|  |  |  |  |
| --- | --- | --- | --- |
| **Title** | Rainbow Firmware Architecture | | |
| **Distribution** | Software Archive | | |
| **Author** | **Paul Li** | **Date** |  |
| **Review** |  | **Date** |  |
| **Approved** |  | **Date** |  |
| **Remarks** |  | | |

Contents

[1 Introduction 3](#_Toc470163482)

[1.1 Definitions, Acronyms, and Abbreviations 3](#_Toc470163483)

[2 System overview 3](#_Toc470163484)

[2.1 Context 3](#_Toc470163485)

[2.2 Software Overview 4](#_Toc470163486)

[3 Detail design 5](#_Toc470163487)

[3.1 Data Storage 5](#_Toc470163488)

[3.1.1 EEP Storage 5](#_Toc470163489)

[3.1.2 History data 6](#_Toc470163490)

[3.2 Data and Interface to data 8](#_Toc470163491)

[3.2.1 T\_DATA\_OBJ 8](#_Toc470163492)

[3.2.2 T\_DATACLASS 9](#_Toc470163493)

[3.2.3 T\_UNIT 9](#_Toc470163494)

[3.2.4 Subsystems 9](#_Toc470163495)

[3.2.5 Unified interfaces 9](#_Toc470163496)

[3.3 Behavior 9](#_Toc470163497)

[3.3.1 Basic Rule for all tasks 10](#_Toc470163498)

[3.3.2 Schedule 10](#_Toc470163499)

[3.3.3 Measure 11](#_Toc470163500)

[3.3.4 Fluid 12](#_Toc470163501)

[3.3.5 Data Storage 13](#_Toc470163502)

[3.3.6 Heat Control 13](#_Toc470163503)

[3.4 Communication 13](#_Toc470163504)

[3.4.1 Local communication 13](#_Toc470163505)

[3.4.2 Can open communication 14](#_Toc470163506)

[4 Design Decisions 14](#_Toc470163507)

[5 Performance Analyze 14](#_Toc470163508)

[6 Risk 14](#_Toc470163509)

[References 14](#_Toc470163510)

[Revision History 14](#_Toc470163511)

**List of Figures**

[Figure 1: Rainbow Deployment 3](#_Toc470163472)

[Figure 2: Software Overview 4](#_Toc470163473)

[Figure 3: EEP Storage Overview 5](#_Toc470163474)

[Figure 4: Data Log Storage 7](#_Toc470163475)

[Figure 5: Data and Interfaces 8](#_Toc470163476)

[Figure 6: Basic Task Realization 10](#_Toc470163477)

[Figure 7: Schedule Realization – Calibration 11](#_Toc470163478)

[Figure 8: Schedule Realization – Measure 11](#_Toc470163479)

[Figure 9: Measure realization 12](#_Toc470163480)

[Figure 10: Fluid Control realization 13](#_Toc470163481)

# Introduction

The documents presents the architecture of Rainbow firmware. The firmware architecture mainly describe the firmware realized in measure board.



Figure 1: Rainbow Deployment

## Definitions, Acronyms, and Abbreviations

|  |  |
| --- | --- |
| **Term** | **Definitions** |
| MB | Measure Board |
| RO | Relay output board |
| IB | Input IO board |
| AO | 4~20mA output board |
| HMI/MMI | HMI board |
| OO | Object Oriented |
| EEP | eeprom |

# System overview

## Context

The rainbow project contains three different level boards.

* IO boards

Total 3 different boards now; realizes the functions below:

* the relay output,
* input io board
* 2-40mA analoge output.
* Measure board

Realize the functions below:

* measure
* fluid control
* key parameter storage
* history data storage
* communication to UI
* communication to IO boards
* HMI/MMI board

Realize the functions below:

* UI
* History data storage
* Modbus to outside
* Extend ethernet connection to outside
* SC connection
* Communication to Measure board
* Bootloader for measure board.
* ...etc

The interface between IO boards and measure boards is can. The interface between HMI and Measure boards is uart also. The can interface is reserved for further usage. The detail deployment is mentioned in Figure 1.

## Software Overview

This firmware is divided into different packages. The basic rule is to seperate the functions and data. Refine the system to several objects and [abstract](http://www.baidu.com/link?url=V0LahK2w768-BgR0b4Xq06cL9z-3keY9ieqWaLi_E99J94RvBxYDN2JCBnc6OFerEsWiFfD1U1SdJUjuh0VogwBpjBDwXIdniqGN6KDZDkK) each package as one object with its own behavior.



Figure 2: Software Overview

The whole system is divided into four parts.

1. Data storage

This package includes 2 packages:

* EEP storage for configuration data and key process data
* History data storage for all history data

1. Data and interfaces to data

The data package includes three parts:

* T\_DATA\_OBJ

Define the interfaces to acess different data types, like simple type, arrays, structures; also includes valid check, data format convert; trigger eep storage..etc

* T\_DATACLASS

Defines the interfaces to storage different group data.

* T\_UNIT,

Which behaviors as a basic subsystem uses the interfaces of T\_DATA\_OBJ and T\_DATACLASS. Realize the access to the data defined in this subsystem, and also the data storage.

* Subsystems

Which inherited from T\_UNIT, use the common interfaces directly or overload the interfaces to have specific behaviors;

1. Communication

The packages have 2 parts:

* The communication to UI;

Currently use the uart to do the data exchange, which provides the unified interfaces to access all data defined in subsystems.(the data could also be one action trigger defined in subsystems).

* The communication to IO boards

Plan to use the Canopen realize the communication. Measure board acts as the NMT-master.

1. Behavior:

Realizes all the tasks for the function of measure board. This is detailed in chapter 3.

# Detail design

This chapter mainly describe the detail realization of the packages mentioned in chapter 2.

## Data Storage

The data storage is realized tobe the sw common component of Hach DDC.

The firmware now only needs to handle the two different type of data storage.

* One is the configuration data and key process data; those data needs to be stored when power off and read back when system startup. Those data now is stored in EEP, which is highly required to be safty stored.
* History data storage, which stores the history data in case user wants to get the history data. The history data mainly has predefined file size and when file is full, the old record is removed when new record is stored.

The below chapters mainly describe the architectures of thes two packages.

### EEP Storage

The EEP has the limit of write times and could be byte writable. This feature is considered in the realization.

To make a safety storage, the EEP data has backup and also CRC checksum for the whole data.

The total EEP is separated into 4 parts:



Figure 3: EEP Storage Overview

The sequence to update one data stored in EEP:

1. Copy the updated data to ram shadow from the working ram.
2. Update the Head in ram about the checksum of new data, write count and the checksum of itself...etc.
3. Update the working head in EEP to the new head.
4. Update the working data parts (only update the new updated parts in ram shadow)
5. Update the backup head in EEP to the new head.
6. Update the backup data parts (only update the new updated parts in ram shadow)

When device start-up, it starts with the following sequence:

1. Read back the working head from EEP. Gets the last valid head data according to the write counter and checksum of itself.
2. Read back the working data from EEP to ram shadow, check the data integrity with the checksum in head. If equals, go to step 5.
3. Read back the backup head from EEP. Gets the last valid head data according to the write counter and checksum of itself.
4. Read back the backup data from EEP to ram shadow, check the data integrity with the checksum in head. If equals, go to step 5, else go to step 6;
5. Copy the valid data from ram shadow to working ram. Updates the working area in EEP if it fails.
6. If all failed, trigger the data to load defaults and save the data to EEP. Issue alarms.

This could resolve the problem: device power off when device is updating the data in EEP. Device could not get the newest updated data but could get the valid data last stored.

Due to the write time limit, the head data shall have several store places to average the times to store to one place.

There are several data are cyclically updated like the pump/valve working times/counter, the usage of reagent…etc. Those data are not updated to EEP immediately when data is updated. Those data are buffered to the working ram and trigger the storage flag. The storage action is executed in 1 minutes.

#### Common component

To make user easy to extend the interfaces and functions, the parts user needs to modify are now in two file:

* IIC interfaces realization.
* Macro-definitions about the EEP (like the EEP size, the partition of EEP, Ram and shadow location...etc.)

### History data

The target is to have a common realization of data storage. This will be one part of the software common components.

The history data is stored into nor flash (SPI interface). The data log has the below special requirements:

* The data is appended only with max file size predefined, if file is full, the oldest data is updated with the new data.
* Only special functions to format the whole file.
* Each file has same defined structure and length.
* The data needs to be accessed by search filter (start time, end time and type filter)

The architecture of the data log package is described in Figure 4 below:

It divides into 3 layers:

1. Flash interfaces: this mainly handles the functions to the flash chip.
2. Flash file system: handles the new data append, reset file and also the data append read with filter.
3. Log file instances: the realization of all log data and public interfaces.

* Common interfaces of Log file instance (Log\_data in Fig4)
* All files instances (LogData\_A, LogData\_B ...etc. in Fig4)
* Interfaces needs to be extended (log data interfaces in Fig4).

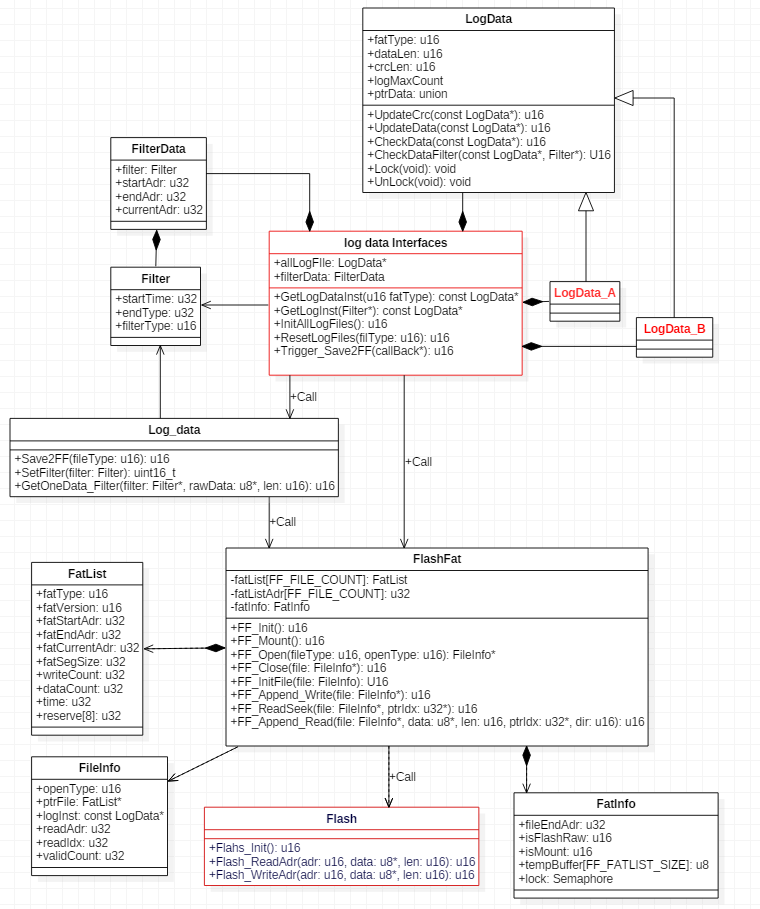


Figure 4: Data Log Storage

#### Common component

To make user easy to extend the interfaces and functions, the parts user need to modify are now in two file:

Log data device specific file has the below parts:

* Structure definition of each data log;
* Realization of each Instance of LogData,
* The functions to get the LogData instance through filter and fatType;

Flash related specified file has the below parts:

* Realize the functions: Flash\_Init; Flash\_ReadAdr; Flash\_WriteAdr;
* Macro-definition about Flash, like sectors/capacity

## Data and Interface to data

This is the core part of the architecture, it has direct connection to all other parts. The data is shared by all other tasks, and it shall have protection for access. To have unified interfaces to those data is a must, the unified interfaces include all the operations related to the data like storage, load defaults, write and read.

The designed architecture described in Fig 5 for data is to reach such requirements.

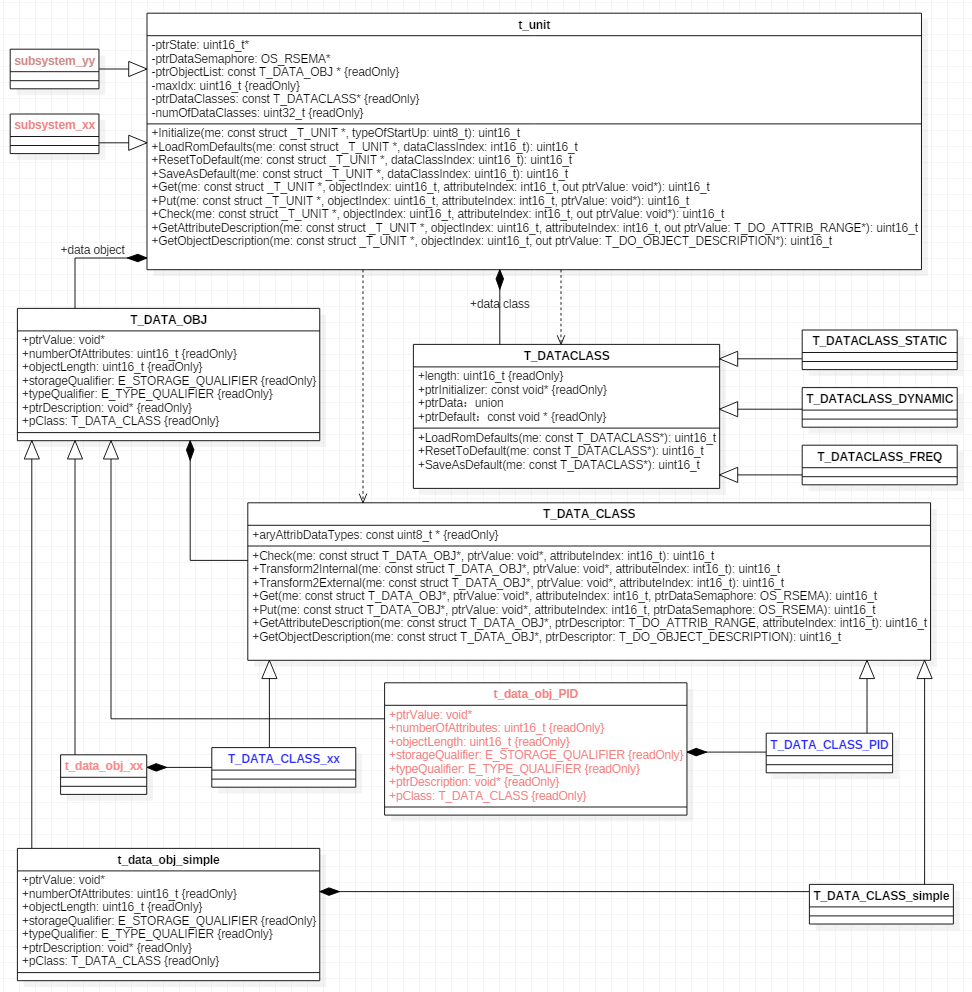


Figure 5: Data and Interfaces

### T\_DATA\_OBJ

This chapter explain how this part work with the other parts

* Consists all the information of the data itself like value, length, attributes, storage type..etc.
* Consists the method to access to the data object. The functions is defined in T\_DATA\_CLASS;
* t\_data\_obj\_simple is one realization of T\_DATA\_OBJ. This could be used as reference. If existed data objects do not meet the requirements, user could extend the t\_data\_obj like the example “t\_data\_obj\_PID”;
* Each data is one instance of one type of T\_DATA\_OBJ,
* Each data is part of t\_unit.

### T\_DATACLASS

T\_DATACLASS handles the default operation for one group of data. It now realizes three different types:

* Dynamic data, which normally is initialized as default when startup. The operation could initialize the data group to be default value during run-time.
* Static data, which normally is stored in EEP chip, it is initialized from EEP data during startup. It is stored to EEP immediately when data is changed during run-time.
* Frequent data, which is stored in EEP chip, but cyclically updated, it is initialized from EEP data during startup. It is stored to EEP in 1 minutes when data is changed during run-time.

The function of T\_DATACLASS gives interfaces for each t\_unit to restore the data to default value. Those functions are used by t\_unit directly.

### T\_UNIT

T\_UNIT have the following features:

* Consists all data instance
* Consists all T\_DATACLASS instance;
* Provide unified interfaces to the data instances and also T\_DATACLASS;
* Device needs to define the data instances and dataclass for each t\_unit instance;
* Device needs to overload the interface to have the specific operations for the specified accessed data.

T\_UNIT is the final realization of the data and data interfaces.

### Subsystems

Each subsystem is one instance of T\_UNIT; it has specified data and data class; it may have overloaded function for the specified accessed data.

### Unified interfaces

Base on the realization of the T\_UNIT, T\_DATA\_OBJ, T\_DATACLASS; it realizes the functions below:

* Data access protection from multi access;
* When data is stored in EEP, the dirty flag is set automatically when data is updated and the eep chip data is updated to the new data automatically by EEP task.
* All the access to the data are unified to Get/Put only,
* Each data could be found by subsystem Id, object id and attribute id automatically.
* It is easy to make load default for each data class;
* It is easy to add one simple proxy to work with communication protocols lile Canopen, Modbus, Hart, profibus, fieldbus...etc.;

## Behavior

This part is device specific, all tasks are described in this chapter.

This chapter is divided into four parts below.

### Basic Rule for all tasks

There are few rules to have separate tasks.

* The task shall be able to abstract to be one Object. It has its own behaviours realized in the task and also it could receive the message or event from outside.
* The behaviour of each task cold be realized as limited state machine, which could be easy to extend and also task could response to the new message/event from outside as soon as possible.

The basic realization could like this:

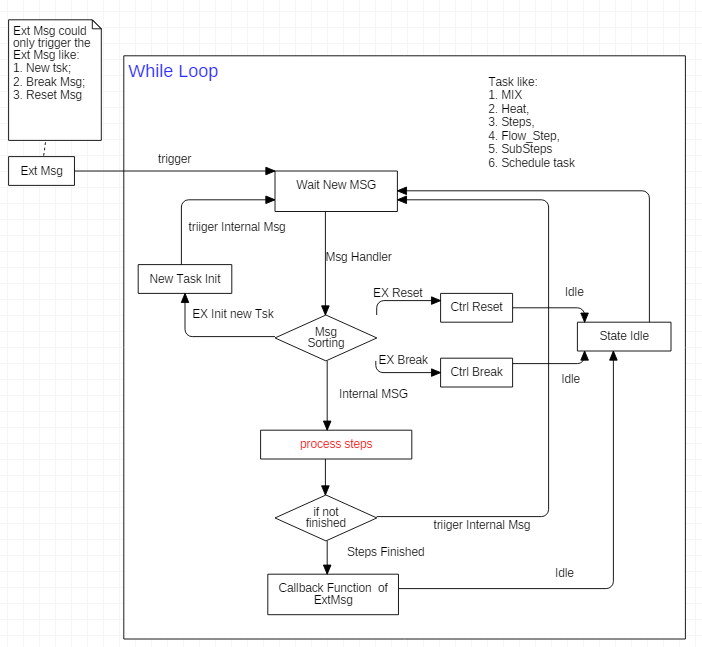


Figure 6: Basic Task Realization

### Schedule

From rainbow requirements, the period schedule has 3 types (calibration, clean, measure, IO action);

So divided into 5 tasks for schedule:

* Schedule manager: manage the trigger/reset/break of all other 4 schedules.
* IO schedule, which receive message from outside, then do the action of its own.
* Clean, which manages the clean steps and receive the event from outside. it sends message to the flow task to do specific flow action.
* Calibration, which manages the calibration steps and receive the event from outside. it sends message to the flow task to do specific flow action and also send message to measure task to get measure data..
* Measure, which manages the calibration steps and receive the event from outside. it sends message to the flow task to do specific flow action and also send message to measure task to get measure data.

Each task has its own sub state for the specific behaviour, each task realizes the behaviour by itself.

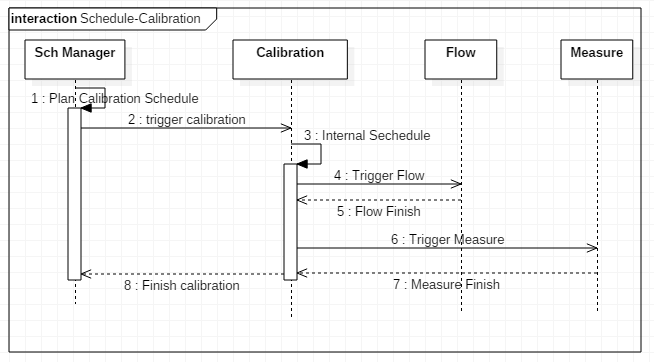


Figure 7: Schedule Realization – Calibration

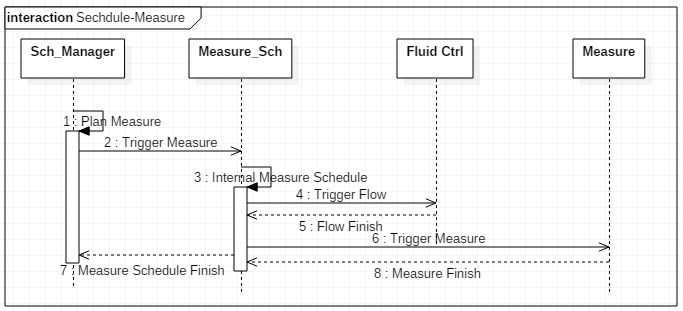


Figure 8: Schedule Realization – Measure

### Measure

The measure is not one task, but one behaviour. It needs to switch the IO state for Led control and also to switch the gain for AD converter. The behaviour also includes the getting the AD data through SPI and also AD data processing.

The tasks needs to have real time control about IO and get AD data from SPI. To realize those function, it needs one HW timer to trigger one real time task to finalize this function.

The tasks below are realized for this function:

* HW timer to switch IO and trigger the real time task to get AD data through SPI;
* Real time task to get AD data through SPI, it receives the trigger event from HW Timer.
* One high priority task to collect the data and do data processing. It receives the event from real-time task.

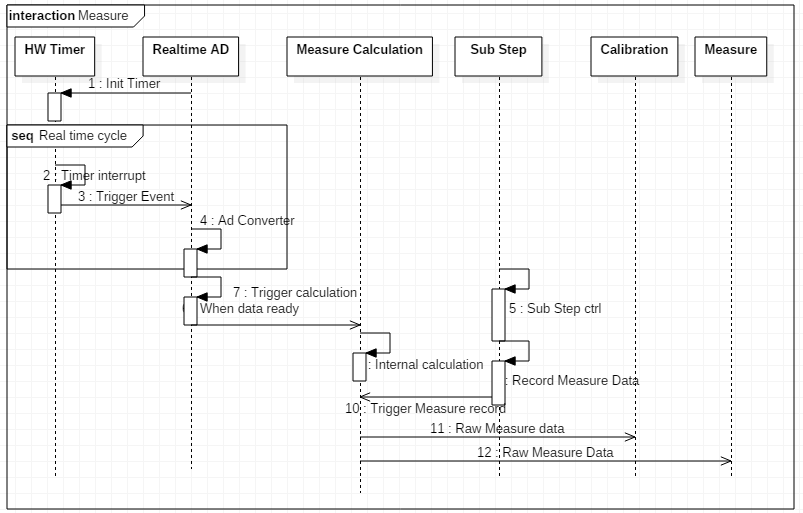


Figure 9: Measure realization

### Fluid

The fluid control system contains 12 valves and 5 step motors. It needs to realize the specific flow steps and also do the single action as required from outside.

Those actions are divided into several tasks:

* Valve Ctrl -> wait new message to control valves open or close.
* Motor Ctrl -> wait new message to control motor with specific step and pps.
* Mix ctrl -> wait new message to control Mix speed, Mix run/stop...etc.
* Substep -> control one specific step action like trigger valve/motor, special actions like delay/mix ctrl ...etc, which normally send message to Mix/Moter/Valve/Heat/Measure..etc. task;
* Flow -> control one flow process like prime, clean...etc which send message to Substep task.

Those tasks realizes the specific flow control. To make the system more flexible, all flow process related actions are parameterized and could be accessed through the data interface. Predefined 150 sub steps divided into 4 group:

* Sub step about valve and step motors, the valve open/close is parameterized, and the step motors direction, pps, and steps...etc are parameterized;
* Sub step about special action: heat ctrl, Mix control, Reset Flow..etc is predefined.
* Sub step about time delay: used for delay between sub steps.
* Flow process: defined separate array of sub steps id. When the process is executed, it executes the predefined sub step according to the ID.

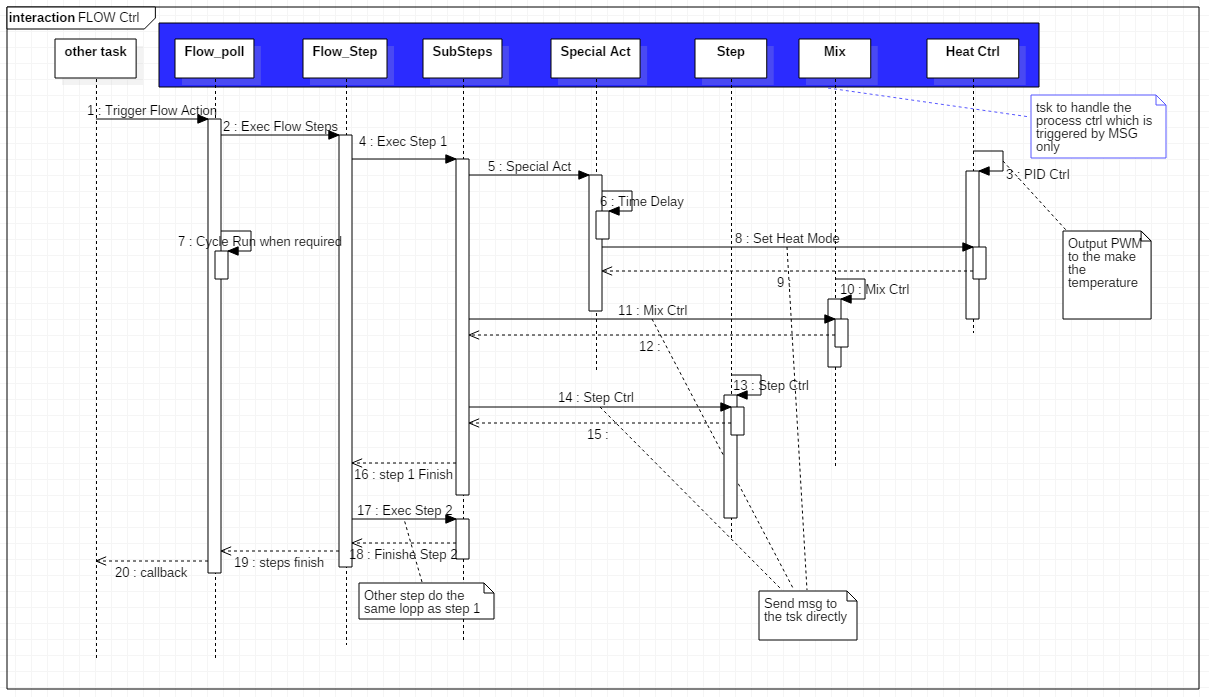


Figure 10: Fluid Control realization

### Data Storage

As mentioned in chapter 3.1; the storage are separated into 2 parts. Each task waits the event from outside and then do the data storage as requested.

### Heat Control

Two tasks are implemented for this behaviour.

* One is to cyclically to get the actual temperatures;
* One is to do PID control

## Communication

### Local communication

The communication to UI board or PC is private. To adapt to the data interface and share almost all information inside of the measure board, the communication support the features below:

* All data interfaces below implemented for master to slave mode.
  + CMD\_POLL
  + CMD\_READ\_OBJ
  + CMD\_WRITE\_OBJ
  + CMD\_READ\_TYPE
  + CMD\_READ\_LENGTH
  + CMD\_READ\_ATR\_NUM
  + CMD\_READ\_RANGE
  + CMD\_READ\_MEM
  + CMD\_WRITE\_MEM
  + CMD\_BURST\_VALUE
* The period burst data could be enabled. This mainly for the data exchanging from measure board to UI or PC.
* The two features above shall be supported for both Uart and Can.

The communication protocols for can and Uart is defined in external documents.

### Can open communication

Can open is planned to be used as the internal communication to IO boards. Measure board acts as NMT-Master. It supports the following features:

//todo

# Design Decisions

* The can interface between measure board and UI board is reserved for further use. This interface is not planned yet.
* Can open is going to use the business edition from other company to avoid the risk.
* No other type of IO boards is planned in can open network.
* Bootloader is not planned to develop for this MCU, the chip is updated through the bootloader in chip.
* The other communication interfaces are not supported. The UI is to manage the convert to other communication protocols/interfaces.
* The IDE and tool chain use the Eclipse and GNU ARM tool chain.
* The bsp uses the source code generated from STM32Cube. Several hal driver from stm32 cube lib may not be suitable for our use case, the adaption shall be made for that.

# Performance Analyze

Due to the unified interfaces to the data is defined, the access to the data is quite simple and security. But it may cause much more cpu time to realize the data access.

The tasks are almost all event triggered, this could make the task more efficient and reusable. Also this causes big workload for OS which needs the OS to be more efficiency.

# Risk

Can open is quite suitable for the use case about measure to IO boards. But it is also quite complex to adapt to. The time for the adaption, integration, test may cost much time.

Several common components are created and initialled in this project. This costs a lot of time and efforts.

References

|  |  |
| --- | --- |
| **Ref.** | **Document** |
| [1] | Firmware requirements Rainbow.doc |
| [2] |  |
|  |  |

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Rev.** | **Description of Version/Changes** | **Primary Author(s)** | **Date** |
| 0.1 | Initial revision. | Paul Li | **2016-12-21** |
|  |  |  |  |
|  |  |  |  |