 결선도

3-1-2. 동작 방식

- 전체 프로그램(MainFunction)에서 MCU와 FM모듈의 인터페이스 부분은 코드 3-1과 같이 CheckRadioPower 함수와 ChangeChannel 함수이다. 각각 20msec마다 수행된다. CheckRadioPower 함수는 코드 3-2와 같다.

- CheckRadioPower 함수에서 수행되는 일은, CheckRadioPowerBuff 변수가 100이 될 때마다 수행되기 때문에 결국 20msec × 100 = 2sec 에 의해 2초마다 수행된다. 2초마다 FM 모듈의 현재 상태가 모노인지 스테레오인지 판별해서 LCD에 표시한다.

- ChangeChannel 함수는 코드 3-3과 같다.

- 스위치가 눌리면 FM 모듈의 채널 주파수를 0.1Mhz 증가시켜서 모노 방식으로 선국한다. 또한 LCD에도 FM 모듈의 현재 주파수를 표시한다. 주파수가 108.0Mhz보다 커지게되면 88.0Mhz로 초기화 된다.

```
void MainFunction(void)
{
        if(SysTickBuffer == 0x01)
        {
                SysTickBuffer = 0x00;
                TaskSwitchBuffer++;

                if(TaskSwitchBuffer > 0x04)
                {
                        TaskSwitchBuffer= 0x00;
                }

                switch(TaskSwitchBuffer)
                {
                        case 0:
                                CheckRadioPower();
                                break;
                        case 1:
                                ReadKey();
                                break;
                        case 2:
                                ChangeChannel();
                                break;
                        case 3:
                                Usart2Recv();
                                break;
                        case 4:
                                break;
                        default
                                TaskSwitchBuffer= 0x00;
                                break;
                }
        }
}
```

코드 3-1. 전체 프로그램에서 MCU와 FM 모듈의 인터페이스 함수 부분

```
void CheckRadioPower(void)
{
        CheckRadioPowerBuff++;
        if(CheckRadioPowerBuff > 99)
        {
                ReadRadio(0xc0, I2c1RecvBuff);
                CheckRadioPowerBuff= 0;

                WriteLCDData(DDRAM_ADDR|LCD_START_LINE1, 0);
                CursorMoveToRight(13);

                RadioStereoMono= (I2c1RecvBuff[2] & 0x80) >> 7;

                if(RadioStereoMono == 1)
                {
                        WriteLCDData('S', 1);
                }
                else
                {
                        WriteLCDData('M', 1);
                }
```

코드 3-2. CheckRadioPower 함수

```
void ChangeChannel(void)
{
        union{
                uint16_t u16tmp;
                struct{
                        uint8_t low;
                        uint8_t high;
                }byte;
        }tmp16;

        if(KeyOn == 1)
        {
                RadioChannelNumber++;
                if(RadioChannelNumber > 200)
                {
                        RadioChannelNumber= 0;
                }

                WriteLCDData(DDRAM_ADDR|LCD_START_LINE1, 0);
                WriteLcdString(PLL_HIGH_FREQ[RadioChannelNumber], 8);

                tmp16.u16tmp= PLL_HIGH_TABLE[RadioChannelNumber];

                WriteRadio(0xc0, tmp16.byte.high, tmp16.byte.low, 0xb8, 0x30, 0x00);
                //WriteRadio(0xc0, tmp16.byte.high, tmp16.byte.low, 0xb0, 0x30, 0x00);

                KeyOn= 0;
        }
}
```

코드 3-3. ChangeChannel 함수

3-2. MCU와 LCD 모듈의 인터페이스

3-2-1. 하드웨어 구성도

- MCU와 LCD 모듈의 인터페이스 방식은 I2C통신을 사용하였다. MCU와 LCD 모듈의 인터페이스 부분은 그림 3-3의 빨간색 사각형으로 표시된 부분에 해당되고, MCU와 LCD 모듈의 결선도는 그림 3-4와 같다.

- 7~14번 핀은 LCD 모듈의 데이터 버스이다.
- LCD 모듈의 RS(4번 핀)는 LCD 모듈의 7~14번 핀으로 명령어가 입력될지, 데이터가 입력될지 결정하는 핀이다.
- LCD 모듈의 R/W(5번 핀)은 LCD 모듈에 WRITE할지, READ할 지를 결정하는 핀이다.
- LCD 모듈의 E(6번 핀)은 LCD 모듈의 시리얼 클럭 선이다.

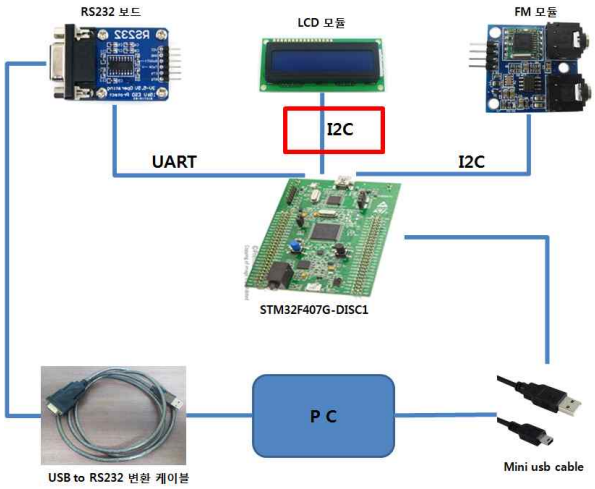

그림 3-3. 전체 시스템 구성도에서
MCU와 LCD 모듈의 인터페이스 부분

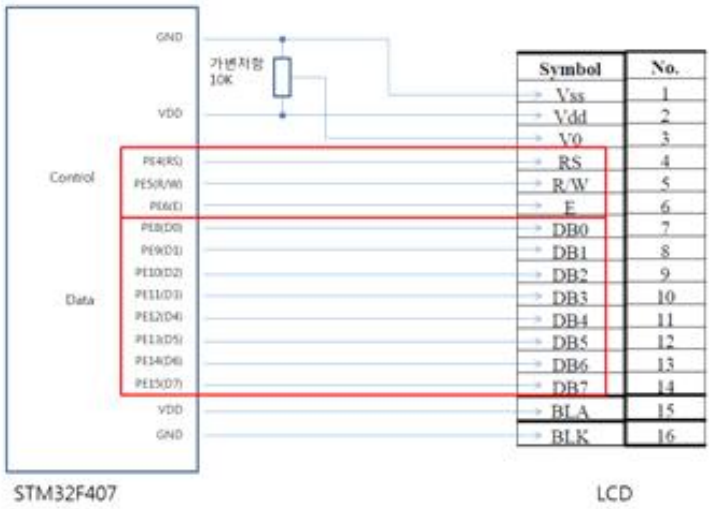

그림 3-4. MCU와 LCD 모듈의 결선도

3-2-2. LCD 모듈 구동 함수

- WriteLCDData 함수는 두 번째 인자를 통해 LCD에 명령어를 입력할지, 데이터를 입력할지 결정하고, 첫 번째 인자를 write 하는 함수이다. WriteLCDData 함수는 코드 3-4와 같다.

- LCD 가 Busy 상태인지 검사하고 Busy 상태라면 return 한다. Busy 상태가 아니면 계속 진행한다.
- 두번 째 인자인 RS가 0이면 명령어 입력이고, 1이면 데이터 입력이다.
- RW를 LOW상태로 변경함으로써 LCD 모듈에 write를 가능하게 만든다.
- LCD에 첫 번째 인자인 data를 write하고, 클럭을 토글시킨다. => LCD에 데이터가 write된다.

- WriteLCD 함수는 WriteLCDData 함수 내부에서 데이터를 write 하는 함수이고 코드 3-5와 같다.

- GPIO의 포트E의 8~15번 핀을 통해 전달받은 인자 data값을 LCD 모듈에 write한다.

```
void WriteLCDData(uint8_t data, uint8_t RS)
{
        uint8_t LCDBusy;

        LCDBusy= LCD_WaitBusy();
        if(LCDBusy == 1)
                return;

        HAL_Delay(1);
        if(RS == 0)
                LCD_RS_LOW();

        else
                LCD_RS_HIGH();

        LCD_RW_LOW();
        HAL_Delay(1);
        WriteLCD(data);
        LCD_EN_TOGGLE();
}
```

코드 3-4. WriteLCDData 함수

```
void WriteLCD(uint8_t data)
{
        union{
                uint32_t u32tmp;
                struct{
                        uint8_t b4;
                        uint8_t b3;
                        uint8_t b2;
                        uint8_t b1;
                }byte;
        }tmp32;
        tmp32.u32tmp= 0;

        tmp32.byte.b3= data;
        GPIOE->ODR= GPIOE->ODR & 0xffff00ff;
        GPIOE->ODR= GPIOE->ODR | tmp32.u32tmp;
}
```

코드 3-5. WriteLCD 함수

3-3. MCU와 RS232의 인터페이스

3-3-1. 하드웨어 구성도

- MCU와 RS232 보드의 인터페이스 방식은 UART 통신이다. MCU와 RS232 보드의 인터페이스 부분은 그림 3-5의 빨간색 사각형으로 표시된 부분에 해당되고, 결선도는 그림 3-6과 같다.

- MCU의 PA3번 핀은 MCU가 RS232 보드로부터 데이터를 수신하는 핀이다.
- MCU의 PA2번 핀은 MCU가 RS232 보드로 데이터를 송신하는 핀이다.

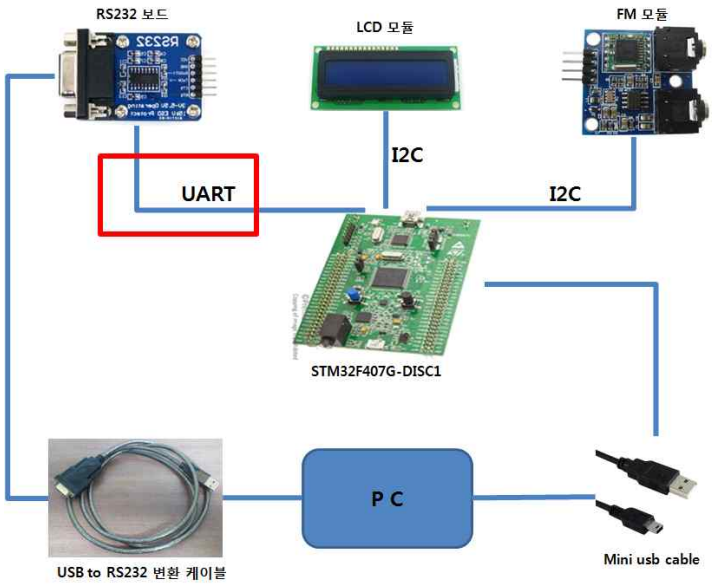

그림 3-5. 전체 시스템 구성도에서
MCU와 RS232 모듈의 인터페이스 부분

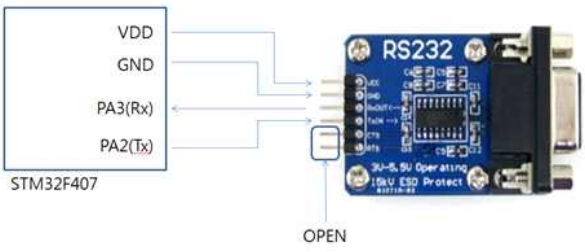

그림 3-6. MCU와 RS232 모듈의 결선도

3-3-2. 동작 방식

- 전체 프로그램에서 MCU와 RS232의 인터페이스 부분은 코드 3-6과 같이 Usart2Recv 함수이다. Usart2Recv 함수는 USART2_IRQHandler(UART2 인터럽트 서비스 루틴) 와 연동해서 PC로부터 메시지를 수신하고, 메시지 수신이 끝나면 수신한 메시지에 따라 FM 모듈의 채널 주파수를 변경한다.

- PC에서 MCU로 “fm w freq 100.0” (100은 주파수이며, 사용자가 원하는 주파수를 입력 할 수있음)이라는 메시지를 송신한다.
- MCU는 USART2_IRQHandler을 통해 PC로부터 한 번에 1바이트의 데이터를 수신하며, Usart2RxBuf 배열에 데이터를 저장한다.
- Usart2Recv 함수에서는 120msec(20msec×6) 마다 PC로부터의 모든 데이터 수신이 끝났는지를 확인한다.
- Usart2Recv 함수에서 PC로부터 한 프레임의 데이터 수신이 끝났음을 확인하면 Usart2Recv의 0~1번째 데이터가 "fm" 인지 확인한다. 0~1번째 데이터가 "fm" 이면 RunFmRadioRequest 함수를 실행한다. RunFmRadioRequest 함수는 코드 3-9와 같다.
- RunFmRadioRequest 함수는 PC 로부터 수신한 Usart2RxBuf[] 배열의 3번째 데이터가 "w" 인지 확인한다. 3번째 데이터가 "w"가 맞으면 5~8번째 데이터가 “freq” 인지 확인한다.
- Usart2RxBuf[] 배열의 5~8번째 데이터가 "freq" 이라면 FM 모듈의 채널 주파수를 100.0Mhz(사용자가 입력한 주파수) 로 선국한다.

```
void MainFunction(void)
{
	if(SysTickBuffer == 0x01)
	{
		SysTickBuffer = 0x00;
		TaskSwitchBuffer++;

		if(TaskSwitchBuffer > 0x04)
		{
			TaskSwitchBuffer= 0x00;
		}

		switch(TaskSwitchBuffer)
		{
			case 0:
				CheckRadioPower();
				break;
			case 1:
				ReadKey();
				break;
			case 2:
				ChangeChannel();
				break;
			case 3:
				Usart2Recv();
				break;
			case 4:
				break;
			default
				TaskSwitchBuffer= 0x00;
				break;
		}
	}
}
```

코드 3-6. 전체 프로그램에서 MCU와 RS232 보드의 인터페이스 부분

```
void Usart2Recv(void)
{
	uint8_t result;

	if(RecvOn == 1)
	{
		Usart2RecvTimer++;
		if(Usart2RecvTimer > 6)
		{
			RecvOn= 0;
			Usart2RecvTimer= 0;
			Usart2RecvIndex= 0;

			result= CheckTwoString(&Usart2RxBuf[0], "fm", 2);
			if(result == 1)
			{
				RunFmRadioRequest();
			}
		}
	}
}
```

코드 3-7. Usart2Recv 함수

```
void USART2_IRQHandler(void)
{
	unsigned char flag;
	HAL_StatusTypeDef Recv;

	flag= __HAL_UART_GET_FLAG(&Usart_2_Com_Handle, UART_FLAG_ORE|UART_FLAG_NE|UART_FLAG_FE|UART_FLAG_PE);
	if(flag == 0)
	{
		if(Usart2RecvIndex == 0)
		RecvOn= 1;

		Recv= HAL_UART_Receive(&Usart_2_Com_Handle, &Usart2RxBuf[Usart2RecvIndex], 1, 1);
		if(Recv == HAL_OK)
		{
			Usart2RecvTimer= 0;
			Usart2RecvIndex++;
		}
		}
	else
	{
		__HAL_UART_CLEAR_FLAG(&Usart_2_Com_Handle, UART_FLAG_ORE|UART_FLAG_NE|UART_FLAG_FE|UART_FLAG_PE);
	}
}
```

코드 3-8. USART2의 인터럽트 서비스 루틴

```
void RunFmRadioRequest(void)
{
	uint8_t result, i;

	if(CheckTwoString(&Usart2RxBuf[3], "w", 1))
	{
		if(CheckTwoString(&Usart2RxBuf[5], "freq", 4))
		{
			for(i=0; i<120; i++)
			{
				if(CheckTwoString(&Usart2RxBuf[9], PLL_HIGH_FREQ[i], 5))
				{
					KeyOn= 1;
					RadioChannelNumber= i-1;
					ChangeChannel();
					break;
				}
			}

			for(i=120; i<201; i++)
			{
				if(CheckTwoString(&Usart2RxBuf[10], PLL_HIGH_FREQ[i], 5))
				{
					KeyOn= 1;
					RadioChannelNumber= i-1;
					ChangeChannel();
					break;
				}
			}
		}
	}
}
```

코드 3-9. RunFmRadioRequest 함수

4. 고 찰

4-1. 본 설계의 핵심 개념

- 20msec 루틴을 이용한 전체 시스템의 소프트웨어 설계
- 스위치 입력 시 폴링방식을 이용한 채터링 방지
- I2C 통신
- UART 통신

4-2. 설계 과정 중에 발생한 문제 및 느낀점

- 설계 진행 초기에 HAL Drivers 라이브러리를 사용 방법을 몰라서 MCU의 Peripheral에 대한 라이브러리 사용법을 익히는 데 시간이 많이 걸렸다.
- I2C 통신으로 FM 모듈을 제어할 때, Ack 신호를 처리하지 않아서 통신이 안되는 상황이 발생했다. 처음에는 문제점이 무엇인지 몰랐지만 데이터 시트를 꼼꼼하게 읽으면서 Ack 신호를 꼭 처리해 주어야 한다는 것을 알게 되었다.
- MCU와 FM 모듈, LCD 모듈의 인터페이스 부분을 설계할 때, 해당 모듈의 데이터 시트에서 어느 부분부터 읽어야 할지 감이 안잡혔다. 그래서 무작정 처음부터 읽어봤지만 너무 시간이 오래 걸리고 스스로 지쳤다. 따라서 MCU와 외부 디바이스의 인터페이스를 설계할 때, 디바이스의 데이터시트에서 필요한 부분을 빠르게 찾아내는 능력이 필요하다는 것을 느꼈다. MCU와 디바이스의 인터페이스에서 사용하는 통신, 필요한 원리가 무엇인지 파악하고, 데이터 시트에서 그 부분부터 보는 것이 시간을 절약하는 방법이라고 생각한다.
- 디바이스의 Vdd,Vcc 허용 범위 값을 반드시 지키는 것이 중요하다고 생각한다. 본 설계에서 사용한 모든 디바이스들의 Vcc는 모두 3.3V이하부터 5V 이상까지 동작해서 주의하지 않아도 되었지만 만약 데이터시트에서 명시된 Vcc 허용 범위 값을 안지킨다면 디바이스가 소손될 가능성이 있다.