

Unicorn Emulator 를 이용한 Arduino 가상화

Team : UniKHUon

2018100847 컴퓨터공학과 이철한

2018102204 컴퓨터공학과 안준섭

1. 서론

1.1. 연구 배경

4차 산업혁명에 의해 산업이 변화하고 있다. 실물재화를 효율적으로 대량생산하는 두번의 변화를 거쳐, 디지털 기기를 이용한 가상재화 생산까지 산업 패러다임은 변화해왔다. 최근 정보통신 기술의 발달로 인공지능이 발전했고, 여러 산업군에 도입하려는 시도가 일어나면서 또다시 산업의 근간이 바뀌려는 시도가 일어나고 있다.

이러한 상황에 맞춰 교육계에서는 정보통신 기술에 유능한 인재를 육성하기 위해 소프트웨어 교육을 실시하고 있다. 대학교는 물론 초·중·고등학교에서도 코딩 교육을 실시하고 있는데, 그 중 아두이노를 사용한 임베디드 소프트웨어 교육도 많이 이뤄지고 있다. 값싼 아두이노 보드를 이용하여 간단한 컴퓨터 기판에 여러 장비들을 연결하여 단순한 기능을 구현할 수 있기 때문이다. 기업이나 대학에 비해 상대적으로 열악한 실습환경을 갖춘 초·중등 교육기관에서 아두이노 실습은 효과적이다.

더불어 싸고 간단한 자동화 작업에 유용하다는 특성 때문에 사물 인터넷, 로봇 및 드론, 공장 자동화 등 다양한 분야에서 쓰이고 있다.

아두이노를 사용하기 위해서는 반드시 하드웨어가 필요하다. 아두이노 컨트롤러 보드가 제어하는 다양한 작업을 프로그래밍하는 것을 목적으로 하기에, 테스트나 동작 결과를 실물로서 확인해야 한다. 그러나 최근 코로나19로 인해 비대면 일상으로 전환되면서 실질적인 아두이노 실습이 어렵게 되었다. 특히, 온라인 교육으로 전환되면서 소프트웨어 교육에 차질을 빚을 수밖에 없었고, 반도체 품귀현상으로 인해 전자제품의 가격이 올라가거나 구하기 어렵게 되었다.

따라서 아두이노를 Unicorn 에뮬레이터 위에서 가상화하여 하드웨어 구매에 들어가는 비용과 환경적 제약을 없애고, 아두이노의 온전한 동작을 보장하는 테스트 가능한 환경을 만든다.

1.2. 연구 목표

본 연구는 Unicorn 에뮬레이터 위에서 동작하는 가상화 아두이노를 제작하는 것을 최종 목표로 한다. 다음의 요구사항은 최종 목표를 위한 세부 사항이다.

첫째, PC 환경 위에서 Unicorn 에뮬레이터를 통한 가상 환경을 구축한다.

둘째, Unicorn 환경 위에서 아두이노 실습에 사용되는 기본동작을 구현한다.

셋째, GUI를 통해 아두이노와 그 주변 모듈의 상태를 확인할 수 있어야 한다.

2. 관련 연구

2.1. 아두이노

아두이노는 사용하기 쉬운 하드웨어 및 소프트웨어를 기반으로 하는 오픈 소스 전자 플랫폼이다. 다수의 스위치나 센서로부터 값을 받아들여 LED나 모터와 같은 외부 전자 장치들을 통제할 수 있다. 아두이노의 장점은 마이크로컨트롤러를 쉽게 동작 시킬 수 있다는 것이며, 모듈의 가격 저렴하고, 자체 IDE를 제공함으로써 다양한 환경에서 쉽게 프로그래밍 할 수 있다는 것이다. 대부분의 경우 8bit 기반 AVR 마이크로컨트롤러를 사용하기 때문에 복잡한 작업을 처리하기보다는 간단한 작업을 수행하는 임베디드 분야에 많이 쓰인다. 2023년 기준보드, 실드, 캐리어, 키트 및 기타 액세서리를 포함하여 약 100여 종류가 있다.

2.2. 에뮬레이터

에뮬레이터는 다른 컴퓨터 시스템의 동작을 모방할 수 있는 하드웨어나 소프트웨어를 일컫는다. 쉽게 말해, 특정 컴퓨터 시스템이 실제로 동작한 것처럼 속이는 행위이다. 에뮬레이터는 주로 소프트웨어 지원이 제한된 시스템이나, 단종된 시스템을 호스트 시스템에서 처리하기 위해 사용된다. 예를 들어, 수많은 프린터에서는 HP 프린터의 에뮬레이터가 내장되어 있는데, 이는 프린터를 지원하는 대부분의 소프트웨어가 HP 프린터를 위해 만들어졌기 때문이다. 이외에도 단종된 구형 기기에서만 동작하는 고전 게임을 돌리기 위해 사용하기도 한다. 에뮬레이터의 종류는 다음과 같이 분류할 수 있다.

- 비디오 게임 콘솔 에뮬레이터
- 하드웨어 에뮬레이터
- 네트워크 에뮬레이터

- 서버 에뮬레이터

2.3. QEMU

오픈소스로 제작된 머신 에뮬레이터이자 가상화 도구이다. 주로 시스템 에뮬레이션에 쓰이며, CPU, 메모리 등 다양한 하드웨어를 가상화 시킬 수 있다. 이 경우, QEMU는 게스트 OS로 하여금 호스트 CPU를 직접 제어할 수 있도록 프레임워크를 제공한다. 유저모드 에뮬레이션을 통해 사용할 수도 있다. 이는 어플리케이션 수준에서 지원하는 방법으로, 원하는 시스템의 동작 에뮬레이션을 자체 명령을 통해 제공한다.

QEMU의 주된 특징은 다양한 아키텍처와 다양한 OS를 지원한다는 것이다. Arm, MIPS, PPC, RISC-V, x86 등 다양한 CPU 아키텍처를 지원하고, Linux, macOS, Windows, UNIX계열 OS에서 모두 사용할 수 있다.

QEMU는 세미호스팅을 제공한다. 세미호스팅은 에뮬레이션 시스템의 동작을 코드로 디버깅하는 것으로, 보통의 경우 이러한 작업은 타겟 하드웨어 메인보드에서 진행된다. QEMU는 시스템의 에뮬레이션을 가상화 하기 때문에 시스템 디버깅에 효과적이다.

2.4. Unicorn Emulator

QEMU에 기반한 가벼운 멀티 플랫폼용 CPU 에뮬레이터이다. QEMU와 유니콘 에뮬레이터의 차이점은 다음과 같다.

첫째, QEMU에 비해 유니콘 에뮬레이터는 가볍고 빠르게 동작한다. 하드웨어 전체에 대한 에뮬레이션을 제공하는 QEMU와는 달리 유니콘 에뮬레이터는 CPU 에뮬레이션만을 지원한다. 불필요한 하드웨어 에뮬레이션 코드를 제거함으로써 CPU 에뮬레이션에만 집중할 수 있다.

둘째, 구조가 단순하며 유지보수가 용이하다. 두 에뮬레이터 모두 다양한 아키텍처를 지원한다는 특성을 지니는데, QEMU는 아키텍처 에뮬레이터들이 사용하는 공유 부분이 존재한다. 즉, 여러 아키텍처를 지원하는 에뮬레이터가 공동으로 사용하는 전역변수와 내부 공유 데이터가 존재한다. 이러한 QEMU의 구조적 특징은 불필요한 빌드 시간을 늘리고, 복잡성을 키우는 문제점을 야기한다. 유니콘 에뮬레이터는 아키텍처별로 데이터와 변수를 분리하여 문제점을 해결하였다.

셋째, 동적 fine-grained instrumentation을 통해 다양한 레벨에서 성능 측정이 가능하다. 정적 컴파일만 측정할 수 있는 QEMU와는 달리, 동적 fine-grained instrumentation을 도입하여 여러 단계에서의 성능 측정이 가능하다.

마지막으로 보안이 우수하다. QEMU에서 지적되었던 문제점 중 하나는 메모리 누수였다. 유니콘 에뮬레이터는 메모리 누수 문제를 해결했고, thread-safe하게 설계하여 취약점을 제거하였다.

이외에도 유니콘 에뮬레이터는 다양한 특징을 지닌다. C, python, java, rust 등 다양한 언어로 구현이 가능하며, 간단하고 직관적인 API를 제공한다. 또 JIT 컴파일 방식을 사용하여 높은 성능을 보여주며, GPLv2 라이선스이므로 무료로 사용하고 배포 가능하다.

2.5. 아두이노 시뮬레이터

현재까지 상용화 되어있는 아두이노 시뮬레이터 사례가 몇 가지 존재한다. Proteus, Tinkercad, Paulware Arduino Simulator, Wokwi는 대표적인 아두이노 시뮬레이터 사례이다. 이 서비스들의 특징은 다음과 같다.

- Proteus: 다양한 기능. 아두이노 외 라즈베리파이 까지도 지원. 시각화가 잘 되어있고 디버깅 가능. 테스트용에도 적합. 그러나 비쌈.
- Tinkercad: 초보자를 위한 여러 프로젝트가 존재. 쉽게 체험할 수 있음. 무료로 사용 가능. 인터넷 환경에서 사용 가능.
- Paulware Arduino Simulator: 아두이노 스케치를 테스트하고 실행할 수 있는 무료 시뮬레이터. 윈도우만 지원.
- Wokwi: 아두이노 설계 및 테스트할 수 있는 웹 서비스.

이 시뮬레이터들은 각각 한계점이 존재한다. 웹 환경에서 동작하거나, 사용할 수 있는 플랫폼이 한정적이다. 여기에 더해 디버깅과 테스트 환경을 마련하기 위해서 상당한 양의 금액을 지불해야 한다. 본 연구는 다양한 플랫폼을 지원하며, 오프라인에서도 작동할 수 있는 테스트 가능한 프로그램을 제작하여 기존 서비스들의 문제점을 해결하고자 한다.

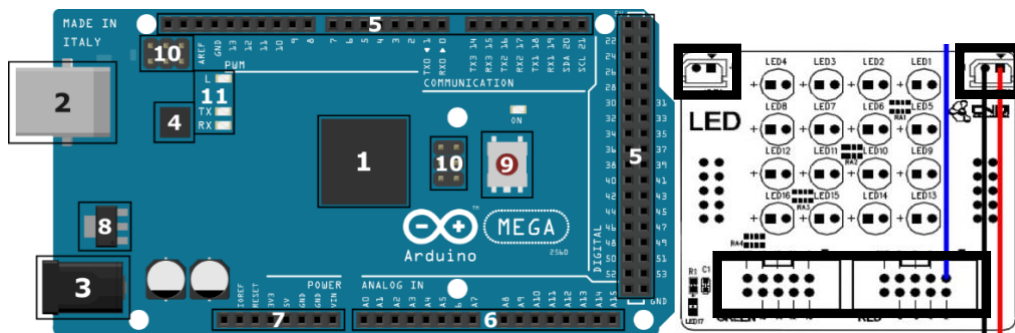
3. 프로젝트 내용

3.1. 시나리오

3.1.1. Unicorn Engine Initialization

프로그램을 시작하기 위해서, 우선 python 코드로 된 유니콘 엔진을 실행시켜서 환경을 구동한다. 유니콘 엔진 코드를 실행시키면, 어셈블리어를 바탕으로 프로그램 실행에 필요한 가상화된 레지스터, 메모리 세팅이 시작되고, 자동적으로 가상화 Arduino 화면으로 넘어간다.

3.1.2. Arduino GUI



[그림 1] 아두이노 가상화 GUI 인터페이스 화면

[그림 1]은 구현하고자 하는 아두이노 가상화 프로그램의 기본 화면이다. 해당 화면에는 좌측에 아두이노 2560 메가 본체의 인터페이스가 있고, 우측에는 4x4 LED 모듈의 인터페이스가 존재한다. 좌측 아두이노 인터페이스에서 사용자가 직접적으로 클릭할 수 있는 부분은 그림의 5,6,7번의 Pin부분이며, 우측 LED 인터페이스에서 사용자가 직접적으로 클릭할 수 있는 부분은 그림의 흑색으로 감싸진 Pin 부분이다. 좌측 아두이노에서 Pin을 하나 선택하고, 우측 LED에서 Pin을 하나 선택하게 되면, 두 Pin을 잇는 가상의 연결선이 형성되며, 시스템상으로 두 Pin이 연결된다. 모든 연결 과정이 완료되고 나면, 가상 아두이노를 동작시킬 코드를 삽입하는 창으로 넘어간다.

3.1.3. Arduino Code Input Window



[그림 2] 아두이노 코드 입력 화면

[그림 2]는 본격적으로 아두이노를 동작시킬 코드를 삽입하는 창을 대략적으로 나타낸 것이다. 화면처럼 콘솔창에 아두이노 코드를 입력하라는 메시지가 출력되고, 사용자의 입력을 기다린다. 사용자는 미리 준비된 아두이노 코드를 해당 창에 입력(혹은 붙여넣기)하고, 엔터를 누른다. 엔터를 누르는 순간부터 프로그램에서 컴파일 및 업로드를 수행하여 가상화된 아두이노에 코드의 동작을 삽입한다. 이 과정을 거친 이후, 해당 코드에 맞는 동작이 아두이노에서 실행된다.

3.2. 요구사항

3.2.1. Unicorn Engine 시작 코드에 대한 요구사항

- Unicorn Engine의 코드는 python으로 작성한다.
- Unicorn Engine 시작 코드에서는 가상화 아두이노 실행에 필요한 main 코드, 어셈블리어로 구현한 가상화 레지스터 및 메모리 세팅 관련 코드를 포함한다.
- 코드를 한번 실행하면, 메모리 및 레지스터 세팅을 자동적으로 시작한 뒤, 곧이어 가상화 아두이노가 실행되도록 자동화된 코드를 구현한다.

3.2.2. 가상 아두이노 본체에 대한 요구사항

- 아두이노의 Pin에 해당되는 부분은 사용자의 마우스 혹은 키보드 입력을 통해 선택할 수

있게끔 구현한다.

- 아두이노의 Pin 이외의 불필요한 인터페이스는 구현하지 않는다.
- 유니콘 에뮬레이터를 사용하여 아두이노의 GPIO 기능을 구현하고, 실제 아두이노의 Memory와 동일한 역할을 하는 가상 아두이노 Memory를 구현한다.
- 적절하지 않은 Pin 연결 혹은 입력 발생 시 오류 메시지를 출력하게끔 구현한다.

3.2.3. 가상 LED 모듈에 대한 요구사항

- LED 모듈의 Pin에 해당되는 부분은 사용자의 마우스 혹은 키보드 입력을 통해 선택할 수 있게끔 구현한다.
- 각 Pin에 맞는 적절한 LED가 동작할 수 있도록 Pin과 LED를 Mapping하여 구현한다.

3.2.4. Code Input System 에 대한 요구사항

- 사전에 작성된 아두이노 소스코드를 해당 콘솔창에 입력하거나, 아두이노 소스코드 파일 경로 자체를 Input으로 받는 기능을 구현한다. 해당 소스코드를 입력받은 이후, 프로그램 내에서 이를 컴파일하고 해석한 결과를 가상 아두이노에 업로드하는 기능을 구현한다.
- 적절하지 않은 파일을 입력 받았을 경우에는 오류 메시지를 출력한다.

3.3. 프로그램 설계

3.3.1. Arduino Simulator Binary 에 대한 설계

- Arduino.c, Arduino.h, User.c 3가지의 소스코드로 이루어진 예시 바이너리를 설계한다.
- Arduino.c, Arduino.h에 대한 설계는 다음과 같다. User.c에서 사용할 아두이노 함수인 digitalWrite(), pinMode(), delay()와 같은 API함수들을 직접 Arduino.c에 구현하고, Arduino.h에 선언한다. 또한, Arduino.c에는 실제 아두이노 Due 모델에서 사용하는 GPIO Pin 및 레지스터와 관련된 전역변수들을 거의 유사하게 차용하여 DDRA[32] ~ DDRD[32], PORTA[32] ~ PORTD[32], PINA[32] ~ PIND[32]를 선언하고, 해당 전역변수와 아두이노 Due의 Digital Pin Number와 Mapping하기 위한 전역배열변수 g_aArduinoDueTable[81]을 선언 및 초기화한다. 이를 통해, 사용자가 User.c에서 pinMode(13, INPUT)과 같은 코드를 작성하였을 경우, 13번 digital Pin과 연결된 가상의 레지스터에 해당하는 전역변수의 값이 설정된다. 또한, setup()과 loop()의 flow가 포함된 main 함수가 이 곳에 구현된다.

```

// DDRx - I/O 설정 레지스터
uint8_t DDRA[32];
uint8_t DDRB[32];
uint8_t DDRC[32];
uint8_t DDRD[32];

// PORTx - DDRx가 출력일 경우 HIGH or LOW 출력
uint8_t PORTA[32];
uint8_t PORTB[32];
uint8_t PORTC[32];
uint8_t PORTD[32];

// PINx - DDRx가 입력일 경우 현재 핀 상태 읽음
uint8_t PINA[32];
uint8_t PINB[32];
uint8_t PINC[32];
uint8_t PIND[32];

```

[그림 3] Arduino.c에 구현할 GPIO Pin 및 레지스터 관련 전역변수

```

int main()
{
    setup();

    while (1)
    {
        loop();
    }

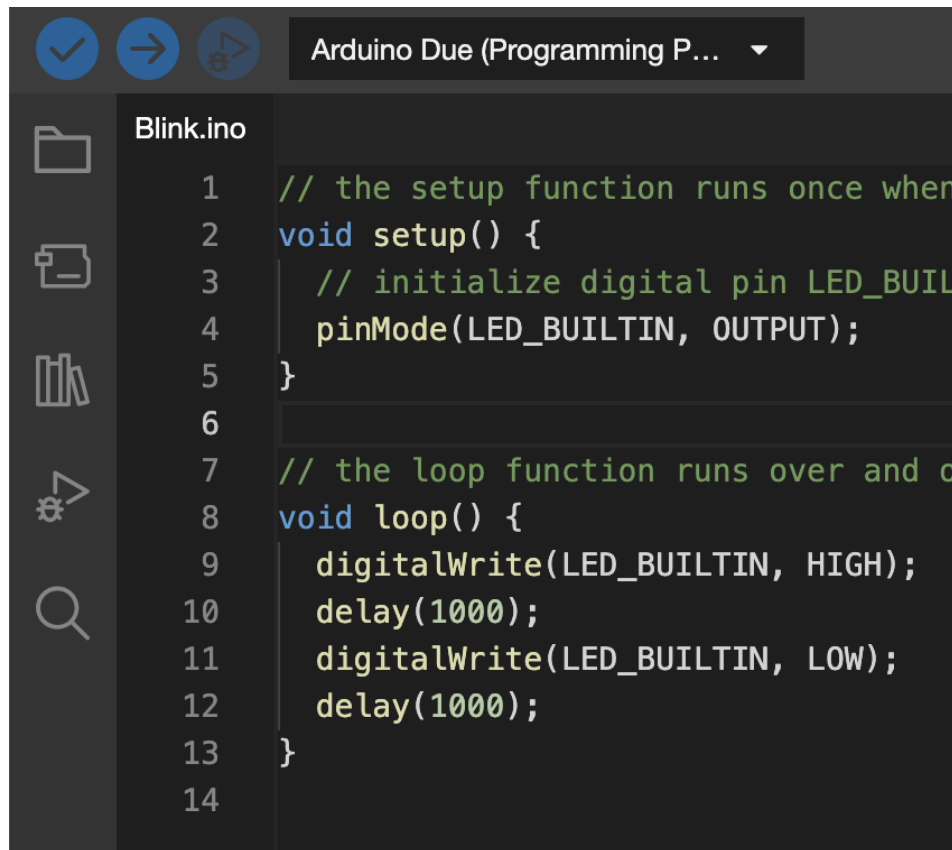
    return 0;
}

```

[그림 4] Arduino.c에 구현할 main 함수 내의 code flow

- User.c에 대한 설계는 다음과 같다. 실제 Arduino IDE에서 User가 직접 코딩하는 코드들이 그대로 이 소스코드에 구현된다. 즉, setup(), loop()와 같은 함수들이 이 소스코드에 구현되며, 위에서 설명한 Arduino.h를 소스코드 최상단에 include하여 추가적인 API 구현을 하지 않아도

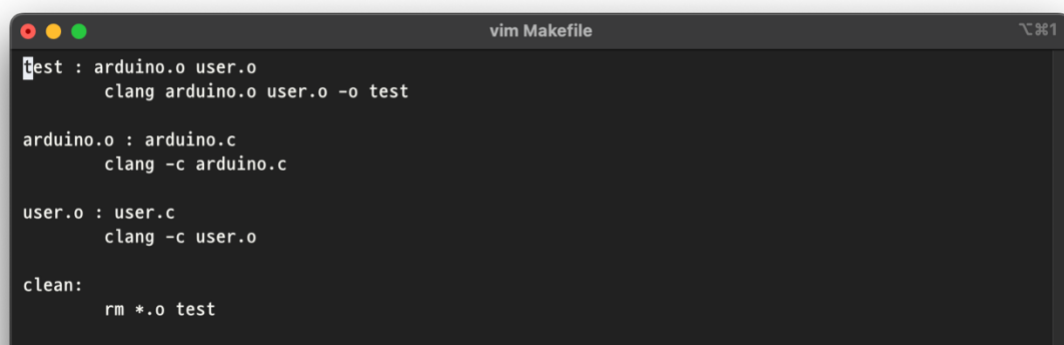
되게끔 모듈화한다. 이는 이후 gcc 등의 컴파일러를 통해 Linking하면서 하나의 binary로 합쳐진다.

A screenshot of the Arduino IDE interface. The top bar shows the 'Arduino Due (Programming P...' dropdown. The left sidebar contains icons for file explorer, search, and other IDE functions. The main editor window displays the 'Blink.ino' file with the following code:

```
1 // the setup function runs once when
2 void setup() {
3     // initialize digital pin LED_BUILT
4     pinMode(LED_BUILTIN, OUTPUT);
5 }
6
7 // the loop function runs over and o
8 void loop() {
9     digitalWrite(LED_BUILTIN, HIGH);
10    delay(1000);
11    digitalWrite(LED_BUILTIN, LOW);
12    delay(1000);
13 }
14
```

[그림 5] User.c에 구현될 코드의 Arduino IDE상의 실제 코드 모습

- 컴파일러를 통해 최종적으로 출력되는 output binary를 이용하여 Unicorn Engine의 Input code 값으로 다시 넣는다.

A screenshot of a terminal window titled 'vim Makefile'. It shows the following Makefile content:

```
test : arduino.o user.o
    clang arduino.o user.o -o test

arduino.o : arduino.c
    clang -c arduino.c

user.o : user.c
    clang -c user.o

clean:
    rm *.o test
```

[그림 6] 최종적으로 컴파일 할 output binary를 도출하는 예시

3.3.2. Unicorn Engine 의 Simulator 에 대한 설계

- 3.3.1 절에서 최종적으로 도출된 output binary는 현재 절에서 설계할 Unicorn Engine 기반의 python 코드의 Input code로 삽입한다. 개략적인 코드의 틀은 아래 그림과 같다.

```
from unicorn import *
from unicorn.arm_const import *
from binascii import *

ARM_CODE32 = b'\x00\x80\x08\x20\x2
```

[그림 7] Unicorn Engine 기반의 Python 코드에 삽입될 Input Binary Code

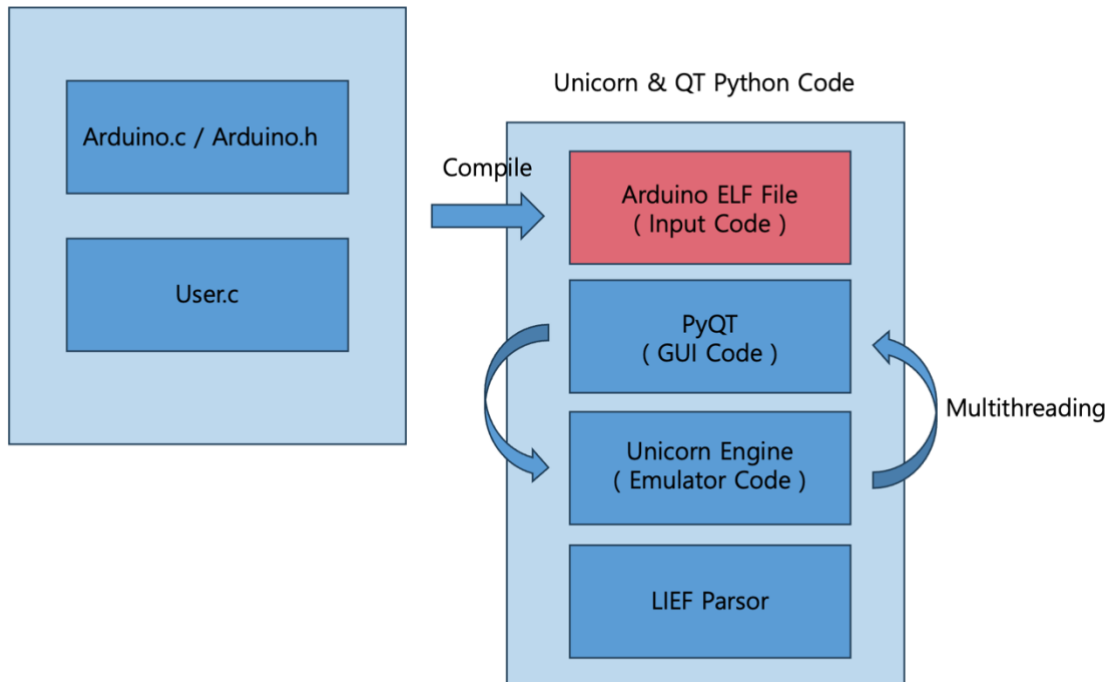
- 이후 Unicorn의 Hook 기능을 이용하여, Line by Line으로 Arduino 예시 binary code의 실행을 hooking한다. 만약, 해당 Line에서 특정 target register와 memory address 값에 변화가 발생하면, Unicorn의 Hook 기능 내에 구현할 print 기능을 통해, 특정 register 혹은 memory address 의 값이 변하였음을 사용자에게 알려준다. 이 기능을 통해 3.3.1절에서 설계하였던 Arduino Pin 입력과 출력을 감지할 수 있으며, 해당 변화를 실시간으로 사용자에게 출력할 수 있을 것이다.
- 상기한 Hook 기능을 확장하여, 사용자 친화적인 GUI 환경을 구축하는 것을 최종적인 목표로 한다. Arduino Due의 모습을 본 뜬 GUI 이미지를 구현하고, 특정 값이 변하면 해당 GUI의 특정 Pin과 연결된 LED가 발광하거나 점멸하도록 기능을 구현한다.

4. 프로젝트 결과

4.1. 연구 결과

현재까지 구현된 프로젝트의 결과물은 아래와 같다.

4.1.1. 프로젝트의 Structure/Flow Diagram



[그림 8] 프로젝트의 전체적인 흐름도 및 구조도

각 구조에 대한 자세한 설명은 이어지는 단락에서 후술한다.

4.1.2. Unicorn Engine 의 Input Code 로 사용할 Arduino 프로그램 (C 언어)

- Arduino.h, Arduino.c

: 사용자가 User.c에서 사용하려는 다양한 아두이노 API 함수 (digitalWrite, pinMode, delay 등) 의 구현부. 아두이노 Due 기반의 Pin 및 Register를 가상화한 전역 변수 (DDRX, PORTX, PINX, gArduinoDueTable) 의 선언 및 초기화 부분으로 구성되어 있다.

```
PinDescription g_aArduinoDueTable[81] =
{
    {DDRA + 8, PORTA + 8, PINA + 8},    // 0, PA8
    {DDRA + 9, PORTA + 9, PINA + 9},    // 1, PA9
    {DDRB + 25, PORTB + 25, PINB + 25},  // 2, PB25
    {DDRC + 28, PORTC + 28, PINC + 28},  // 3, PC28
    {DDRC + 26, PORTC + 26, PINC + 26},  // 4, PC26 and PA29 (disable pa29)
    {DDRC + 25, PORTC + 25, PINC + 25},  // 5, PC25
    {DDRC + 24, PORTC + 24, PINC + 24},  // 6, PC24
    {DDRC + 23, PORTC + 23, PINC + 23},  // 7, PC23
    {DDRC + 22, PORTC + 22, PINC + 22},  // 8, PC22
    {DDRC + 21, PORTC + 21, PINC + 21},  // 9, PC21
    {DDRA + 8, PORTA + 8, PINA + 28},    // 10, PA28 and PC29 (disable pc29)
    {DDRD + 7, PORTD + 7, PIND + 7},     // 11, PD7
    {DDRD + 8, PORTD + 8, PIND + 8},     // 12, PD8
    {DDRB + 27, PORTB + 27, PINB + 27},  // 13, PB27
    {DDRD + 4, PORTD + 4, PIND + 4},     // 14, PD4
    {DDRD + 5, PORTD + 5, PIND + 5},     // 15, PD5
}
```

[그림 9] Arduino Due의 구조를 반영한 Pin mapping register 전역변수의 구현부

```
void delay(uint32_t ms)
{
    if (ms == 0)
    {
        return;
    }

#ifdef __APPLE__ || defined(__linux__) || defined(__gnu_linux__) || defined(__unix__)
    sleep(ms/1000); // Linux의 sleep함수는 단위가 second이므로 주의
#endif
#ifdef _WIN32 || defined(_WIN64)
    Sleep(ms);
#endif
}

void pinMode(uint8_t pin, uint8_t mode)
{
    g_aArduinoDueTable[pin].pIORegister = mode;
    printf("Debug : pinMode()\n");
}

void digitalWrite(uint8_t pin, uint8_t val)
{
    g_aArduinoDueTable[pin].pPORT = val;
    printf("Debug : digitalWrite()\n");
    printf("%d\n", g_aArduinoDueTable[pin].pPORT);
}
```

[그림 10] Arduino.c에 구현한 실제 아두이노 API 관련 함수

- User.c

: 3.3.1절에서 기술한 것을 c코드로 구현한 코드. 사용자는 User.c 파일 안에 Arduino IDE와 동일하게 코드를 구현하기만 하고 컴파일을 진행하면, Arduino Due 기반 Simulator 역할을 하는 바이너리 파일이 만들어진다.

```
#include "arduino.h"

void setup()
{
    // LED_BUILTIN: Arduino에 탑재된 LED 제어 핀(13번 핀)
    // LED_BUILTIN을 출력으로 설정
    pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
    // LED_BUILTIN에 5V인가 - LED ON
    digitalWrite(LED_BUILTIN, HIGH);
    // 1초 지연
    delay(1000);
    // LED_BUILTIN에 0V인가 - LED OFF
    digitalWrite(LED_BUILTIN, LOW);
    // 1초 지연
    delay(1000);
}
```

[그림 11] Arduino IDE의 코드와 동일한 구조를 가진 User.c

- 컴파일 된 바이너리 파일

: gcc, clang, arm-none-eabi 등의 컴파일러로 정상적으로 컴파일하게 되면, 그 자체로도 실행이 가능한 Arduino CLI 기반의 Simulator가 만들어진다. 물론, 이 프로젝트의 최종 목표는 이 파일을 architecture-independent 하도록 Unicorn Emulator에서 구동하는 것이다. 중간 실행 결과물은 아래와 같다. delay(1000)을 통해 1초 간격으로 13번 Pin에 해당하는 전역변수 PORTB[27]에 1과 0을 반복해서 입력하는 것을 알 수 있다.

```
Debug : pinMode()
Debug : digitalWrite()
1
Debug : digitalWrite()
0
Debug : digitalWrite()
1
Debug : digitalWrite()
0
Debug : digitalWrite()
1
Debug : digitalWrite()
0
Debug : digitalWrite()
1
Debug : digitalWrite()
0
Debug : digitalWrite()
1
```

[그림 12] 컴파일하여 실행한 Arduino 실행 바이너리 파일

4.1.3. Unicorn Engine 기반의 에뮬레이터 코드 (Python)

컴파일된 Arduino 코드를 Unicorn Python 코드의 Input Code로 삽입하여 에뮬레이터를 시작한다. 4.1.1절에서 간략히 언급했듯이, 이 프로젝트에서 import한 라이브러리 및 모듈은 아래 사진과 같다. Main engine은 Unicorn이며, ELF Parsor인 LIEF 모듈을 통해 에뮬레이팅을 진행할 바이너리의 ELF 파일 정보를 추출하였으며, GUI는 PyQt 라이브러리를 사용하여 구현하였다.

```
#import unicorn
from __future__ import print_function
from unicorn import *
from capstone import *
from capstone.arm import *
from xprint import to_hex, to_x_32
from unicorn.arm_const import *
import sys
import datetime
import lief
import operator
import threading

# import qt
import os
import sys
from PyQt5.QtWidgets import *
from PyQt5 import uic
from PyQt5.QtCore import *
from PyQt5.QtGui import *
```

[그림 13] 프로젝트 내에서 import한 라이브러리 정보

필요한 라이브러리를 import한 이후, 본격적으로 에뮬레이터를 작동시키기 위해 ELF Parsor를 통해 Arduino ELF파일의 entry point와 main function 등의 address정보를 추출하여 변수에 저장하였으며, 별도의 stack pointer 주소를 설정해주어야 에뮬레이터가 정상적으로 작동하기 때문에, stack과 관련된 정보 역시 임의로 초기화하였다. 마지막으로, 이번 프로젝트에서 PyQt GUI Thread와 ITC (Inter-Thread Communication)을 하기위해 필요한 전역변수인 target_memory_value를 정의하였는데, 이 변수를 통해 원하는 메모리 주소 내의 값의 변화를 추적할 수 있다. 이에 대한 자세한 사항은 후술하도록 한다.

```
code_start_address = list(func_sort.values())[0]
emu_entry_point_address = 0x816c
emu_main_address = func_sort.get('main')
emu_length = 0x20000
STACK_ADDRESS = 0x80000000
STACK_SIZE = 0x100000

target_memory_value = 0 # Core value
```

[그림 14] 에뮬레이션에 필요한 변수들의 정보

변수 설정에 이어서, 본격적으로 Unicorn Engine을 실행하기 위해 Unicorn 상의 가상 메모리에 Code 및 Data, BSS section들을 memory mapping하고, stack pointer를 초기화하는 등의 설정을 하였다. 이렇게 필요한 모든 설정을 마친 이후, 본격적으로 emu_start()를 통해 코드 에뮬레이션을 진행하게 된다. 또한, GUI 환경에서 LED 동작을 제어하는 데에 반드시 필요한 Arduino Pin 관련 전역변수의 변화를 실시간으로 추적하기 위해, 매 instruction 실행마다 특정 memory address의 값의 변화를 추적하는 hook code를 삽입하는 hook_add()를 사용하였다. 관련된 코드는 아래 사진과 같다.

```
mu = Uc(UC_ARCH_ARM, UC_MODE_ARM)
mc = Cs(CS_ARCH_ARM, CS_MODE_ARM)

# Unicorn needs memory address at 0x0
mu.mem_map(0x0, 4 * 1024 * 1024)
mu.mem_map(STACK_ADDRESS - STACK_SIZE, STACK_SIZE)

mu.mem_write(code_start_address, ARM_CODE32)

# initialize machine registers
# stack pointer must be initialized
mu.reg_write(UC_ARM_REG_SP, STACK_ADDRESS)
mu.reg_write(UC_ARM_REG_FP, STACK_ADDRESS)

mu.hook_add(UC_HOOK_CODE, hook_code, copy_mne, begin=code_start_address, end=code_start_address + len(ARM_CODE32))
mu.emu_start(emu_entry_point_address, emu_entry_point_address + emu_length)
```

[그림 15] 에뮬레이션에 필요한 설정 코드 및 실행 코드

```
def hook_code(uc, address, size, user_data):
    target_memory_value = uc.mem_read(0x0, 1)
    global update_value
    update_value = (target_memory_value)
```

[그림 16] 특정 memory address의 값을 추적하는 hooking function

4.1.4. Unicorn Engine 기반의 에뮬레이터 코드 (Python)

코드 에뮬레이션과는 별개로, GUI를 통해 유저 친화적인 인터페이스를 출력하기 위해 PyQt를 사용하여 python 코드를 작성하였다. Unicorn 코드가 작성된 python file에 추가적으로 작성하였다. 중요한 점은, Unicorn Emulator와 PyQt GUI Program이 동시에 동작해야 하기 때문에 Multithreading 개념을 사용하여 두 코드가 독립적으로 실행되도록 하였으며, python에서 제공하는 기본 thread 기능을 사용하여 구현하였다.

```
def set_unicorn(self):  
    self.unicorn_flow = threading.Thread(target=main)  
    self.unicorn_flow.daemon = True
```

[그림 17] PyQt와는 별도로 실행하기 위해 Thread로 실행시키는 Unicorn Thread 코드

또한, Unicorn Emulator와 PyQt 간의 데이터 교환이 LED 동작 등의 행위에 필요하므로, ITC (Inter-Thread Communication)를 통해 두 Thread간의 통신이 이루어지게끔 하였다. ITC 방법으로 공유 전역변수를 이용하여 통신하도록 하였다. 전역변수는 4.1.3절에서 언급하였던 target_memory_value (update_value와 동일)를 사용하였다.

```
if prev_state_value == None:  
    prev_state_value = update_value  
else:  
    if prev_state_value == update_value:  
        continue  
    else:  
        prev_state_value = update_value  
  
# 업데이트 변수에 따른 렌더링 부분.  
for item in self.drawing_item:  
    if item.type != "WIRE":  
        item.update_texture(self.scene)  
    else:  
        item.update_scene(self.scene)  
  
QApplication.processEvents()
```

[그림 18] 전역변수를 이용해 ITC를 수행하여 GUI의 LED를 제어하는 코드

4.1.5. digitalWrite()를 통해 변하는 전역변수의 address를 찾는 과정

이 프로젝트에서 예시로 에뮬레이션하는 Arduino ELF 파일은 digitalWrite(13, HIGH)을 통해 13번 digital Pin에 해당하는 전역변수 PORTB[27]를 1로 초기화하고, delay(1000)을 통해 잠시 대기한 이후, 다시 digitalWrite(13, LOW)를 통해 PORTB[27]를 0으로 초기화하는 과정을 반복한다. 이를 GUI상의 LED에 표현하기 위해선, PyQt 모듈에서 이 전역변수의 값을 끊임없이 추적할 수 있어야 한다. 그렇기에, 해당 전역변수가 Unicorn Emulator상의 Virtual Address의 몇 번지에 위치하는지 파악해야 하므로, 이를 직접 disassembly 코드를 분석하여 확인하였다.

```
0000829c <digitalWrite>:
829c: 04 b0 2d e5    str r11, [sp, #-4]!
82a0: 00 b0 8d e2    add r11, sp, #0
82a4: 0c d0 4d e2    sub sp, sp, #12
82a8: 00 30 a0 e1    mov r3, r0
82ac: 01 20 a0 e1    mov r2, r1
82b0: 05 30 4b e5    strb r3, [r11, #-5] // 위의 pinMode와 동일
82b4: 02 30 a0 e1    mov r3, r2
82b8: 06 30 4b e5    strb r3, [r11, #-6]
82bc: 05 20 5b e5    ldrb r2, [r11, #-5]
82c0: 30 10 9f e5    ldr r1, [pc, #48] @ 0x82f8 <$d>
82c4: 02 30 a0 e1    mov r3, r2 // r3 = 13
82c8: 83 30 a0 e1    lsl r3, r3, #1
82cc: 02 30 83 e0    add r3, r3, r2
82d0: 03 31 a0 e1    lsl r3, r3, #2
82d4: 03 30 81 e0    add r3, r1, r3 // r3 = 0x188c0
82d8: 04 30 83 e2    add r3, r3, #4 // r3 = 0x188c4
82dc: 00 30 93 e5    ldr r3, [r3] //
82e0: 06 20 5b e5    ldrb r2, [r11, #-6] // r2 = 13
82e4: 00 20 c3 e5    strb r2, [r3]
82e8: 00 00 a0 e1    mov r0, r0
82ec: 00 d0 8b e2    add sp, r11, #0
82f0: 04 b0 9d e4    ldr r11, [sp], #4
82f4: 1e ff 2f e1    bx lr
```

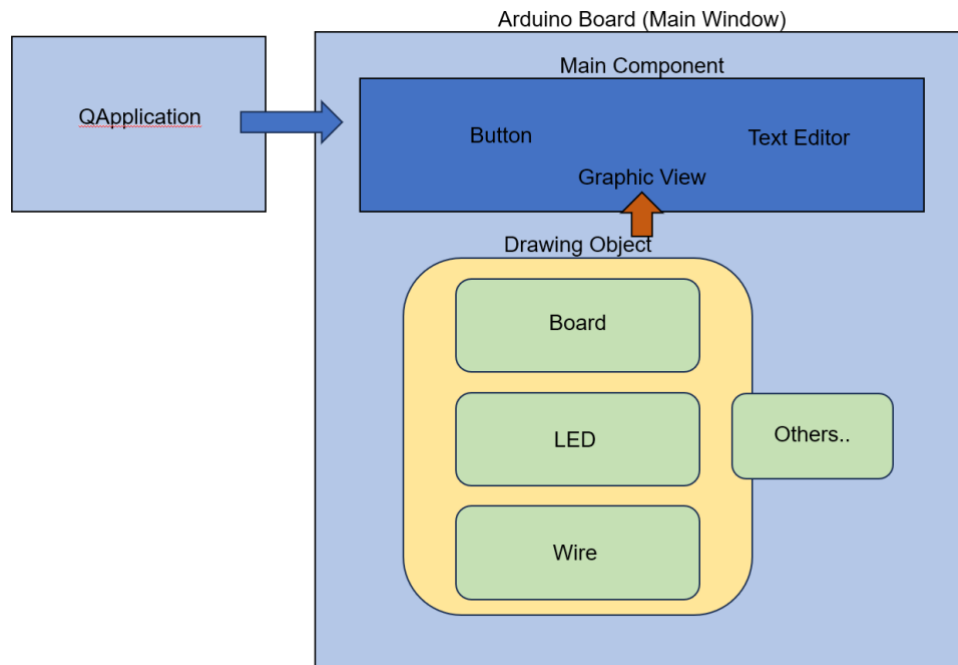
[그림 19] digitalWrite 함수의 바이너리 상의 disassemble된 어셈블리 코드

그 결과, 0x0번지에 해당 전역변수의 값이 변하는 것을 발견하였으며, 이 메모리 주소를 trace하여 상술했던 target_memory_value에 값의 변화를 대입하였다. 이 값이 변할 때마다 GUI LED가 깜빡이게 된다.

4.1.6 PyQt를 이용한 GUI 구현

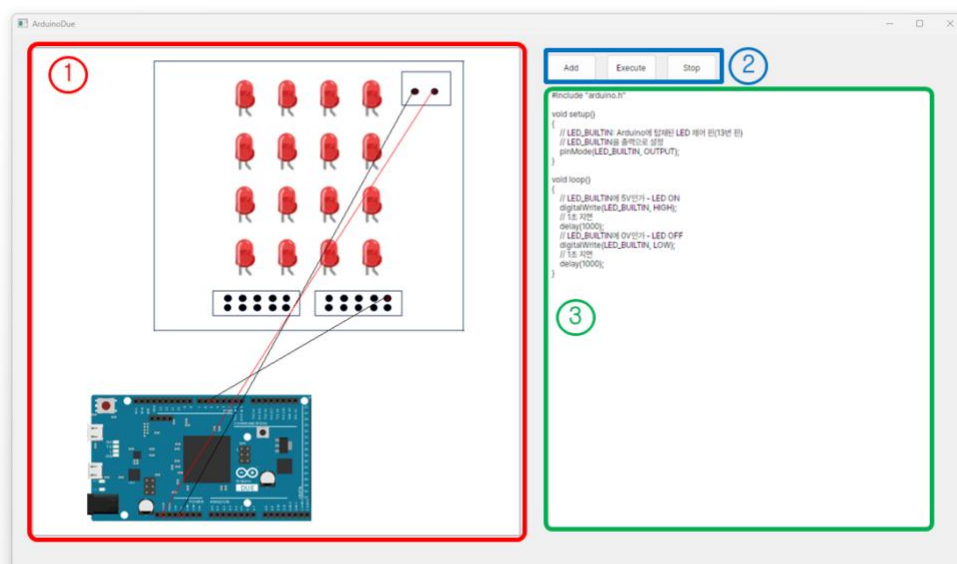
GUI 구현을 위해 Python 버전의 QT플랫폼을 이용하였다. QApplication이라는 클래스를 이용하여 실행을 제어하며, 이때 Main Window를 렌더링한다. Main Window 안에는 동작 여부를 확인하는

여러 버튼들과, 사용자의 코드를 입력 받고 실행하는 Text editor, 에뮬레이션 결과를 보여주는 Graphics View가 주 요소로 존재한다. 특히, Graphics view는 Arduino의 동작을 시각적으로 보여줘야 한다. 그 안에서 렌더링 될 여러 객체들을 정의하여 각각의 변화를 현재 장면에 추가하면, Graphics view에서 곧바로 렌더링 한다. GUI 프로그램의 구성도는 다음과 같다.



[그림 20] PyQt를 이용한 GUI 프로그램 구성도

[그림 20]에서 볼 수 있는 모든 구성 요소들을 구현한 결과는 다음과 같다.



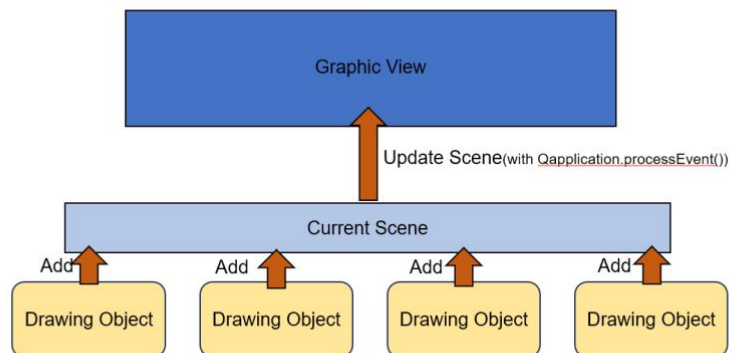
[그림 21] PyQt를 이용한 GUI 프로그램

다음의 내용은 [그림 21]에 나타난 GUI 프로그램 구성 요소에 대한 설명이다.

① Graphics View

Arduino의 현재 상태를 나타내는 부분이다. 앞서 짧게 설명했듯, 그려질 객체들을 현재 장면에 추가하고, 그 장면을 Graphics view에 적용하는 순서로 구동된다.

Drawing Object라는 기본 클래스 위에 Board, LED객체를 정의하여 Arduino update value의 변화에 대해 각기 다른 업데이트 동작을 할 수 있도록 구성하였다. 여기서 Wire의 경우 추후 기술하겠지만, 기술적인 문제로 Drawing Object가 아니라 별도의 클래스로 분류하였다. [그림 22]는 이벤트에 따른 Graphics View의 동작을 잘 설명해준다.



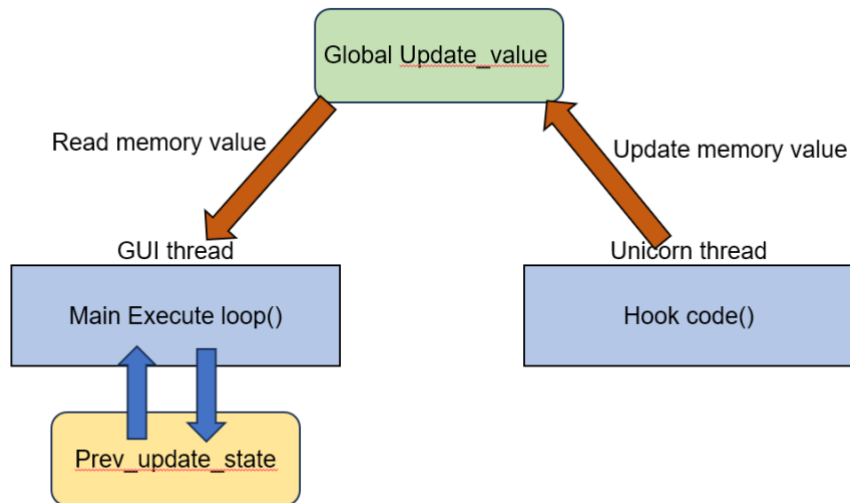
[그림 22] Graphics View의 실행 흐름도

② Buttons

세 가지 버튼이 존재하며, 각각 Add, Execute, Stop으로 이뤄져 있다. 각각의 기능은 다음과 같다.

Add 버튼은 Graphics View에 모듈을 추가하는 버튼이다. 추후 기술하겠지만, 현재 단계에서는 Unicorn Engine의 불완전성과 LED 모듈 동작만을 목적으로 동작하기에, Arduino 모듈 추가는 어렵다는 결정을 내렸다. 이 버튼 동작에 관한 동작은 차후 업데이트에서 구현될 것이다.

Execute 버튼은 [그림21]의 3번 구역의 사용자 코드를 기반으로 현재 Arduino 동작을 실행시킨다. 본 프로젝트에서는 기술적인 문제로 인해, 미리 만들어진 custom Arduino 코드에 기반하여 GUI 프로그램에 결과를 보여주는 방식으로 동작한다. [그림 16]과 [그림 18]을 보면, 전역변수 update_value를 이용하여 GUI Main Window내 update_value의 전 상태와 대조하여 업데이트 여부를 결정한다. [그림 23]을 통해 이 과정에 대해 더 쉽게 이해할 수 있을 것이다.



[그림 23] 에뮬레이션 실행 시 업데이트 동작 흐름도

마지막으로 Stop 버튼은 현재 실행 상태를 확인하고, 만약 실행되고 있다면 정지시키고 초기 상태로 되돌린다. [그림 24]에 나와있는 코드는 이에 관한 전체적인 설명이다.

```

# 정지 버튼 시 동작
def StopExecuting(self):
    if self.loop_signal == True:
        self.loop_signal = False

    # 종료 후 초기 상태로 설정.
    for item in self.drawing_item:
        if item.type != "WIRE":
            item.set_default_texture(self.scene)
        else:
            item.update_scene(self.scene)
    QApplication.processEvents()
  
```

[그림 24] Stop 버튼 동작 코드

③ Text Editor

Text Editor는 Arduino IDE에서 경험할 수 있듯, 모듈 동작을 제어하는 사용자의 코드를 담고 있는 부분이다. 이 섹션을 통해 사용자는 여러 동작 제어를 위한 프로그래밍을 할 수 있으며, 코드 작성 완료 후 Execute 버튼을 누름으로써 코드 저장과 에뮬레이션 실행을 자동으로 하게 된다. 현 상태에서는 아쉽게도 추후 기술할 문제들로 인해 완전한 동작을 지원하지 않는다.

4.2. 평가 및 개선 사항

본 프로젝트를 진행하며 크게 두 가지의 문제점을 발견할 수 있었다. 하나는 에뮬레이션 과정 중의 문제이며, 다른 하나는 GUI 구현 문제이다. 각각에 대한 상세한 내용은 다음과 같다.

에뮬레이션 과정에서 발견된 문제는 총 2개이다. 첫번째 문제는 본 프로젝트에서 사용된 가상 Arduino의 13번 핀을 통한 LED제어 과정 중, digital Write(13, HIGH) 함수 동작에서 제대로 된 메모리 주소에 값이 작성되지 않는다는 것이다. digital Write(13, HIGH)는 가상 Arduino의 PORTB[27]에 해당하는 메모리 값을 바꿔준다. Elf parsing 결과를 통해 해당 영역은 .bss 영역의 0x190e0에 위치한다는 것을 알게 되었고, 이를 통해 에뮬레이션에도 그대로 적용될 것이라 예측했다.

```
Sections
=====
                NULL      0      0
.init          PROGBITS   8000
.text          PROGBITS   8018
.fini          PROGBITS   87a4
.rodata        PROGBITS   87bc
.ARM.exidx     ARM_EXIDX   88f
ARM_UNWIND LOAD
.eh_frame      PROGBITS   8810
.init_array    INIT_ARRAY 18814
.fini_array    FINI_ARRAY 1881c
.data          PROGBITS   18820
.persistent    PROGBITS   1902f
.bss           NOBITS     19024
.noinit        NOBITS     191e4
.comment       PROGBITS   0
```

[그림 25] elf parsing 결과

```
_retarget_lock_try_acquire FUNC GLOBAL 8bccc
_register_exitproc          FUNC GLOBAL 86e4 bc
_stack_init                 NOTYPE WEAK 80e4 0
PORTB                       OBJECT GLOBAL 190e0 20
_retarget_lock_close_recursiveFUNC GLOBAL 86c0
g_aArduinoDueTable          OBJECT GLOBAL 18824
```

[그림 26] PORTB의 메모리 주소 부분

그러나 컴파일 과정 혹은 Unicorn Engine의 에뮬레이션 과정 중 한 부분에서 예기치 못한 동작으로 해당 메모리 번지에 값이 써지는 것이 아닌, 0x00번지에 값이 갱신되는 현상을 발견했다.

프로젝트 기간동안 이에 대한 원인을 규명하지 못하였으며, 차후 개선 작업 시 가장 우선적으로 해결해야 할 사항으로 고려된다.

두번째 문제는 Arduino 코드에 작성된 delay()함수가 제대로 동작하지 않는다는 것이다. 이 또한 제대로 된 원인을 규명하지 못하였으며, 컴파일 시 사용된 arm-none-eabi-gcc 컴파일러가 컴파일 과정 중 delay()함수에 대한 내부 동작을 변형했을 것이라 예측하고 있다. 이 문제로 인해 현재 구현된 프로그램에서는 LED의 동작이 불규칙적으로 이뤄지고 있다.

GUI에서도 문제점을 발견할 수 있었는데, 아직 완전한 기능을 지원하지 못한다는 것이다. Graphics View에서 렌더링 될 각종 모듈을 지원하고 있지 않으며, 모듈과 보드 간 전선 연결을 마우스로 조작할 수 없다는 것도 포함된다. 아직 완전한 동작을 지원하지 않고 있으므로 추후 이 부분의 보강도 필요할 것으로 보인다.

5. 결론 및 기대효과

가상화 아두이노 구현을 통해 실제 하드웨어의 제약을 받지 않고 아두이노와 LED, 모터 등 다양한 모듈들을 컴퓨터 상에서 GUI로 조작하고 테스트할 수 있도록 하는 것이 최종 목표이다. 특히, 향후에는 현재 프로젝트 상에서 임의로 짠 C언어로 된 아두이노 예시 프로그램 외에도, 직접 Arduino IDE로 컴파일한 원본 ELF 파일을 이 프로젝트의 Input Code로 넣어서, 성공적으로 에뮬레이션을 동작시키는 것이 궁극적인 목표라고 할 수 있다. 아두이노 내의 하드웨어를 Unicorn Emulator를 통해 구현할 수 있으므로, GPIO와 같은 하드웨어 종속적인 기능들을 사용할 수 있다. 이렇게 구현된 기능들은 추후 아두이노를 다루는 수업에서 교육적인 목적으로 사용하거나, 아두이노와 관련된 실험적인 기능을 구현하는 연구 분야에서 응용할 수 있을 것으로 기대된다.

경제적, 시간적인 측면에서도 기대 효과를 볼 수 있다. 아두이노 하드웨어에 들어가는 비용을 최소화할 수 있기 때문이다. 그리고 사용자가 직접 하드웨어를 세팅하고 모듈들을 연결해서 사용하는 것에 비해 간편하게 아두이노를 조작할 수 있으므로 시간적인 비용도 줄일 수 있을 것이다.

6. 참고 문헌

[1] Unicorn Engine Documentation : <https://www.unicorn-engine.org/docs/tutorial.html>

[2] 4 Arduino Simulators You Can Use in Your Electronics Projects :

<https://www.makeuseof.com/arduino-simulators-electronics-projects/>

[3] 'AIoT 소프트웨어' 조진성 교수님 강의자료

[4] Arduino Official Information: <https://www.arduino.cc/>

[5] Unicorn Engine Presentation: <https://www.unicorn-engine.org/BHUSA2015-unicorn.pdf>

[6] QEMU Documentation: <https://www.qemu.org/docs/master/>