MCU - RaspberryPi

1단계: 기반 환경 구축 (Raspberry Pi 5 & InfluxDB)

모든 개발의 시작은 안정적인 환경을 구축하는 것입니다.

Raspberry Pi OS 설치 및 기본 설정:

- Raspberry Pi Imager를 사용하여 최신 64비트 Raspberry Pi OS를 설치합니다.
- SSH, VNC 등 원격 접속 설정을 완료하고 네트워크에 연결합니다.
- sudo apt update && sudo apt upgrade 명령으로 시스템을 최신 상태로 업데이트합니다.
- RaspberryPi

개발 도구 설치:

- Python 개발 환경을 위해 python3-pip, python3-venv 등을 설치합니다.
- git, build-essential 등 기본적인 개발 패키지를 설치합니다.

InfluxDB 설치 및 설정:

- Raspberry Pi (ARM64)용 InfluxDB 2.x 버전을 공식 문서에 따라 설치합니다.
- influxd 서비스를 활성화하고, 웹 UI (http://<라즈베리파이_IP>:8086)에 접속하여 초기
  사용자, 조직(Organization), 버킷(Bucket)을 생성합니다.
- API 통신에 사용할 인증 토큰(Token)을 생성하고 안전한 곳에 저장합니다.
- 라즈베리파이 InfluxDB 설치 & Grafana 설치

2단계: 하드웨어 연결 및 통신 기초 테스트

소프트웨어 개발에 앞서 물리적인 연결과 가장 기본적인 통신이 정상적으로 동작하는지 확인해야
합니다.

RS-485 트랜시버 연결:

- Raspberry Pi 5의 UART(Tx/Rx) 핀에 USB to RS-485 컨버터 또는 HAT 형태의 RS-485
  트랜시버를 연결합니다.
- 컨버터의 A/B 라인을 제어 대상 MCU 측 RS-485 트랜시버의 A/B 라인과 교차 연결합니다.

기초 통신 테스트:

- pyserial 라이브러리를 설치합니다 (pip install pyserial).
- 간단한 Python 스크립트를 작성하여, 정의된 시리얼 포트(e.g., /dev/ttyUSB0)를 열고
  간단한 바이트 데이터를 MCU로 전송하고 응답을 수신하는지 확인합니다. 이 단계에서는
  복잡한 로직 없이 '신호가 오고 가는가'에만 집중합니다.

▼ Test

1. RS-485 트랜시버 연결
2. 라즈베리 파이 UART 설정
   a. RS-485 통신을 위해 라즈베리 파이의 UART0를 시리얼 콘솔이 아닌 일반 통신용으로 사용하도록 설정해야 합니다. 터미널에서 다음 명령어를 실행합니다.

   sudo raspi-config

   - Interface Options → Serial Port 로 이동합니다.
   - Would you like a login shell to be accessible over serial? 질문에 <No>를 선택하여 시리얼
     콘솔을 비활성화합니다.
   - Would you like the serial port hardware to be enabled? 질문에 <Yes>를 선택하여 하드웨어
     UART를 활성화합니다.
   - 설정을 저장하고 Finish를 선택한 후, 재부팅(prompt to reboot)하여 변경사항을 적용합니다.
3. `pyserial` 라이브러리 설치: 라즈베리 파이 터미널에서 다음 명령어를 실행하여 pyserial을 설치합니다.

pip install pyserial

4. Python 테스트 스크립트 작성:

```
https://github.com/haengmina/JNU_xslab.git
```

JNU_xslab/SmartFarmST32_local_backup/extra/script/rs485_test.py

라즈베리파이 터미널 STM32 통신 실행

python3 rs485_test.py /dev/ttyUSB0 19200 0x4653500D004C003C --read_version --read_fan_status --verbose

통신 분석 결과

1. 시리얼 포트 및 노드 선택 (성공!)
    - Opened /dev/ttyUSB0 @ 19200 8N1 (timeout=2.0s): 시리얼 포트가 올바르게 열렸습니다.
    - [NODE_SELECT_TX] write 12 bytes: 및 [RX] START found.
    - [NODE_SELECT] Node selected successfully.: 가장 중요합니다! 라즈베리 파이가 보낸 노드 선택 요청을 STM32가 정확히 수신하고 응답했습니다. 이는 물리적인 RS-485 통신 라인과 기본적인 프로토콜 계층이 정상적으로 작동한다는 의미입니다.
2. 펌웨어 버전 읽기 (부분 성공)
    - Attempting to read firmware version...
    - [FIRMWARE_VERSION_TX] write 4 bytes:
    - [RX] START found. skipped_before_start=0
    - [FIRMWARE_VERSION] Received version: : 응답은 받았지만, 버전 정보가 비어있습니다. STM32 펌웨어에서 FIRMWARE_VERSION_RESPONSE 패킷의 데이터 필드를 제대로 채우지 않았거나, 디코딩 과정에서 문제가 있을 수 있습니다.
3. 레지스터 읽기 - 팬 상태 (실패)
    - Attempting to read Fan Power Enable (Reg 0x0160)...

- [REG_READ_TX] write 8 bytes:
- [RX] timeout waiting START(0x7E). skipped=0
- [REG_READ] No response.: REG_READ 요청을 보냈지만, STM32로부터 아무런 응답을 받지 못했습니다. 이는 STM32 펌웨어가 이 명령을 제대로 인식하지 못했거나, 처리 중 오류가 발생했음을 의미합니다.

5. MCU 설정

   MCU 펌웨어 개발 목표

   - 라즈베리 파이로부터 UART(시리얼 통신)를 통해 데이터를 수신한다.
   - 수신된 데이터 뒤에 "ACK: " 라는 문자열을 붙여 응답 메시지를 만든다.
   - 만들어진 응답 메시지를 UART를 통해 라즈베리 파이로 다시 전송한다.

   개발 단계

   1단계: UART 주변장치(Peripheral) 초기화

   MCU의 main() 함수가 시작될 때, 가장 먼저 UART 통신에 필요한 하드웨어 설정을 해야 합니다. 이 과정은 보통 MCU 제조사가 제공하는 HAL(Hardware Abstraction Layer) 라이브러리를 사용하여 수행합니다.

   - 보드레이트 (Baud Rate): 라즈베리 파이와 동일하게 설정 (예: 9600)
   - 데이터 비트 (Data Bits): 보통 8비트
   - 패리티 (Parity): 보통 None (패리티 없음)
   - 스톱 비트 (Stop Bits): 보통 1비트
   - RX/TX 핀 설정: MCU의 어떤 핀을 UART의 수신(RX) 및 송신(TX) 핀으로 사용할지 설정합니다.

   2단계: 데이터 수신 처리 (인터럽트 방식 추천)

   데이터가 언제 들어올지 모르기 때문에, 수신은 인터럽트(Interrupt) 방식으로 처리하는 것이 가장 효율적입니다.

   - 수신 인터럽트 활성화: UART 설정 시, "데이터 수신 완료(Receive Data Register Not Empty)" 인터럽트를 활성화합니다.
   - 인터럽트 서비스 루틴 (ISR) 작성: 데이터가 1바이트 수신될 때마다 자동으로 호출되는

함수입니다. 이 함수 안에서는 다음 작업을 수행합니다.

1. 수신된 1바이트 데이터를 UART 데이터 레지스터에서 읽어옵니다.
2. 미리 준비된 버퍼(배열)에 차곡차곡 저장합니다.
3. 메시지의 끝을 나타내는 문자(예: 개행 문자 \n)가 수신되면, "메시지 수신 완료"
   플래그(flag)를 true로 설정합니다.

3단계: 메인 루프(Main Loop)에서의 처리

MCU의 메인 루프(while(1) 또는 loop())는 계속해서 반복 실행되며 다음과 같은 작업을
수행합니다.

1. "메시지 수신 완료" 플래그가 true인지 확인합니다.
2. 플래그가 true라면,
   - 응답 메시지를 만듭니다. (예: "ACK: " + 수신된 메시지)
   - UART 송신 함수를 호출하여 응답 메시지를 라즈베리 파이로 전송합니다.
   - 수신 버퍼를 초기화하고, "메시지 수신 완료" 플래그를 다시 false로 설정하여 다음
     메시지를 기다립니다.

3단계: 데이터 프로토콜 정의 및 파서(Parser) 구현

안정적인 통신을 위해 명확한 규칙, 즉 프로토콜을 정의하고 이를 처리할 소프트웨어를 구현해야
합니다.

RS-485 패킷 구조 정의:

- 센서 데이터 요청, 제어 명령, 응답 등 각 상황에 맞는 패킷 구조를 설계합니다.
- 예: [STX] [Device_ID] [Command] [Data_Length] [Data_Payload] [CRC/Checksum] [ETX]
  - STX/ETX: 패킷의 시작과 끝을 알리는 고유 바이트

- Device_ID: 여러 장치가 연결될 경우를 대비한 주소
- Command: 데이터 요청, 팬 켜기/끄기 등 명령 구분
- CRC/Checksum: 통신 오류 검증을 위한 필수 요소

패킷 핸들러 구현:

- Python으로 패킷을 생성(요청)하고 분석(응답)하는 클래스 또는 모듈을 작성합니다. 이 핸들러는 체크섬 검증, 데이터 추출, 오류 처리 기능을 포함해야 합니다.

Update - commit : 아날로그 센서 데이터를 읽어오도록 함.

4단계: 센서 데이터 수집 및 물리량 변환

정의된 프로토콜을 사용하여 실제 센서 데이터를 주기적으로 수집하고 의미 있는 값으로 변환합니다.

데이터 수집기(Collector) 개발:

- 주기적으로 (e.g., 5초마다) MCU에 센서 데이터 요청 패킷을 전송하는 스크립트를 작성합니다.
- MCU로부터 응답 패킷을 받아 파싱하고, 원시 데이터(Raw Data)를 추출합니다.

물리량 변환 로직 구현:

- 센서 데이터시트를 참조하여, MCU에서 받은 원시 데이터(e.g., ADC 값)를 실제 물리량(온도, 습도, 조도 등)으로 변환하는 함수를 구현합니다.

update : raspberry pi / SmartFarm/collector.py 파일 생성.

▼ 첫번째 실행 로그

```
mini@mini:~/Desktop/SmartFarm $ python3 collector.py /dev/ttyUSB0
19200 0x4653500D004C003C --verbose
[21:22:38.083] Opened /dev/ttyUSB0 @ 19200 8N1 (timeout=2.0s)
[21:22:38.083] Target Node SN: 0x4653500D004C003C
[21:22:38.083] Data collection interval: 5.0s
[21:22:38.083] [NODE_SELECT_TX] write 12 bytes:
0000 7E 20 08 3C 00 4C 00 0D 50 53 46 6E ~ .<.L..PSFn
[21:22:38.083] [NODE_SELECT_TX] wrote=12 bytes (flush done)
```

[21:22:38.095] [RX] START found. skipped_before_start=0
[21:22:38.096] [NODE_SELECT] Node selected successfully.
[21:22:38.096] [REG_READ_TX] write 8 bytes:
0000 7E B0 04 00 02 02 00 CA ~.......
[21:22:38.096] [REG_READ_TX] wrote=8 bytes (flush done)
[21:22:38.108] [RX] START found. skipped_before_start=0
[21:22:38.109] [REG_READ] OK. Addr: 0x0200, Width: 2, Value: 7c02
[21:22:38.109] Temperature: 63.6 °C
[21:22:38.109] [NODE_SELECT_TX] write 12 bytes:
0000 7E 20 08 3C 00 4C 00 0D 50 53 46 6E ~ .<.L..PSFn
[21:22:38.109] [NODE_SELECT_TX] wrote=12 bytes (flush done)
[21:22:38.121] [RX] START found. skipped_before_start=0
[21:22:38.121] [NODE_SELECT] Node selected successfully.
[21:22:38.121] [REG_READ_TX] write 8 bytes:
0000 7E B0 04 40 01 02 00 89 ~..@....
[21:22:38.121] [REG_READ_TX] wrote=8 bytes (flush done)
[21:22:38.133] [RX] START found. skipped_before_start=0
[21:22:38.133] [REG_READ] OK. Addr: 0x0140, Width: 2, Value: e803
[21:22:38.133] Humidity: 100.0 %RH
[21:22:38.133] [NODE_SELECT_TX] write 12 bytes:
0000 7E 20 08 3C 00 4C 00 0D 50 53 46 6E ~ .<.L..PSFn
[21:22:38.134] [NODE_SELECT_TX] wrote=12 bytes (flush done)
[21:22:38.146] [RX] START found. skipped_before_start=0
[21:22:38.146] [NODE_SELECT] Node selected successfully.
[21:22:38.146] [REG_READ_TX] write 8 bytes:
0000 7E B0 04 60 01 01 00 AA ~..`....
[21:22:38.146] [REG_READ_TX] wrote=8 bytes (flush done)
[21:22:38.158] [RX] START found. skipped_before_start=0
[21:22:38.158] [REG_READ] OK. Addr: 0x0160, Width: 1, Value: 01
[21:22:38.158] Fan Status: ON
[21:22:38.158] [NODE_SELECT_TX] write 12 bytes:
0000 7E 20 08 3C 00 4C 00 0D 50 53 46 6E ~ .<.L..PSFn
[21:22:38.158] [NODE_SELECT_TX] wrote=12 bytes (flush done)
[21:22:38.170] [RX] START found. skipped_before_start=0
[21:22:38.170] [NODE_SELECT] Node selected successfully.
[21:22:38.170] [REG_READ_TX] write 8 bytes:
0000 7E B0 04 80 01 01 00 4A ~......J
[21:22:38.170] [REG_READ_TX] wrote=8 bytes (flush done)

[21:22:38.182] [RX] START found. skipped_before_start=0
[21:22:38.182] [REG_READ] OK. Addr: 0x0180, Width: 1, Value: 01
[21:22:38.182] Sensor Power Status: ON

...

^C[21:22:50.478] Serial port closed.
Traceback (most recent call last):
File "/home/mini/Desktop/SmartFarm/collector.py", line 299, in <module>
main()
~~~~^^
File "/home/mini/Desktop/SmartFarm/collector.py", line 287, in main
time.sleep(args.interval)
~~~~~~~~~~^^^^^^^^^^^^^^^
KeyboardInterrupt

🚨 첫번째 로그 분석, 온도, 습도 계산 로직 수정 필요.

5단계: InfluxDB 연동 및 데이터 저장

수집된 데이터를 시계열 데이터베이스에 저장하여 분석과 시각화의 기반을 마련합니다.

InfluxDB 클라이언트 설정:

- influxdb-client-python 라이브러리를 설치합니다 (pip install influxdb-client).
- 1단계에서 저장한 URL, Token, Org, Bucket 정보를 사용하여 InfluxDB에 연결하는 데이터베이스 핸들러 모듈을 작성합니다.

데이터 쓰기(Write) 로직 구현:

- 4단계에서 변환된 물리량을 InfluxDB의 Point 프로토콜 형식에 맞게 구성하여 DB에 저장합니다.
- Measurement (측정 항목, e.g., environment), Tags (데이터 분류를 위한 메타정보, e.g.,
  location=greenhouse1, sensor=temp-A), Fields (실제 측정값, e.g., temperature=25.5,
  humidity=60.1) 구조를 잘 활용하는 것이 중요합니다.

6단계: 제어 로직 구현 및 MCU 제어

수집된 데이터를 바탕으로 시스템이 자율적으로 판단하고 동작하게 만드는 핵심 단계입니다.

자동 제어 알고리즘 설계:

- InfluxDB에서 최신 센서 데이터를 조회하거나, 데이터 수집기에서 직접 값을 받아 특정 조건에 따라 제어 명령을 내리는 로직을 구현합니다.
- 예: if 최신_온도 > 28.0: 팬_켜기()

제어 명령 전송:

- 팬_켜기()와 같은 함수 내에서 3단계에서 만든 패킷 핸들러를 사용하여 MCU에 제어 (e.g.,
  팬 켜기) 명령 패킷을 RS-485를 통해 전송합니다.
- MCU는 이 패킷을 수신하여 실제 릴레이나 모터 드라이버를 제어하도록 펌웨어가 구현되어
  있어야 합니다.

7단계: 데이터 시각화 및 동작 검증

구축한 전체 시스템이 의도대로 동작하는지 눈으로 확인하고 알고리즘을 개선합니다.

InfluxDB 대시보드 생성:

- 웹 브라우저로 InfluxDB UI에 접속하여 'Boards'(대시보드) 메뉴로 이동합니다.
- 새로운 대시보드를 만들고, 센서 데이터(온도, 습도 등)와 제어 상태(팬 동작 여부 등)를 시간 축에 따라 보여주는 그래프 셀을 추가합니다.

알고리즘 검증:

- 대시보드를 통해 센서 값의 변화와 제어 시스템의 반응을 실시간으로 모니터링합니다.
- 예를 들어, 온도 그래프가 설정값(28°C)을 넘는 순간, 팬 동작 상태를 나타내는 그래프가
  0에서 1로 바뀌는지 확인함으로써 제어 알고리즘의 동작성을 직관적으로 검증할 수 있습니다.
