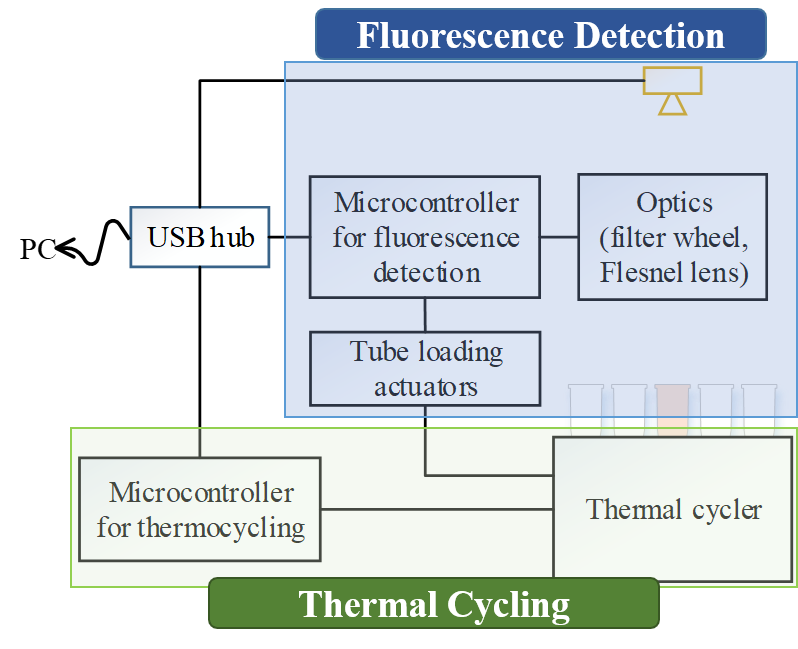
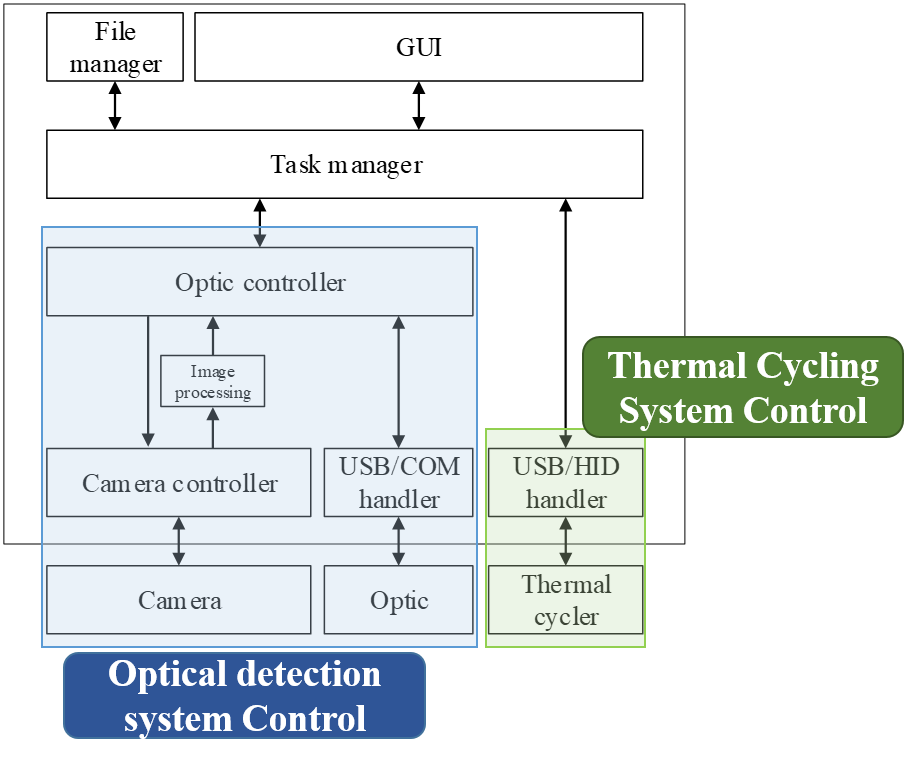
|  |
| --- |
| **Duxcycler prototype** |
|  |

## 전체적인 구조

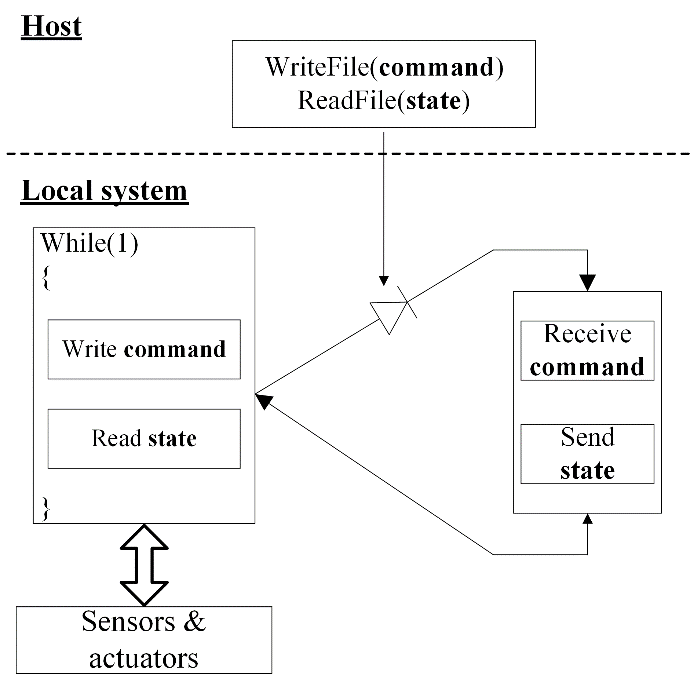
Duxcycler prototester는 기존의 DNA 증폭 만 갖춘 Duxcycler를 Multiplex real-time PCR로 발전시키기 위해 개발 진행중이다. 따라서 열 순환 기능만 갖춘 기존 장비에 다중 형광 검출기능을 추가하는 것이 목표이다.

Duxcycler의 하드웨어는 열 순환을 담당하는 증폭부와, 형광을 검출하는 검출부로 나뉘게 된다. 증폭부는 PCR protocol을 받아 fan, peltier, NTC thermistor 등의 sensor 및 actuator 들로 chamber의 온도를 제어하며, 모니터링 한다. 검출부는 PCR protocol의 매 사이클마다 형광물질을 카메라로 검출하며 튜브를 loading 하기 위한 servo motor, linear stepper motor를 이용하여 emission filter을 움직이는 filter wheel 그리고 25개의 튜브를 조사하기 위한 led등이 사용된다. 증폭부와 검출부는 서로 독립적인 microcontroller를 사용하고 있으며, host-local system으로 구성되어 있다.

전체적인 소프트웨어 구조는 위 오른쪽 그림과 같이 구성된다. GUI는 사용자의 요청에 따라 이벤트를 발생시켜 Task manager에게 전달한다. Task manager는 발생한 이벤트에 알맞은 명령어를 증폭부와 검출부 하드웨어를 제어하는 Optic controller에 전달한다. 증폭부는 USB/HID통신으로 Thermal cycler의 microcontroller와 연결되어 있다. Optic controller는 USB/COM 통신으로 검출부의 microcontroller와 연결되어 있으며, 형광 이미지를 얻기 위해 USB camera를 제어하는 Camera controller와 연결되어 있다. Camera controller에서 형광 이미지를 취득할 경우 이미지 처리를 통해 각 형광의 25개 튜브의 밝기 값을 얻어 Task manger에 전달한다.

## Host-Local system



증폭부와 검출부의 microcontroller는 Host-Local system 으로 사용자 PC에서 구동되는 응용 프로그램과 연결된다. Host-Local system은 아래와 같은 순서로 작동한다.

1. Local system은 loop를 돌면서 이전 loop에 받은 command를 주변 sensor와 actuator 들에게 전달 후, 현재 state를 읽어 저장한다.
2. Host 에서 command를 보내거나 state를 요청하면, ISR(Interrupt Service Routine) 발생한다.
3. 발생한 ISR 에서는 받은 command를 저장하고, 저장되어 있던 state를 host로 전송한다.
4. 위의 1, 2, 3과정 반복

이와 같은 Host-Local system을 사용할 때의 이점은 아래와 같다.

* Host 가 user-interface 를 담당하므로 Local-system 부분을 더욱 소형, 저가격화 할 수 있다.
* Local system의 수동적 통신으로 인해 위와 같이 software가 간단해짐
* 기능적 분할로 인해 개발 및 유지보수, 추가적인 기능 확장에 유리해짐

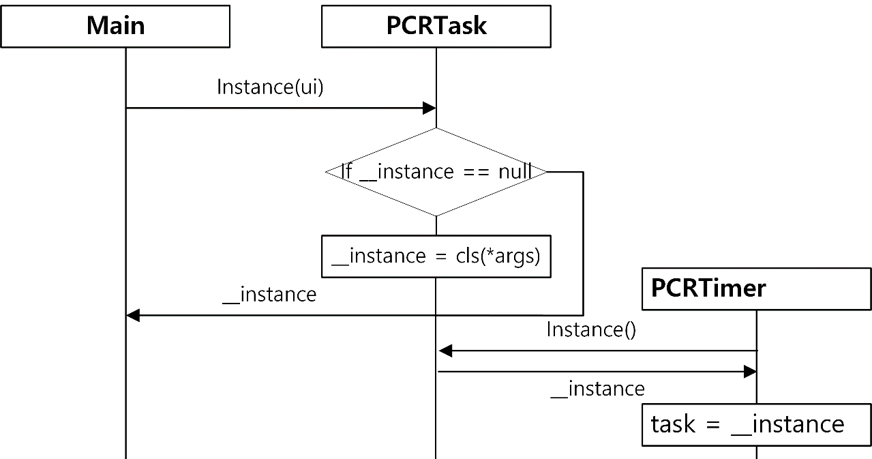
## Host-Local system

전체 파일 구조는 아래와 같다.

* **Root**
  + main.py
    - **ui**
      * main\_ui.py
  + conf.json
  + **pcr**
    - **constants**
      * config.py
      * constant.py
    - **hid**
      * hid\_controller.py
      * rx\_action.py
      * tx\_action.py
    - **optic**
      * camera.py
      * optic.py
      * shot\_thread.py
    - file\_manager.py
    - logger.py
    - protocol.py
    - serial\_ctrl.py
    - ~~timer.py~~
    - task.py

## main.py

2021/09/15 기준으로 작성됨



if \_\_name\_\_=="\_\_main\_\_":

    # PCR Task

    pcr\_task = PCRTask.instance(ui.window)

    # PCR Task(timer)

    pcr\_timer = PCRTimer()

    pcr\_timer.start()

    # Start main ui

    ui.app.exec()

    pcr\_task.optic.close()

    # Exit process

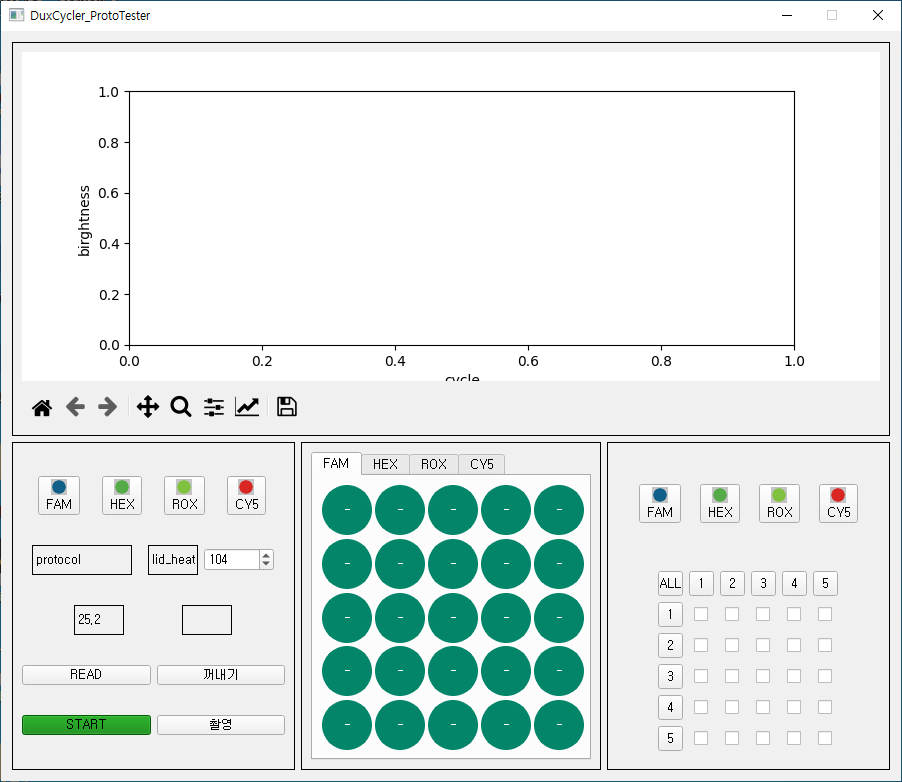
    sys.exit()

* PCR Task 인스턴스 생성 🡪 timer 생성 후 실행 🡪 main\_ui 실행 🡪 (ui 종료시)camera thread 종료 🡪 process 종료 순으로 짜여 짐.
* task.py의 class PCR\_Task는 Singletone 기법으로 구현되어 있기 때문에 PCR\_Taks.instance()로 인스턴스를 생성해야 한다.
  + PCR\_Task가 single tone 임에 따라 다른 class 및 code와 결합도가 매우 높다는 것을 항상 염두 해 두어야 한다. (circular import 조심!)
* ~~210915 기준 camera\_worker.py 안의 thread가 ui 종료 시 계속 실행되는 문제 때문에 pcr\_task.cam\_worker.close() 해줘야만 정상적으로 thread kill이 된다.~~

## ui/main\_ui.py

PyQt5로 짜여진 Main UI

Duxcycler에 Multiplex real-time PCR 기능이 요구됨에 따라 PCR중 형광 분석 및 모니터링 기능이 요구된다. 🡪 기존의 Duxcycler 프로그램의 GUI에서 개선 필요함



Main UI 의 계층도는 아래와 같이 구성된다.

* MainUI
  + GraphFrame : 형광 값을 plot하여 증폭곡선을 display
  + ControlFrame : Start, Stop, 형광 선택 등 Duxcyler 의 제어를 담당하는 부분
  + ImageFrame : 가장 최근의 형광 값 display
    - ImageTab : 여러 형광의 형광 값을 display 하기위해 QTabWidget 사용
  + GraphSelectionFrame : GraphFrame 에서 plot될 튜브와 형광을 선택하는 부분

## conf.json

Prototype 단계이기 때문에, 실험 중 custom 필요한 변수들 json 파일로 묶어 놓았다.

* cam\_name
* cam\_settings
  + Arducam\_IMX298
    - width
    - height
    - exposure
    - focus
    - gain
    - gamma
    - whitebalance
    - low\_light\_compensation
    - exe\_filename
    - centers
    - rad
  + optic
    - filter\_wheel
      * coarse\_speed
      * fine\_speed
      * max\_speed
      * accel
      * filter\_postion
        + FAM
        + HEX
        + ROX
        + CY5
    - servo\_motor(아직 미구현)
    - fluoresences
    - channel
      * FAM
      * HEX
      * ROX
      * CY5
    - led
      * pwm

## pcr/constants/config.py

conf.json 파일을 읽어 변수에 저장해 준다.

## pcr/constants/constant.py

Duxcycler 와 HID 통신시 필요한 command와 state 등을 상수로 정의해 놓는 곳

|  |  |
| --- | --- |
| **Command** | |
| **Description** | **value** |
| NOP | 0x00 |
| TASK\_WRITE | 0x01 |
| TASK\_END | 0x02 |
| GO | 0x03 |
| STOP | 0x04 |
| PARAM\_WRITE | 0x05 |
| PRAM\_END | 0x06 |
| PAUSE | 0x07 |
| CONTINUE | 0x08 |
| BOOTLOADER | 0x55 |

|  |  |
| --- | --- |
| **State** | |
| **Description** | **value** |
| READY | 0x01 |
| RUN | 0x02 |
| PCREND | 0x03 |
| STOP | 0x04 |
| TASK\_WRITE | 0x05 |
| TASK\_READ | 0x06 |
| ERROR | 0x07 |
| REFRIGERATION | 0x08 |
| PAUSE | 0x09 |
| INCREASE | 0x10 |

|  |  |
| --- | --- |
| **State** | |
| **Description** | **value** |
| INIT | 0x00 |
| COMPLETE | 0x01 |
| INCOMPLETE | 0x02 |
| RUN\_ REFRIGERATIO | 0x03 |

## pcr/hid/hid\_controller.py

Duxcycler 펌웨어와 hid 통신 ‘rw’ 수행과 관련됨

* import hid
  + Windows 환경에서 hid 패키지를 사용할 경우, “C:\Windows\System32”에 “hidapi.dll” 파일이 필요하다.
    - file : <https://github.com/libusb/hidapi/releases>
    - refer : <https://pypi.org/project/hid/>

|  |  |
| --- | --- |
| **Function\_name** | **Description** |
| read() | \_\_hid\_device 를 사용하여 BUFSIZE 만큼 Rx\_buffer를 read 한다. |
| write() | \_\_hid\_device 를 사용하여 매개변수로 받은 Tx\_buffer를 write한다. |

* read, write 함수 모두 hid logging을 위한 decorator 달려있음
  1. 순환 import 방지하기 위해 line 42 에 logger import(재설계 필요한가?)

hid\_controller.py 안에서 발생하는 exception은 모두 “windll” 패키지의 MessageBoxW() 를 사용하여 display 한다.

# Duxcycler's vid & pid

VENDOR\_ID      = 0x04D8

PRODUCT\_ID     = 0xFB76

PRODUCT\_ID\_BMX = 0xEF7F

...

# Max size of rx\_buffer

BUFSIZE        = 65

TIMEOUT        = 3

* hid 연결에 필요한 vender id 와 product id 가 정의되어 있다.
* Buffer size는 65, hid 통신 timeout 시간은 3초

try:

    hid\_devices = hid.enumerate()

    hid\_devices = list(filter(device\_filter, hid\_devices))

    ...

    # Only use first one in the devices list.

    \_\_hid\_device =  hid.Device(hid\_devices[0]['vendor\_id'], hid\_devices[0]['product\_id'])

# manufacturer and serial

manufacturer, serial = \_\_hid\_device.manufacturer, \_\_hid\_device.serial

* hid\_devices : 모든 hid connection 중 VENDOR\_ID 와 PRODUCT\_ID\_BMX 혹은 PRODUCT\_ID 가 일치하는 list
* \_\_hid\_device : 장치와 연결된 hid 객체, 현재는 Multi connection 을 지원하지 않으므로 hid\_devices 의 첫번째 요소만 사용하여 연결

## pcr/hid/rx\_action.py

hid\_controller.py에서 읽어온 buffer를 dictionary 형태로 바꾸는 역할

|  |  |
| --- | --- |
| **Function\_name** | **Description** |
| set\_buffer(raw\_data) | hid 에서 읽어온 rx\_buffer를 dictionary 형태로 변환하여 저장 |

* 변환된 rx\_buffer(dictionary) 는 직접 rx\_action 모듈을 import 하여 접근해야 함.

Rx\_buffer의 index는 아래와 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| Buffer name | Index | Buffer name | Index |
| RX\_STATE | 0 | RX\_RES | 1 |
| RX\_CURRENTACTNO | 2 | RX\_CURRENTLOOP | 3 |
| RX\_TOTALACTNO | 4 | RX\_KP | 5 |
| RX\_KI | 6 | RX\_KD | 7 |
| RX\_LEFTTIMEH | 8 | RX\_LEFTTIMEL | 9 |
| RX\_LEFTSECTIMEH | 10 | RX\_LEFTSECTIMEL | 11 |
| RX\_LIDTEMPL | 13 | RX\_LIDTEMPH | 12 |
| RX\_CHMTEMPH | 14 | RX\_CHMTEMPL | 15 |
| RX\_PWMH | 16 | RX\_PWMDIR | 18 |
| RX\_PWML | 17 | RX\_LABEL | 19 |
| RX\_TEMP | 20 | RX\_TIMEH | 21 |
| RX\_TIMEL | 22 | RX\_LIDTEMP | 23 |
| RX\_REQLINE | 24 | RX\_ERROR | 25 |
| RX\_CUR\_OPR | 26 | RX\_SINKTEMPH | 27 |
| RX\_SINKTEMPL | 28 | RX\_KP\_1 | 39 |
| RX\_KI\_1 | 33 | RX\_KD\_1 | 37 |
| RX\_SERIALH | 41 | RX\_SERIALL | 42 |
| RX\_SERIALRESERV | 43 | RX\_VERSION | 44 |

## pcr/hid/tx\_action.py

hid\_controller.py에서 Duxcycler 펌웨어로 보낼 Tx\_buffer를 만들어 주는 역할

Tx\_buffer의 index는 아래와 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| Buffer name | Index | Buffer name | Index |
| TX\_CMD | 1 | TX\_ACTNO | 2 |
| TX\_TEMP | 3 | TX\_TIMEH | 4 |
| TX\_TIMEL | 5 | TX\_LIDTEMP | 6 |
| TX\_REQLINE | 7 | TX\_CURRENT\_ACT\_NO | 8 |
| TX\_BOOTLOADER | 10 |  |  |

|  |  |
| --- | --- |
| **Function\_name** | **Description** |
| make\_nop() | NOP(0으로 채워진) Tx\_buffer 생성 |
| make\_taskWrite(action, preheat, line) | Task write Tx\_buffer 생성   |  |  | | --- | --- | | TX\_CMD | Command.TASK\_WRITE | | TX\_ACTNO | GOTO : 250 or Label(int) | | TX\_TEMP | 해당 protocol의 target 온도 | | TX\_TIMEH | 온도 유지시간(High bit) | | TX\_TIMEL | 온도 유지시간(Low bit) | | TX\_LIDTEMP | Lid heater 온도(preheat) | | TX\_REQLINE | 해당 protocol의 line number(line) | | TX\_CURRENT\_ACT\_NO | 해당 protocol의 line number(line) | |
| make\_taskEnd() | Task end Tx\_buffer 생성   |  |  | | --- | --- | | TX\_CMD | Command.TASK\_END | |
| make\_go() | Task go Tx\_buffer 생성   |  |  | | --- | --- | | TX\_CMD | Command.GO | |
| make\_stop() | Stop Tx\_buffer 생성   |  |  | | --- | --- | | TX\_CMD | Command.STOP | |
| make\_bootLoader() | Boot loader 용 Tx\_buffer 생성   |  |  | | --- | --- | | TX\_CMD | Command.BOOTLOADER | |
| make\_requestLine(request\_line) | Request line 용 Tx\_buffer 생성   |  |  | | --- | --- | | TX\_REQLINE | request\_line | |

## pcr/timer.py

100ms 간격으로 loop 하며 pcr의 task를 실행시키는 타이머

class PCRTimer(QTimer):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.task = PCRTask.instance()

        # Must be use precise timer

        self.setTimerType(Qt.TimerType.PreciseTimer)

        self.setInterval(TIMER\_DURATION)

        self.timeout.connect(self.run)

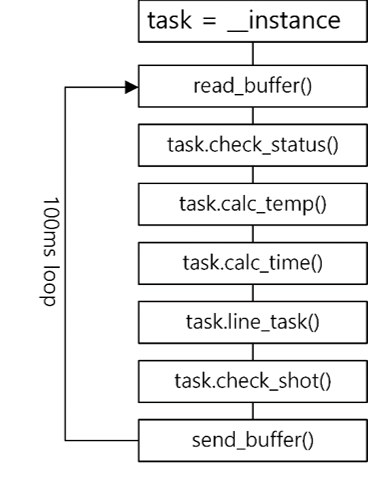
        self.cnt = 0

* PyQt5.QtCore 의 QTimer 상속
* Qtimer의 기본값이 CoarseTimer 이므로 Duxcycler 펌웨어와 원활한 hid 통신을 위해 Qt.Timer.PreciseTimer 로 설정해 주어야 함
* setInterval() 함수로 Timer의 interval 을 100ms 으로 설정
* timeout.connect() 로 Timer와 PCR\_Timer 안에 정의되어 있는 self.run() 함수 연결

아래는 class PCRTimer들의 함수들 이다.

|  |  |
| --- | --- |
| **Function\_name** | **Description** |
| read\_buffer(self) | * hid 통신으로 firmware 가 보낸 Rx\_buffer를 읽어온다. * hid\_controller.py 에서 읽어온 buffer를 RxAction.set\_buffer() 이용하여 RxAction 으로 변환 |
| send\_buffer(self): | * PCR\_Task 인스턴스에 정해진 command 변수에 따라 Duxcycler firmware에 Tx\_buffer를 보낸다. * tx\_action.py 의 함수들을 사용하여 tx\_buffer 생성 |
| def run(self): | * 100ms 마다 실행되며 PCRTask의 함수들 실행 |

run() 함수는 아래와 같이 PCRTask의 함수들을 실행시킨다.



## pcr/task.py

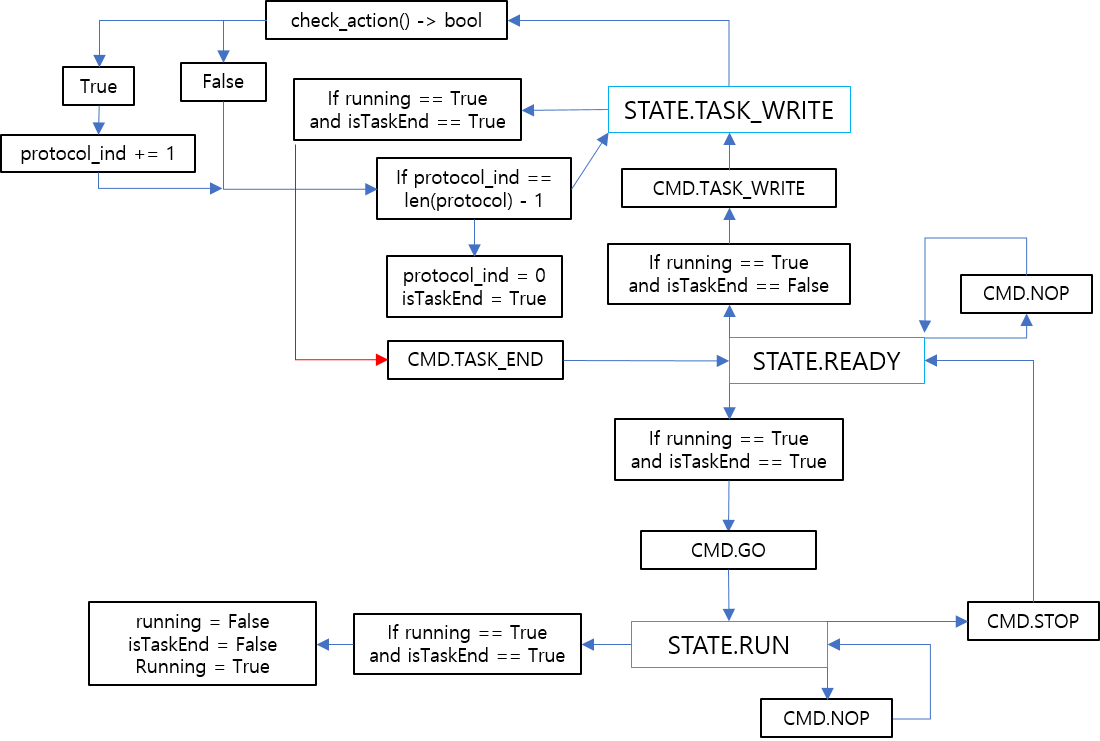
PCRTask class를 포함하고 있으며 이는 singletone class로 구현되어 있다. 증폭부와 검출부에 알맞은 명령어를 setting하며, task management를 하는 곳

아래는 class PCRTask 들의 함수들 이다.

|  |  |
| --- | --- |
| **Function\_name** | **Description** |
| pcr\_start(self) | MainUI에서 Start 버튼 동작 시 실행되는 함수 |
| pcr\_stop(self) | MainUI에서 Stop 버튼 동작 시 실행되는 함수 |
| set\_preheat(self, preheat) | MainUI에서 preheat 설정 시 저장하는 함수 |
| load\_protocol(self, protocol\_name) | PCRTask init시 또는 MainUI에서 protocol을 읽을 시 Protocol을 읽어오는 함수 |
| get\_device\_protocol(self) | 최근 Protocol 파일 읽을 시 사용되는 함수   * 220322 : self.deviceCheck flag 제거하고 get\_device\_protocol call을 timer -> PCR\_Task.\_\_init\_\_() 으로 옮김 |
| check\_action(self) | self.check\_status() 에서TASK\_WRITE 로 프로토콜 라인을 보낸 후 rx\_buffer 를 check 하는 함수 |
| check\_status(self) | 읽어온 rx\_buffer의 Status를 기반으로 flag들을 변경해 주는 함수 |
| calc\_temp(self) | Chamber와 LID heater 온도 계산 후 MainUI에 Display 해주는 함수 |
| line\_task(self) | 현재 PCR protocol의 loop를 계산해주는 함수   * 220505: 형광 값 update시 UI 멈추는 오류가 있어 형광 display를 line task에서 하도록 수정(self.update\_vals) |
| calc\_time(self) | 현재 protocol의 남은 시간을 계산해 주는 함수   * Duxcycler에서 해당 protocol의 남은 시간을 display 하도록 요구되지 않았으므로 현재 사용하지 않음 |
| check\_shot(self) | shot 할지 확인하는 함수   * shot logic : 현재 protocol이 10초 남았고, 다음 protocol이 “GOTO” 일 때 shot을 찍도록 설정 |
| set\_update\_vals(self, fluor, values) | optic.py 에서 이미지 처리 후, line\_task() 에서 형광 값을 display 할 수 있도록 self.update\_vals에 push 해 주는 함수 |
| chamber\_take\_out(self, state) | MainUI에서 꺼내기 버튼 동작 시 실행되는 함수 |

* check\_status() 의 command setting 과정은 아래와 같다.
  + State(Constant.State) 기반으로 작성되었으나, logic error 와 code 가독성을 위해 "이전 커맨드" 기반으로 변경하였음.
  + 커맨드 setting 을 위한 flag 변수들은 아래와 같이 존재한다.

|  |  |
| --- | --- |
| **Command** | |
| **Variable** | **Description** |
| self.state | Duxcycler 펌웨어의 status |
| self.command | Duxcycler 펌웨어에 보낼 command |
| self.running | 현재 Duxcycler가 running 중인지 |
| self.finishPCR | running중 PCR protocol을 완료 했는지 |
| self.taskEnded | running 중 PCR protocol을 Duxcycler 펌웨어에 모두 보냈는지 |



## pcr/file\_manager.py

PCR을 진행하면서 생성되는 data(형광 이미지, 형광 값) 등을 저장하고, 관리하는 기능 담당

* 기본적으로 Duxcycler의 data 파일 구조는 아래와 같다.
* C:\mPCR
  + data
    - expt\_name([serialnumber]\_[ %Y-%m-%d\_%H-%M-%S]) # 실험 폴더
      * img # 이미지 폴더
        + [형광]\_[current\_loop].png # 형광 이미지
        + …
      * expt\_name([serialnumber]\_[ %Y-%m-%d\_%H-%M-%S]).xlsx # 형광 값 저장되는 엑셀 파일
    - …
  + log
    - [serialnumber]\_[ duxcycler\_시작시간;%Y-%m-%d\_%H-%M-%S].log # 로그 파일
    - …
  + protocols # 프로토콜 폴더
    - protocol.txt # 프로토콜 파일
    - …
  + recent\_protocol\_python.txt # 최근 프로토콜 이름이 저장되는 파일

class PCRFileManager의 함수는 아래와 같다.

|  |  |
| --- | --- |
| **Function\_name** | **Description** |
| @staticmethod  gen\_expt\_name() | PCR protocol 시작 시 실험 이름을 생성하는 함수 |
| def start\_task(self, expt\_name, fluors) | 실험 이름을 기반으로 data 폴더 안에 실험 폴더와 이미지 폴더 경로, 형광 값이 저장되는 엑셀파일을 생성해 주는 함수 |
| end\_task(self) | PCR 끝날 시 실행 되는 함수 |
| save\_img(self, img, fluor, loop) | shot으로 얻어진 형광 이미지를 저장 하는 함수 |

* class PCRXlsx 에서 형광 값을 엑셀 파일에 저장

## pcr/logger.py

Duxcycler prototype의 logging 기능 담당

* 다른 code 들의 readability를 위해 decorator 사용하여 만들어짐
* log level 재설계 필요
* circular import 문제 조심할 것

## pcr/optic/optic.py

Arduino(Adafruit QtPy) 와 Camera(IMX298)을 통합하여(검출부) 제어하기 위한 용도로 만들어 짐.

class PCROptic():

    def \_\_init\_\_(self, file\_manager, main\_ui, task):

        # main ui

        self.main\_ui = main\_ui

        self.task = task

# Get File manager

        self.file\_manager = file\_manager

        # Serial Connection

        self.com\_port = valid\_ports()[0]

        self.arduino\_serial = SerialTask(serial\_port=self.com\_port)

        # Shot thread start

        ## self.shot\_thread = ShotThread(self.arduino\_serial, self.file\_manager, self.main\_ui)

        self.shot\_thread = ShotThread(self.file\_manager, self.main\_ui, self.task)

        self.shot\_thread.start()

* PCROptic instance 생성시 filemanger, main\_ui, task 를 parameter로 받는다.
* SerialTask instance 생성 🡪 ShotThread instance 생성 순으로 진행됨

class PCROptic의 함수는 아래와 같다.

|  |  |
| --- | --- |
| **Function\_name** | **Description** |
| shot(self, loop, selected\_fluors) | 형광 이미지를 취득 하고 🡪 이미지 처리 🡪 형광 값을 얻기 위해 shot\_thread에서 최신 프레임을 얻어 오는 함수 |
| chamber\_take\_out(self) | chamber를 꺼내고, 넣기 위해 사용하는 함수 |
| go\_home(self) | Filter wheel의 linear stepper motor를 homming 하기 위해 사용하는 함수 |
| close(self) | shot\_thread를 완전히 끄기 위해 instance 소멸 시 사용하는 함수 |

## pcr/optic/shot\_thread.py

PCR Task와 비동기적으로 Shot을 하여 형광 이미지를 얻기 위한 Thread class. 아래와 같은 순서로 처리된다.



1. class PCRTask 의 check\_shot() 함수에서 조건이 만족 될 시 class PCROptic 의 shot(loop, fluors) 함수 실행
2. class PCROptic 의 shot(loop, fluors)는 shot\_thread의 shot() 함수 실행함. 전달 받은 loop, fluors 그대로 전달.
3. shot\_thread의 shotting\_flag = True
4. serial.goto로 filter wheel을 해당 형광의 emission filter 위치로 움직임
5. serial.set\_ledon() 으로 LED ON
6. Camera exposure time의 3배를 LED ON 해야 완전히 조사된 형광 이미지를 얻기 때문에 time.sleep(exposure\_time \* 3)
7. cam\_worker(가장 최신 frame 얻기 위해 image buffer를 비우고 있는 thread class)에서 shot() 함수 실행하여 최신 프레임 가져옴
8. seiral.set\_ledoff() 로 LED OFF
9. 사용자가 선택한 모든 형광을 찍기 위해 4~8을 반복
10. 얻은 각 형광의 이미지를 형광 이미지 폴더에 저장
11. camera.py의 tubes\_intensity() 함수(이미지 처리)를 통해 각 형광 이미지를 형광 값으로 변환
12. 형광 값들을 엑셀 파일에 저장
13. task.set\_update\_vals() 함수를 통해 UI 에 증폭곡선과 형광 값 display

## pcr/optic/camera.py

Arducam IMX298 카메라 모듈을 사용하여 가장 최신 프레임을 가져오는 Thread



* 어떠한 경우에도 최신 frame으로 업데이트 해야 하기 때문에 독립적인 Thread로 구현
* cv2에서 IMX298의 focus, exposure, low-light compensation, white-balance 제어 설정이 구현되어 있지 않으므로, SETTING\_EXE\_FNAME 사용하여 이를 설정
* DirectShow.IAMCameraControl을 사용하여 카메라 setting 제어 🡪 2205015 : 기존의 C++로 짜여져 빌드된 SETTING\_EXE\_FNAME.exe를 없애고 python comtypes 를 이용하여 카메라 설정

class IMX298BufferCleaner 의 함수는 같다.

|  |  |
| --- | --- |
| **Function\_name** | **Description** |
| set\_imx298(self, focus, exposure, lowlight) | setting\_exe\_fname.exe 를 실행시켜 camera setting을 제어하는 함수 |
| \_cam\_init(self) | camera setting을 제어하는 함수 |
| run(self) | looping을 통해 image buffer에 있는 이미지를 최신 프레임으로 유지하기 위한 함수 |
| shot(self) | 가장 최근 프레임을 return 해주기 위한 함수 |
| close(self) | IMX298BufferCleaner thread class를 정상적으로 종료하기 위한 함수 |

def tubes\_intensity(fluor, img):

    values = [i for i in range(25)]

    channel = CHANNELS[fluor]

    \_img = img[:, :, channel]

    for tube\_ind, center in enumerate(CENTERS):

        x, y = center[0], center[1]

        mask = np.zeros(\_img.shape)

        cv2.circle(mask,(int(x), int(y)), RAD, 255, -1)

        values[tube\_ind] = np.mean(\_img[mask == 255])

* 형광 이미지를 input으로 받아 각 튜브의 형광 값을 return 하기 위해 tubes\_intensity() 함수 사용
* tube\_center\_position 🡪 RAD 크기의 mask 이미지 생성 🡪 해당 형광에 알맞은 color channel 값만 mask 모양으로 가져와 평균 🡪 형광값

## pcr/serial\_ctrl.py

Arduino QtPy와 serial 통신을 하기 위한 역할

검출부의 microcontroller 펌웨어의 구조는 아래와 같다.



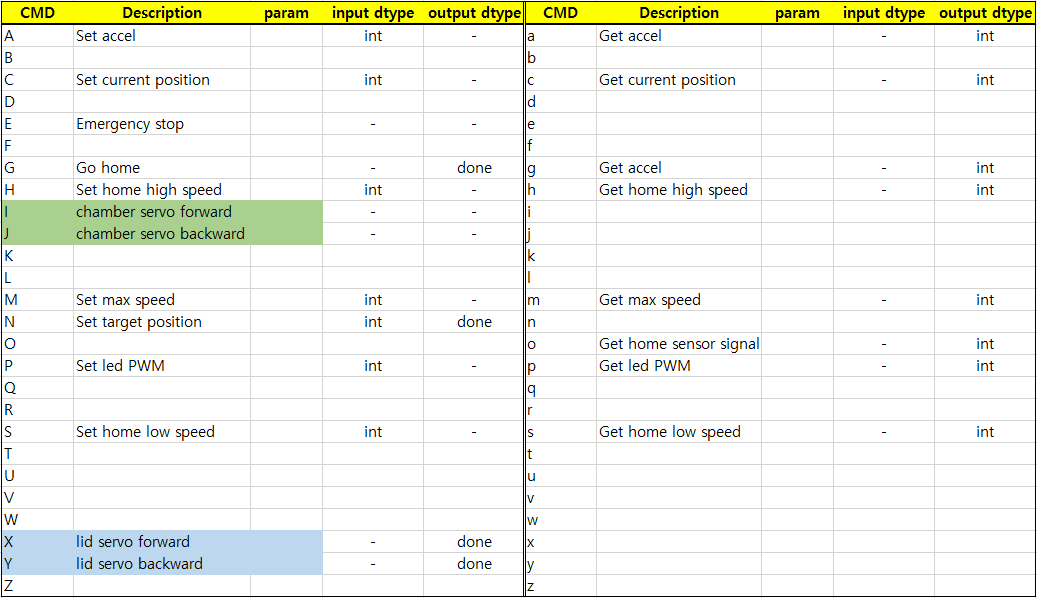
1. sensor, actuator 및 serial 초기화
   * sensor 및 actuator 들에게 mapping된 pin 번호

|  |  |
| --- | --- |
| **Description** | **pin** |
| Stepper motor A- | A9 |
| Stepper motor A+ | A10 |
| Stepper motor B+ | A8 |
| Stepper motor B- | A7 |
| Hall sensor | 6 |
| LED\_ENABLE | 4 |
| Servo motor 1 | A2 |
| Servo motor 2 | A3 |

* + stepper motor setting에 사용되는 parameter 변수들

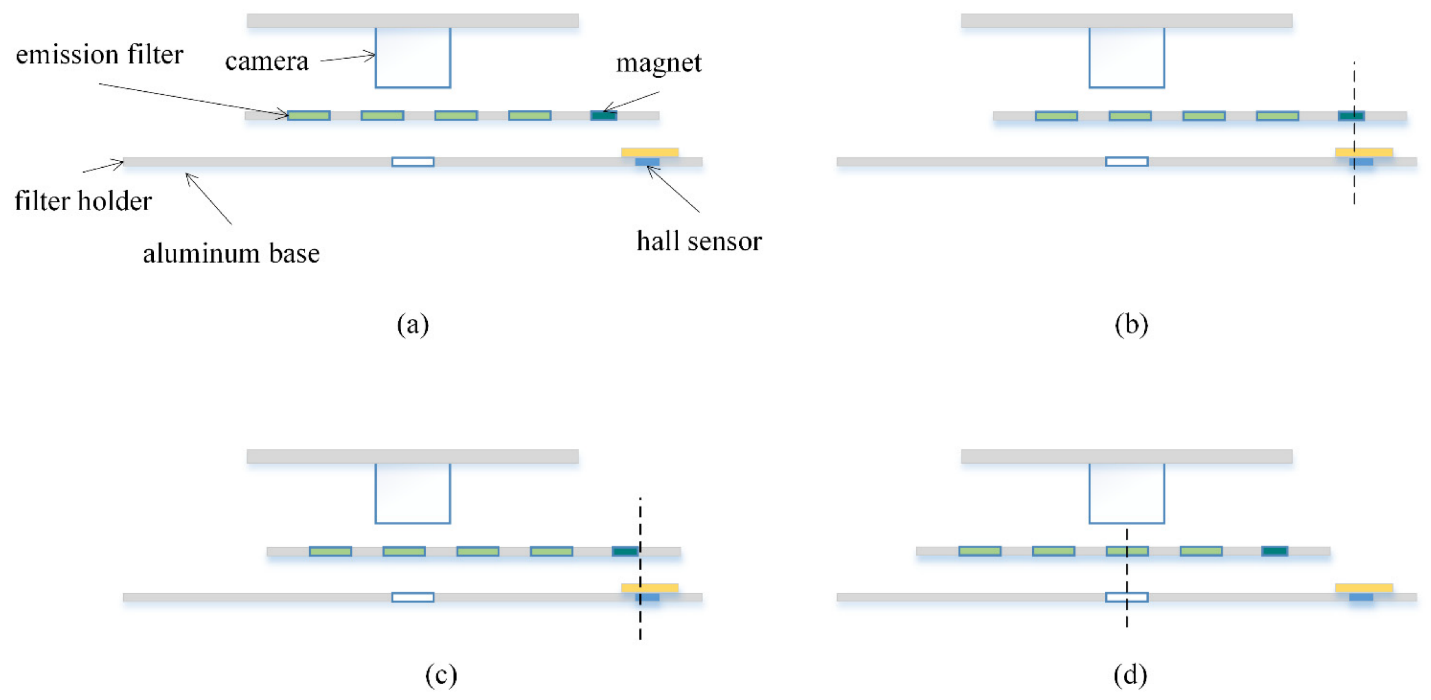
|  |  |
| --- | --- |
| **Variable** | **Description** |
| homeHighSp | Homming중 coarse search시 motor speed |
| homeLowSp | Homming중 fine search시 motor speed |
| mSpeed | filter 순회 중 motor speed |
| acc | filter 순회 중 stepper motor accelation |
| targetPos | target stepper motor position |
| targetDir | target stepper motor direction(현재 미사용) |
| curPos | current stepper motor position |

1. Hall sensor pin의 state가 High 🡪 LOW, LOW 🡪 HIGH 로 변할 때 homing 을 위한 ISR 등록
2. Ardudino loop 문에서 serial 통신 read가 가능할 때 command processing
   * 검출 부에는 아래와 같은 command 가 등록되어 있음



1. command processing 과정에서 runAllowed 변수가 True가 되었다면 stepper motor 를 동작
2. runAllowed = True 도중 stepper motor가 실제 동작 중이 아니라면 runAllowed를 Flase로 바꾸고 serial로 “done” 메시지 보냄

검출부는 각 형광 이미지를 얻기 위해 Filter wheel로 emission filter를 순회한다. emission filter 를 순회한 후, homing을 하기위한 순서는 아래와 같다.





1. 사용자가 선택한 각 형광의 emission filter들의 위치를 순회하며 shot을 한다. 순회가 끝났을 시 filter wheel의 상태가 그림(d) 와 같다고 가정
2. SerialTask에서 Go home command를 검출부 펌웨어에 전송
3. 검출부 펌웨어 에서는 homing = True, runAllowed = True로 flag 설정
4. stepper motor의 speed를 homeHighSp 로 설정 후, 그림(b)와 같이 빠른 속도로 Hall sensor 방향으로 search
5. Hall sensor의 pin이 LOW 🡪 HIGH 로 바뀌었다면 ISR 발생
6. homing = True, homeFine = False 이므로 stepper motor의 speed를 homeLowSp로 설정 후, 그림(c)와 같이 비교적 낮은 속도로 search(homeFine = True로 변함)
7. Hall sensor의 pin이 HIGH 🡪 LOW로 변했다면 다시 ISR 발생하여 stepper motor를 stop 후 원래 setting 값으로 재설정 후, done 보냄