**마이크로프로세서 기말프로젝트**

20171533 박성준

20171591 최성우

1. **요약**

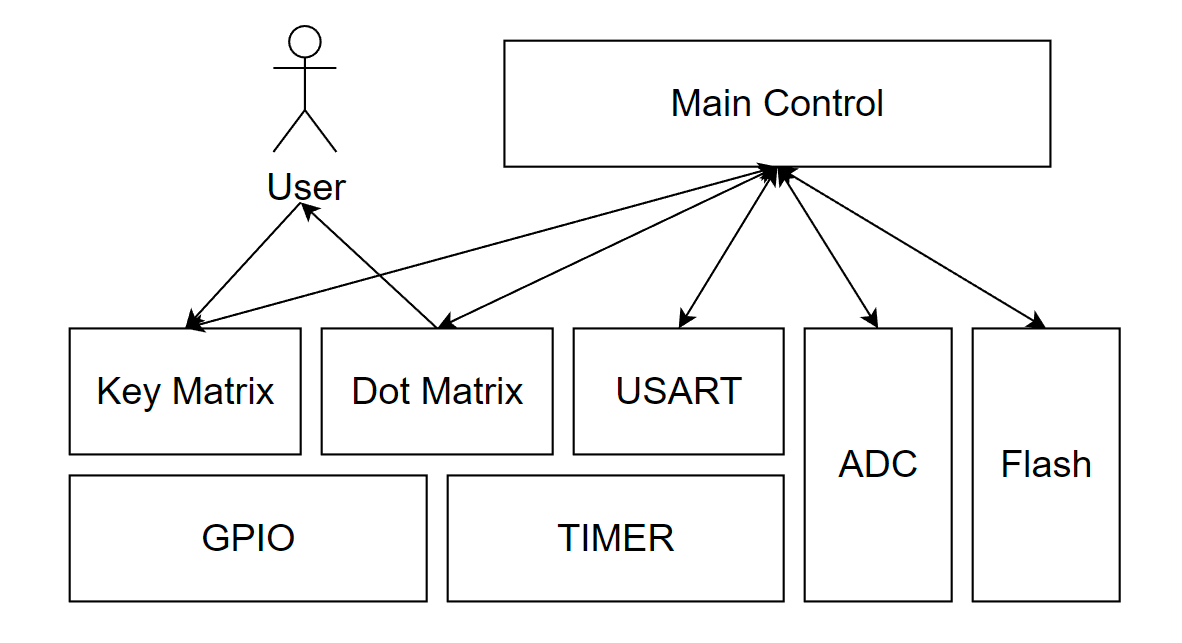
이 설계는 STM32 Nucleo-F103RB board (이하 ‘board’)를 이용해서 시계를 구현하는 것으로, 두 개의 보드를 이용해 각 입력과 출력을 담당하게 하되, 둘 사이의 데이터 송수신으로 그 기능을 구현하는 것이었다. 이를 위해서 특히 interrupt와 이를 제어하는 timer의 정교한 제어를 통해 각 procedure가 간섭없이 수행되도록 하였다. 더하여, modular한 설계를 통해 각 Hardware를 추상화하고, 이를 state단위로 제어함에 따라, 그 확장성과 코드 구성의 용이성, 안정성을 기하였다. 이러한 과정을 통해서 구현한 본 설계 결과는 의도한 바에 따라 잘 동작하였다.

1. **System architecture**

이 설계는 그 use case와 hardware specification에 따라, 적절한 architecture를 구성하고, 이에 의하여 이루어졌다. 우선, 각 하드웨어를 추상화하기 위한 모듈과 그 포함 구조를 구체화하였다. 그 다음, 이 응용에서 사용하는 정보가 어떠한 경로를 통해서 이동하는지를 명시하여 그 모듈의 구성에 참고하였다. 마지막으로, 최종적인 시스템의 제어는 state단위로 제어하기 위해 이를 서술하는 diagram을 그렸다.

* 1. 모듈 구조

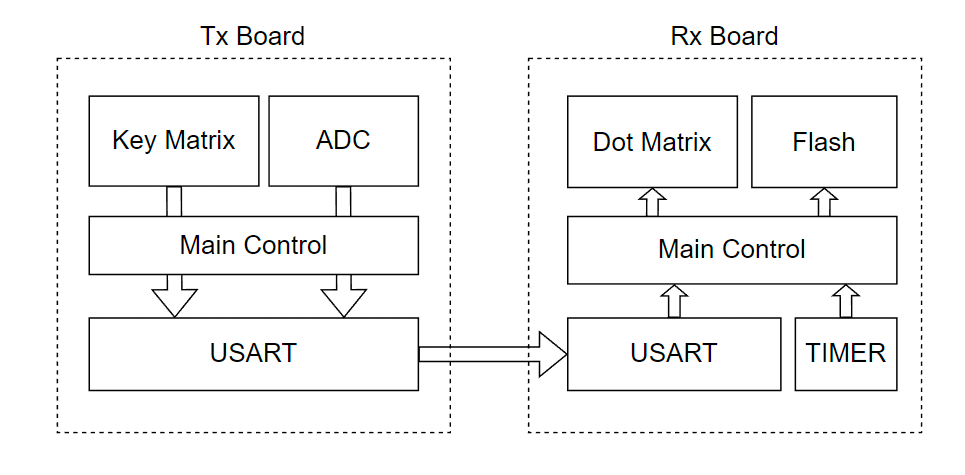
이 설계에서 사용한 전반적인 module구조는 다음과 같다. 이와 같이, 각 hardware를 적절한 수준으로 추상화하였으며, main control (주로 main함수에서 수행함)이 이들을 비교적 단순하게 제어할 수 있도록 하였다.



* 1. 데이터 흐름 구조

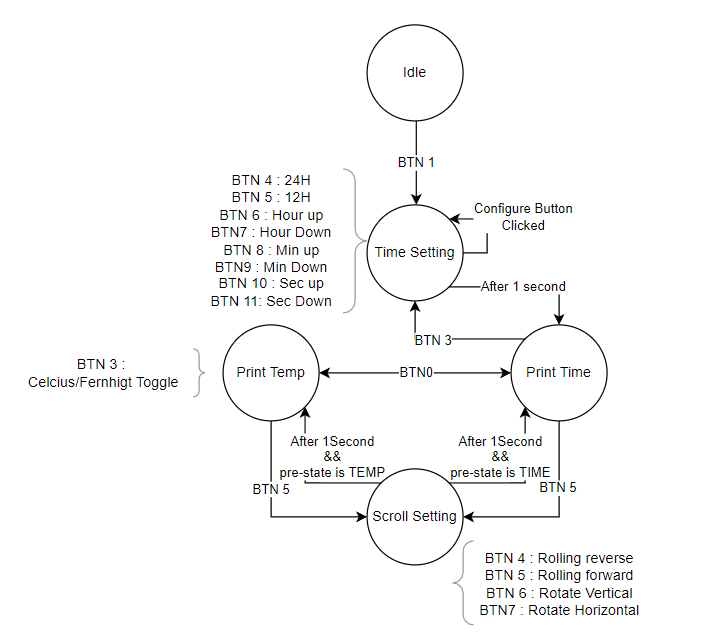
이 설계에서는 입력을 위한 board가 사용자의 key 입력, 측정한 온도 등을 적절히 조합하여 출력 board에 이를 전송하는 구조로 그 기능이 구현되었다. 따라서, 각 모듈 사이에서 이들 정보가 이동하는 flow를 구체화하였다. 이를 통해서 각 모듈에 필수적으로 구현되어야 하는 data interface를 식별하였다.

단, 구현의 용이성을 위하여, 송신단에서는 입력이 있는 경우에 즉시 그 정보를 송신하고, 수신단에서는 수신된 정보를 적절한 위치에 저장한 뒤, 주기적으로 그 정보를 참조하여 출력에 반영하는 방법을 사용하였다.



* 1. main함수 state diagram구조 (출력 board)

위에서 식별한 각 module에 대해서 각각의 구현을 진행한 뒤, 이를 통합하는 과정을 진행하였다. 이 때, 그 통합은 다음과 같은 state diagram에 의함으로써, 상태기반의 제어프로그램을 구현하였다.



1. **모듈 별 구현 방법과 주요 함수**
   1. GPIO
      1. 구현 방법

이 코드는 board에서 제공하는 GPIO의 설정과 그 값의 입출력을 단순하게 하기 위하여 구성되었다. 단순하게 열거형으로 입력 받은 pin의 번호를 통해 해당 pin의 구성 설정을 수행하고, 해당 GPIO의 ODR에 접근하여 입출력을 담당한다.

특히, 이를 위해서 적절한 수의 GPIO Pin을 묶은 객체를 ‘GPIO Port’라 하였다. 그리고 이에 관련된 정보를 저장하기 위한 구조체를 이용해서 통합적으로 접근하고 관리할 수 있도록 하였다.

* + 1. 주요 함수
       1. write\_val\_to\_GPIO / get\_val\_from\_GPIO

이 함수는 인자로 주어진 GPIO port에 어떠한 값을 쓰거나, 읽는 작업을 수행한다. 그 pin의 번호가 GPIO port자료구조에 모두 포함되어 있음에 따라, 이러한 작업이 수행 가능하였다. 한편, 쓰고자 하는 값이 valid한지의 여부 등도 검사하는 기능을 지닌다.

* 1. TIMER
     1. 구현 방법

이 모듈은 board에 제공된 4개의 타이머를 관리하는 모듈로, 4개의 타이머를 모두 사용하며, 각각의 목적에 맞게 사용한다.

Timer1은 시계 본연의 기능을 충실히 수행할 수 있도록, 오직 시간의 흐름만을 확인한다. 이를 통해서 타 동작과의 간섭을 최소화하였고, 시간을 정확하게 측정할 수 있도록 하였다.

Timer2는 일반목적 타이머로 사용하였다. 타이머 모듈에는 함수 포인터 배열과 실행 주기 배열을 선언하였고, Timer2는 매 2ms마다 interrupt를 발생시킨다. 그 ISR에서는 상기한 실행주기 배열에 따라 연결된 함수를 수행한다. 예를 들면 아래의 표와 자료구조를 유지하여, 동작한다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 인덱스 | 함수 포인터 | 실행 주기(tick) | 최근실행 이후 Tick (Tick\_cnt) | 동작 |
| 1 | NULL | - |  |  |
| 2 | Flash\_display | 100 | 100 | Tick\_cnt초기화 후 Flash\_display실행 |
| 3 | NULL | - |  |  |
| 4 | Usart\_send | 500 | 125 | Tick\_cnt++ |
| 5 | Print\_time | 300 | 234 | Tick\_cnt++ |
| … | … | … | … | … |

위의 표를 보면, 최근 수행 이후, 100tick이 흘러 그 수행 주기가 도래한 함수가 호출된다. 이 과정은 매 Timer2의 ISR에서 수행된다. 따라서, 유저는 호출을 원하는 함수의 포인터와 그 실행주기를 Timer module에 제시하는 것 만으로 그 주기적 실행이 가능하도록 하여 일반목적 타이머를 구현하였다.

Timer3,4는 dedicated 된 목적을 갖는 타이머이다. 위의 일반목적 타이머 설계는 그 사용이 용이하다는 높은 장점과, run-time configurability를 보장하지만, 그 실행 속도가 느린 단점이 있다. 사용자는 일반적으로 100ms정도의 지연을 감수할 수 있지만, 시각적인 UI는 10ms정도의 지연만으로도 큰 불편을 느낄 수 있다. 따라서, 시각적 표현을 위한 동작 (dot matrix row scan / rolling)은 이와 같이 dedicated된 타이머를 사용하여 충분한 동작속도를 확보하고자 하였다. 한편, 최대한 높은 반응성을 보이기 위하여 key matrix row scan에도 본 타이머를 사용하였다.

* + 1. 주요 함수
       1. now\_time

이 함수는 시스템이 시작되고 난 뒤 흐른 시간을 출력하는 함수로, Timer1에 의해서 계산된 정확한 시간을 반환한다. 비록, Timer1의 ISR에서 시계의 정보를 유지하기 위한 동작을 수행하나, 이 함수를 이용하여 USART의 Time-out등을 구현한다.

* + - 1. request\_function\_call

이 함수는 상기한 Timer2의 함수 실행 테이블에 인수로 전달한 함수를, 인수로 전달한 시간마다 수행할 수 있도록 한다. 이를 위한 테이블의 관리 등을 수행한다.

한편, 이 함수 코드의 전문은 아래와 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* 1. USART
     1. 구현 방법

이 프로젝트에서 두 보드 간의 통신을 담당하는 이 모듈은 확장성과 안정성을 중점적으로 추구하여 설계하였다. 이를 위해서 USART는 계층적 소프트웨어 구조를 갖도록 구현하였다.

우선, 각 application에서 이 USART에 어떠한 메시지의 전송을 요청하는 인터페이스를 마련하였다. 이 application에는 key입력의 전송, 연결성 확인을 위한 응용 등이 있다.

각 application에서 전송한 메시지는 이 USART의 데이터프레임에 캡슐화 된다. 이 때, 그 데이터 프레임은 구조체의 형태로, 그 첫 멤버는 반드시 0xFFFFFFFF값을 갖는 정수형 변수로 하였다. 이는 후술할 바에 따라 데이터의 전송 시작을 알리는데 사용된다. 이렇게 데이터프레임화 된 정보는 USART에 queueing된다. 단, queueing이 되는 때에 다른 어플리케이션이 이 캡슐화를 요청하면, 기존의 동작을 ‘잠시’ 정지할 방법이 없다. Time-out에 의한 context switching을 구현하지 않았기 때문이다. 따라서, 각 어플리케이션에 따른 우선순위를 정하고, 사용자 입력을 최우선 하여 queueing되도록 하였다.

그 다음에는, 상기의 queue에 데이터프레임이 있는 경우, 이를 전송한다. 이 때에는 데이터프레임 구조체를 char형 변수로 cast하여 전송한다. 이때, 타 interrupt에 의해서 그 전송이 preemption되어도 문제가 없도록 race condition을 해소하는 방안을 마련하였다.

이를 수신 측에서 받은 뒤, 다시 데이터 프레임을 복원한다. 이때, 처음의 0xFFFFFFFF를 감지하는데, 이를 통해서 송신 측의 오류 등으로 전송이 intercept된 경우에도 안정적인 동작을 보이도록 하였다. 또한, 어떤 데이터프레임이 수신되기 시작하고 나서 일정 시간동안 그 수신이 끝나지 않으면(time-out) 해당 프레임을 drop하여 부적절한 정보가 처리되는 것을 방지하였다.

그리고 그 수신된 프레임의 목적 어플리케이션을 식별하고 이를 de-Multiplexing함으로써 USART의 동작이 완료된다.

한편, 연결성은 일정 주기마다 임의의 수를 전송하고 상대측은 이를 수신즉시 복창하도록 하여 구현하였다. 이 상대 측의 송신 값이 기 전송한 값과 동일한 경우 연결이 구성되었다고 판단한다.

* + 1. 주요 코드
       1. 메시지 Queueing 중, 타 작업흐름에 의한 간섭 처리

텍스트이(가) 표시된 사진

자동 생성된 설명

위와 같이, 어떤 프로그램 흐름에 의해서 이미 queueing이 수행중인 경우, 우선순위를 확인하여 queueing을 정지한다.

* + - 1. 수신단에서 타임아웃과 preamble 처리

텍스트, 실내, 스크린샷이(가) 표시된 사진

자동 생성된 설명

위와 같이, 첫 데이터 수신 이후로 일정 시간이 경과하면, 이미 수신한 모든 프레임을 버린다. 이때 타임아웃 값은 1초이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

한편, 위와 같이, 처음의 0xFF를 4번 식별한 이후에 수신 버퍼에 그 바이트 정보를 입력할 수 있도록 한다.

1. **단계별 구현 방법**
   1. Step1

우선, step1을 구현할 때에는 위의 모든 모듈을 구현한 이후에 이를 조합하는 방법을 사용하였다. 특히, 시간을 출력하기 위한 dot-matrix를 잘 제어하는 것이 핵심이었는데, 다음과 같은 코드를 이용하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이때, print\_string\_display는 주어진 문자열을 dot-matrix에 표시하는 것으로, 문자열을 적절히 조작한 뒤, 이를 rolling하는 방법으로 표출한다. 한편, 모든 폰트는 아스키코드에 의거하여 저장하였으므로, 위와 같이 별도의 변환작업을 수행하지 않아도 되도록 하였다.

* 1. Step2

그 다음 단계의 주요 설계인 통신은 다음과 같은 핵심 구문을 이용해 구현하였다.

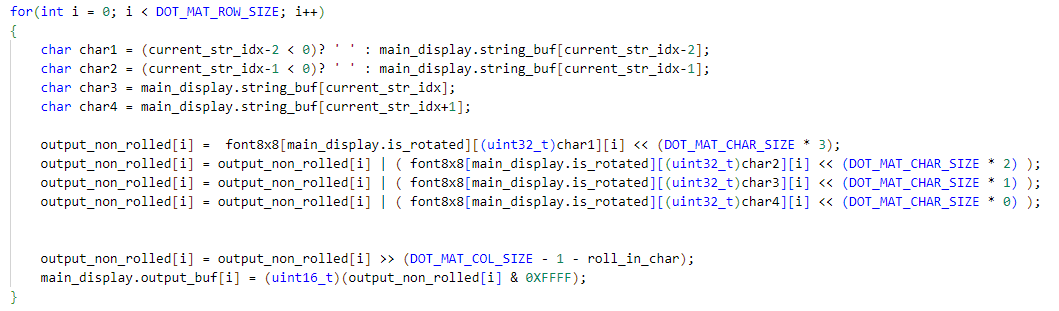
텍스트이(가) 표시된 사진

자동 생성된 설명

USART의 하위 계층에서 수신 어플리케이션과 그 payload값을 주어준 경우, 위와 같은 switch문을 이용해 적절한 동작을 수행하도록 하였다. 이 때, 온도 값은 전역변수 저장소에 저장한다. 명령은 이와 같이 저장한 다음, state machine을 수행하였다.

* 1. Step3

Dot-matrix 2개를 통한 출력이 핵심인 이 단계는 다음과 같이 구현하였다.



이는 출력 버퍼를 충분히 크게 하고, rolling counter에 따르는 적절한 bit가 저장되도록 연산을 수행함으로써 그 동작이 이루어질 수 있도록 한다.

* 1. Additional: Flash

Flash를 이용하여 24시간제 표시 여부, 화면 재생 방법 등을 저장하고, 매 최초 시작 시 이를 불러오는 방법을 사용하여, 전원공급이 일시적으로 중단되어도 이들 정보가 잘 유지될 수 있도록 하였다. 특히, 이를 위해서 플래시 sector의 lock을 해제하고 그 내용을 지우는 등의 과정이 필요한데, 이는 다음과 같은 핵심적인 코드를 이용하였다.

텍스트, 실내, 스크린샷이(가) 표시된 사진

자동 생성된 설명

이 때, 각 함수는 STM32의 표준 라이브러리에 정의된 바를 이용하였다.

1. **토의**

토의에서는 이 구현에서 주되게 고려한 문제점 혹은, 설계 자체의 문제점을 분석한다. 또한, 주된 설계 고려사항 또한 기술한다.

* 1. Race condition문제

이 구현에서는 Timer에 의한 인터럽트로 다양한 프로그램의 실행 흐름이 발생한다. 이 흐름은 다른 흐름에 대해서 pre-emption하기에, 문제가 발생한다. 기존에 화면의 점멸을 위해서 다음과 같은 코드를 사용하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Flash display는 Timer2에 의해서 주기적으로 반복되는 함수로, 불이 켜진 경우 (blinking\_on 이 1일 때) row scan을 멈추고 모든 row에 1을 쓴다. 이로써 불이 꺼져야 한다. 그러나, row scan ISR이 수행되는 중, column에 적절한 출력을 인가하고 row에 출력을 인가하기 전의 시점에 위의 루틴이 실행될 수 있다. 이 때에는 이 루틴에 의해 모든 row를 끈 뒤, 다시 row scan ISR로 복귀하면 기존에 출력하고자 한 row를 켬에 따라 불이 완전히 꺼지지 않는 문제가 있다. 이러한 사항을 고려하며, race condition이 발생하지 않도록 노력하며 프로젝트를 수행했다.

1. **결과 및 결론**

위와 같은 과정을 통하여 이 구현을 수행한 결과, step1에서 step4에 이르는 모든 동작에 대해서 적절한 결과를 얻을 수 있었다. 나아가, 이 보드의 Flash를 제어하는 동작도 정상적인 동작을 보였다. 이에 따라, 그 아키텍처의 설계와 구현이 적합함을 검증하였고, 이를 통해서 마이크로프로세서 환경에서 그 하드웨어를 적절히 제어하는 방법을 확인하였다.