

NATIONAL UNIVERSITY OF HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FALCUTY OF TRANSPORTATION ENGINEERING
DEPARMENT OF AEROSPACE ENGINEERING

-----000-----



Report #1

ArduPilot

Overview, basic structure, problems and solutions

Student: Nguyễn Anh Quang

Student ID: V1002583

Supervisor: Prof Emmanuel GROLLEAU

Poitier, Spring 2015

Abstract

This report is the first report for the Project called” Implementation a flight command for a Quadricopter”.

This report focuses on introducing about **ArduPilot** and **Maple IDE**. Along with the introduction, problem of using ArduPilot for FlyMaple and the solution for this will be discussed.

Mục lục - Table of Contents

Abstract	ii
Mục lục - Table of Contents	iii
Danh mục hình ảnh - List of Figures	iv
Chương 1	1
Sơ lược về máy bay không người lái và hệ thống lái tự động – An introduction to Unmanned Aerial Vehicle and autopilots.....	1
1.1 Unmanned Aerial Vehicle	1
1.2 Autopilot system.....	4
Chương 2	7
Cấu trúc của ArduPilot – Software architecture of ArduPilot.....	7
2.1 Ardupilot	7
2.1.1 Overview	7
2.1.2 ArduPilot main code	8
2.1.3 Basic structure	10
2.1.4 ArduPilot threading in FlyMaple controller board.....	14
2.1.5 Building firmware for main board – Cross compiling	17
2.1.6 GNU make	19
2.2 ArduPilot problems and suggested solution	22
2.2.1 Problems	22
2.2.2 Solution.....	24
References	26

Danh mục hình ảnh - List of Figures

Figure 1-1: Structure of ArduPilot codebase.....	8
Figure 1-2: AP_HAL structure and the supported boards.....	9
Figure 1-3: Multi-layers structure	10
Figure 1-4: ArduPilot structure	12
Figure 1-5: General functions in the abstract hardware layer	12
Figure 1-6: Mapping in Harware Mapping Layer(HAL_FLYMAPLE_Class.cpp).....	13
Figure 1-7: Tasking for a single thread processor in ArduPilot.....	15
Figure 1-8: Threading in PX4	16
Figure 1-9: FlyMaple threading in ArduPilot from Scheduler.cpp.....	16
Figure 1-10: Stages of Compilation [8].....	17
Figure 1-11: Commands to compile upload to main board in Terminal.....	18
Figure 1-12: Build successfully and uploading firmware	19
Figure 1-13: ArduPilot Files.....	20
Figure 1-14: config.mk structure pointing out the necessary drivers for FlyMaple [14].....	21
Figure 1-15: Summary of “make” process by targets and rules.....	21

Figure 1-16: Summary of “make” process by objects and file formats	22
Figure 1-17: UART_Test.pde [16]	Error! Bookmark not defined.
Figure 1-18: Modification for FlyMaple	Error! Bookmark not defined.
Figure 1-19: Default process and suggested solution.....	25

Chương 1

Sơ lược về máy bay không người lái và hệ thống lái tự động – An introduction to Unmanned Aerial Vehicle and autopilots

This chapter gives an overview about Unmanned Aerial Vehicle (UAV) as well as the autopilot system.

1.1 Unmanned Aerial Vehicle

According to a definition of US Army, Unmanned Aerial Vehicle (UAV), or Unmanned Aircraft (UA) is “*an aircraft that does not carry a human operator and is capable to flight with or without human remote control*” [1]. With many advantages comparing to other kind of aircraft such as compatibility, mobility, safety for pilots...many countries and companies around the world have developed mobility and safety for pilots, UAV nowadays.

The beginning of UA and UAV in general can be found in several documents. Some references believe that the first Unmanned Aerial Vehicle was used on August 22, 1849 when the Austrians used balloons to attack the city of Venice [2]. Meanwhile, according to J.F.Keane and S.S.Carr, the using of radio signal to

control aircraft has been studied since 1911, eight years after the first manned flight of Wright brothers [3]. The development of UAV, like the development of aircraft and any other technologies, is close to the requirements of the wars. During World War I, World War II and many other wars after that, UAV has been used as drones to train the pilots, balloons to surveillance, etc.

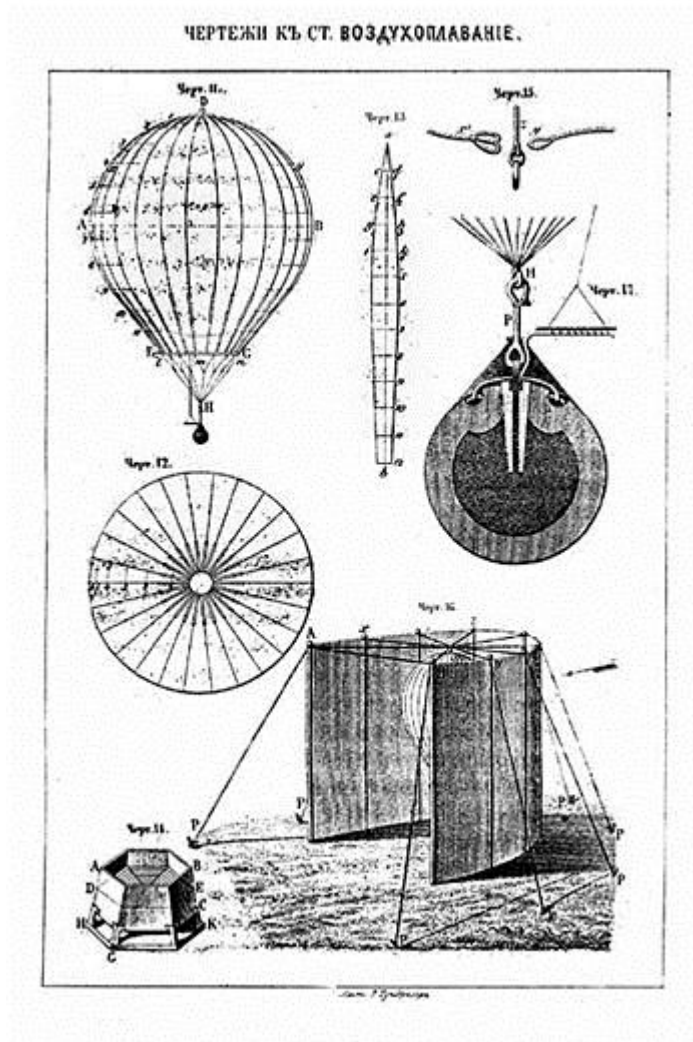


Figure 1-1: Bombing by Balloon, 1848
(Source : Prof. Jurij Drushnin, Moscow, Russia)

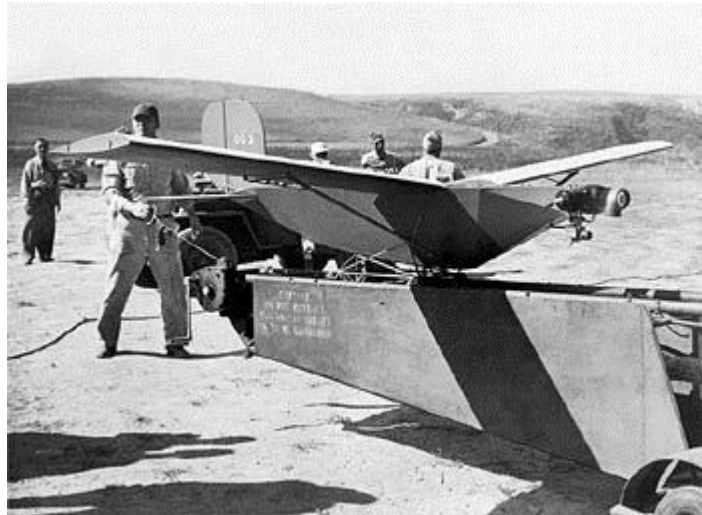


Figure 1-2: OQ-3, Righter O-15-3, El Paso, Texas, 1941(Source: **Righter Family Archives**) [4]



Figure 1-3: Reginald Denny, OQ-3 launch (1947) [4]

From 21st century, noticing the benefits of this technology, many countries and companies has studied this kind of aircraft. At this time, UAV can be found in many sizes, shapes that are suitable for various type of missions (surveillance, mapping, patrolling, scientific, weathering, etc). Some applications of this aerial vehicle can be seen in **figure 1-2**.

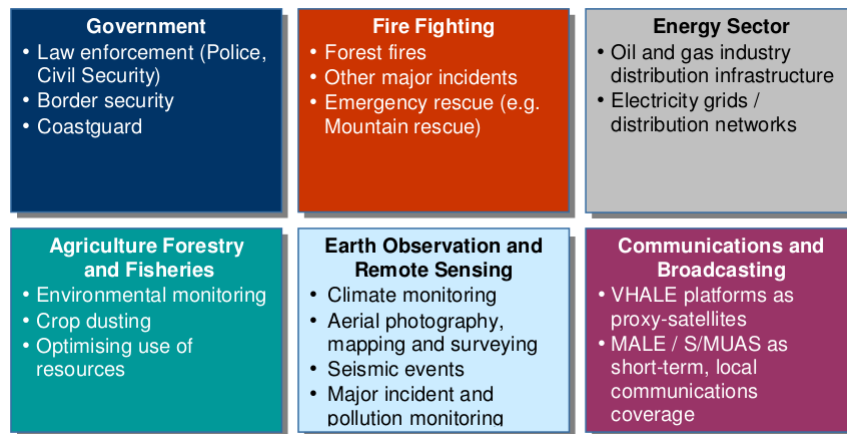


Figure 1-4: Applications of UAV [5]

1.2 Autopilot system

The development of UAV is relative with the development of autopilot. From the beginning of using wind to drive the UAV, which has totally no controllability, to using radio signal to have the directly control from a distance later, it can be conclude that the creation of autopilot system has brought UAV to a higher level. With autopilot systems, UAV can operate without the integration of human, the real unmanned state in its name.

Unlike radio control, which is easy to implement and apply, autopilot system requires many complicated supporting components. However, autopilot has the advantages that other controlling methods cannot achieve; in particular, it can be operated out of sight and can last a very long duration.

The history of autopilot system can be traced back to 1911 with the name of Sperry family. **E.A.Sperry** is an American inventor, which is famous as the co-author of the gyrocompass, one of the most important components for UAV and aircraft in general nowadays [6]. Gyrocompass is a type of non-magnetic compass, it is not affected by the magnetic field of the earth and have a very special behavior while the gyroscope is rotating. As can be seen from **figure 1-5**, there is a fixed axis that will not change while changing the direction of the leg. This behavior will create a difference between the angle of two axis. By

measuring this difference, the angle that the base axis has changed will be noticed. In 1914, **Lawrence Sperry** demonstrated a simple autopilot system operating the rudder and elevators in a straight level flight by flying an aircraft with hands away from the controls [7].

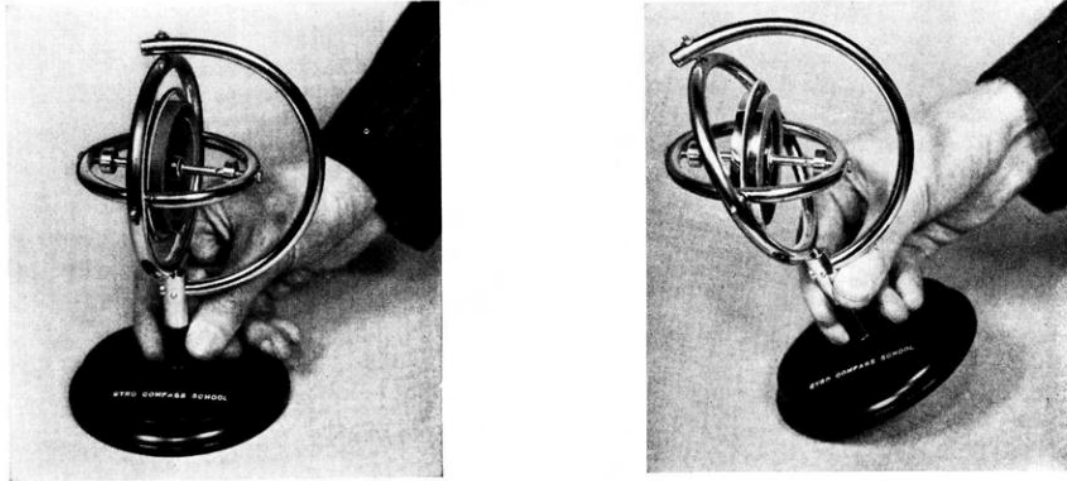


Figure 1-5: Gyrocompass behavior while the gyroscope is rotating

Later, **Elmer Sperry Jr**, son of **Lawrence Sperry**, continued to develop the autopilot system. In 1930, he demonstrate an autopilot system that “*keep an aircraft on true heading and altitude for three hours*” [8].



Figure 1-6: Controlled by Sperry gyroscopes, a pilotless flight in 1930 [9]

At this time, with the development of technology and science, autopilot system is much more precise and of course much more complicated. An autopilot system for a UAV and for aircraft in general today usually includes the following components:

- Sensors: barometer, gyroscope, accelerometer, magnetometer ...
- GPS system
- Control board with a suitable firmware.

Among them, control board and in particular the firmware is the most important component since it will not only access the data of the sensors or the GPS but also do all the computation. The strength of the autopilot depends mainly on this control board, for example, its memory, its computational ability, its communication protocol between components, etc.

This project object is implementation a flight command for a quadricopter. In this project, **FlyMaple** is our control board. More details about this board as well as the embedded system of this project can be found from the work of **Bùi Nhã Đạt** [10]. The autopilot calculation and the implementation of different control method can be found in the work of **Ana Castilo Benito** [11] In order to create the control firmware for the board, this project uses **ArduPilot**, a very popular framework to create autopilot firmware for UAV. The architecture of ArduPilot, its advantages, its drawbacks as well as the utilization of it in FlyMaple will be discussed in the next chapters of this report.

Chương 2

Cấu trúc của ArduPilot – Software architecture of ArduPilot

This part gives an overview about the structure of ArduPilot. It is showing how its layered software architecture has allowed its portability and how it relies on the native drivers of the target boards.

2.1 Ardupilot

2.1.1 Overview

“ArduPilot and ArduPilot Mega (APM) is an open source autopilot system supporting multi-copters, traditional helicopters, fixed wing aircraft and rovers” [11]. As an open source software, it is developed and contributed to by a large number of programmers around the world for many kind of drones, hardware as well as different versions of firmware. It is used worldwide as a common software platform for programming and controlling the autopilot systems.

The name ArduPilot comes from the combination of Arduino, which is a very popular company providing the board (hardware) and the language to communicate with the hardware (software), and autopilot.

At this time, ArduPilot supports Fixed-wing plane (ArduPlane), multicopters and traditional helicopters (ArduCopter) and ground vehicles (APMrover2). However, knowing the basics of this framework, users can create the autopilot for almost any unmanned vehicles. The full list of projects related to ArduPilot can be found in [11].

As mentioned, ArduPilot can be used in various hardware. Some specific main boards are being supported by the development team of ArduPilot: Pixhawk, PX4 FMU, FlyMaple, APM, etc. In this project, our quadricopter uses FlyMaple main board because it is available at the lab.

2.1.2 ArduPilot main code

ArduPilot includes more than seven hundred thousand code lines just for the basic code [12], which can be provided into five different sub-components as in **Figure 1-1**

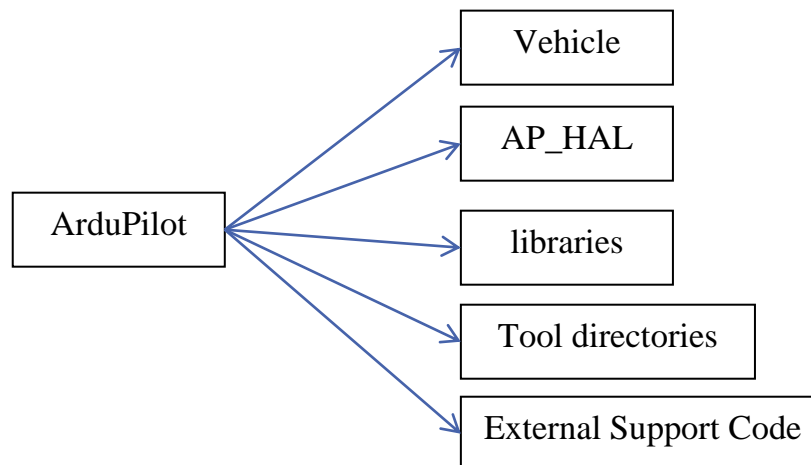


Figure 2-1: Structure of ArduPilot codebase

The first component is “**Vehicle**”, it contains any related code determining the unmanned autopilot systems, which are ArduPlane, ArduCopter and APM2rover, as mentioned above.

“**AP_HAL**” defines the code glue used to target specific main boards. Using Doxygen¹ to generate the documentation of ArduPilot, we can have **fig 2-2**.

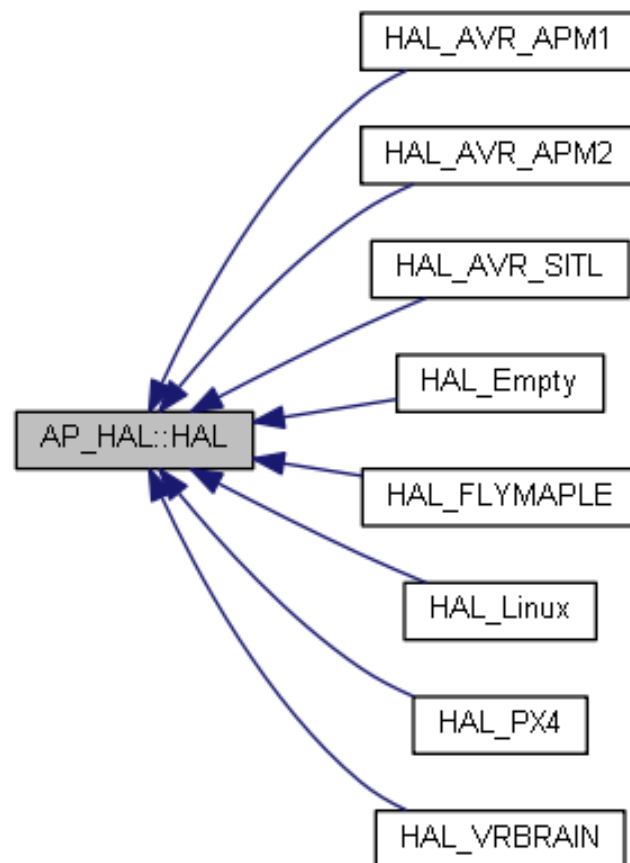


Figure 2-2: AP_HAL structure and the supported boards

From the figure above, we can see that ArduPilot supports various types of boards. Even if the board is not in the list, ArduPilot can be ported to it by specializing as “AP_HAL_EMPTY”.

¹ Doxygen is a tool for generating documentation from annotated C++ sources, but it also supports other popular programming languages such as C, Objective-C, C#, PHP, Java, Python, IDL (Corba, Microsoft, and UNO/OpenOffice flavors), Fortran, VHDL, Tcl, and to some extent D. More information about this tool can be found at <http://www.stack.nl/~dimitri/doxygen/>

“Libraries” are the most important part of this framework, giving the users the functions and codes to communicate with and control the board. For example, for a quadricopter, the libraries provide users the required tools to calculate the Pulse Width Modulation (PWM) signal to control the rotor, to communicate with the controller, GPS system, sonar systems, etc.

“Tools directories” gives users the tools to do some system checking (sensors, GPS...). It also includes the tools used for simulation, testing new control method. This simulation will be discussed later in this report. **“External Support Code”** supports code needed for new generation of main boards, such as Pixhawk.

2.1.3 Basic structure

ArduPilot is constructed as a multi-layers software platform. In concept, a multi-layer structure can be described as in **fig 2-3**.

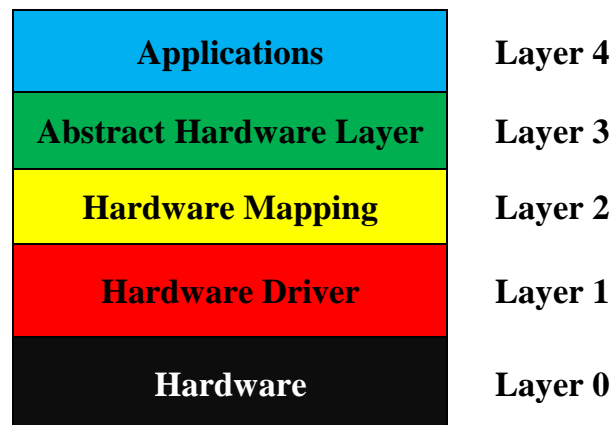


Figure 2-3: Multi-layers structure

Multi layered architectures are widely used in hardware independent application. One popular example is the **Open Systems Interconnection (OSI)** model, which is a conceptual model describing the common idea for many communicating systems, the internet for example. The idea of this model is that the system can be built from different physical components; allow different hardware can communicate with each other regardless the underlying structure. This is essential for a network, like the internet, because it does not limit the availability of the hardware. That is one of the most important benefits that make

communication systems like internet and mobile phone network has developed rapidly in a very short time.

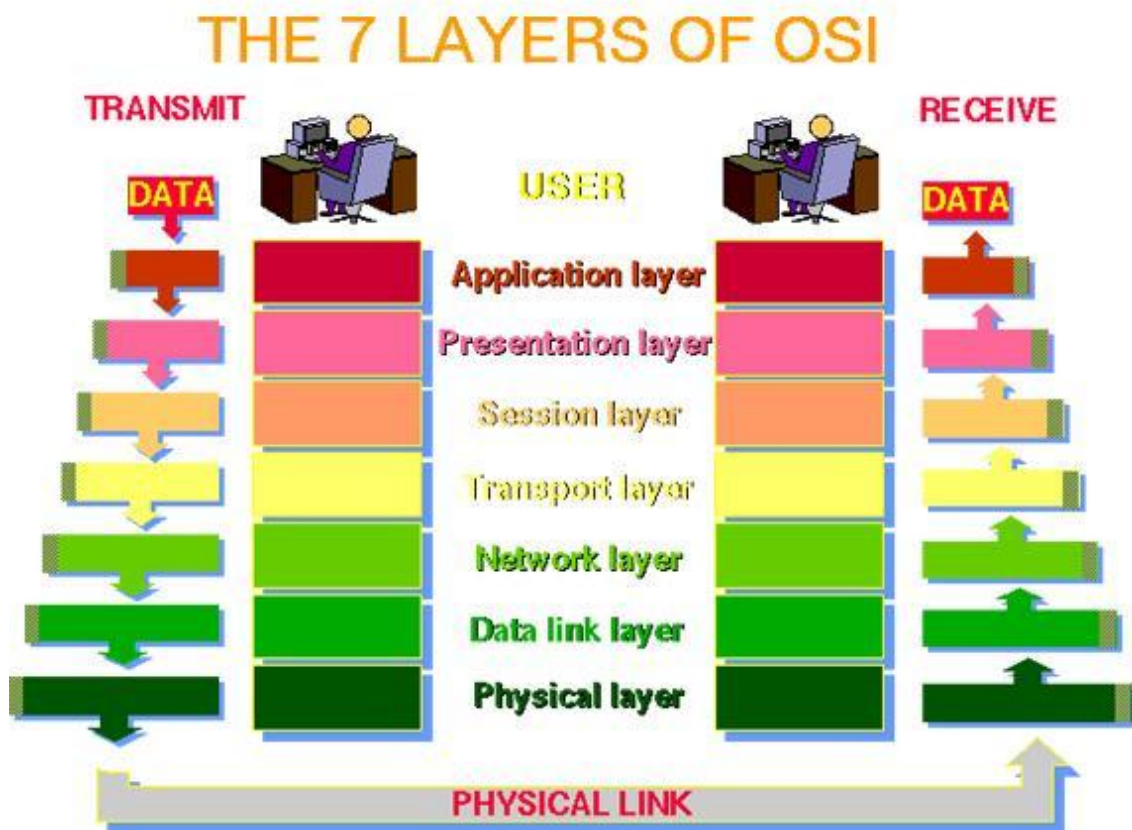


Figure 2-4: Multi layered structure of OSI model [14]

The principle of the multi layered structure is that the layer n uses the services provided by layer $n-1$ and then contribute its own services to the next $n+1$ layer. With the structure in the figure 2-3, **hardware** is **layer 0**, which is the physical layer. **Layer 1** is the Driver specific for the hardware in layer 0, using the facilities of the hardware and provides the next layer the required protocols to communication with the hardware. The next layer, **Hardware Mapping** will use these protocols and then offer its own services to the Abstract Hardware Layer, etc. The advantages of this architecture, as can be seen is that layer n is only depends on the services offered by layer $n-1$. In other words, **Application**, the highest layer, is independent with the **Hardware**. This will make the application becomes multi targets, reduce the amount of works needed to make a compatible version of that application for other hardware. If we replace the

hardware, which comes with its native drivers, only the hardware mapping layer has to be adapted for the application to work

Figure 2-5 presents the multi-layers structure of ArduPilot on a FlyMaple

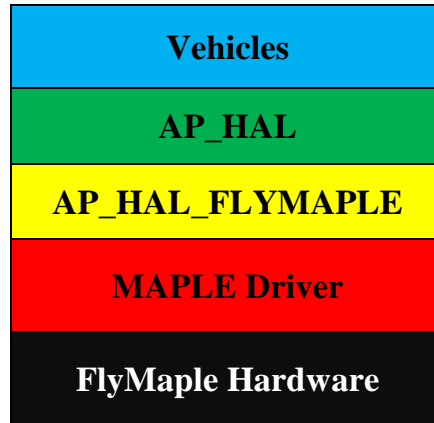


Figure 2-5: ArduPilot structure with FlyMaple hardware

The Vehicles layer includes the command for specific vehicle such as ArduPlane, ArduCopter. AP_HAL is the abstract layer, with the general functions in **figure 1-5**.

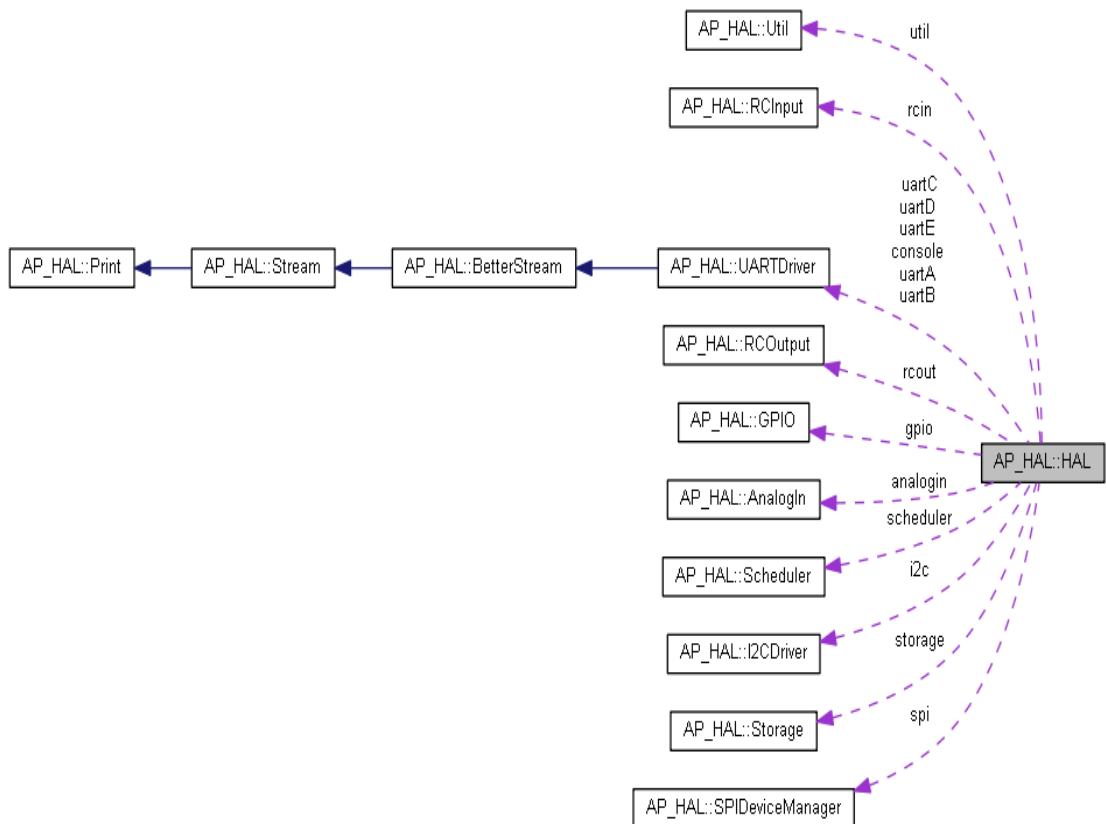


Figure 2-6: General functions in the abstract hardware layer

Programmers will program their autopilot functions based on this layer, with the name of the functions as defined in this layer. **AP_HAL_FLYMAPLE** connects the abstract systems in this layer to the correct place in FlyMaple board. As can be seen from **figure 2-6**, **AP_HAL** defines serial ports from **uartA** to **uartE** and the **console** as the communication channels and uses them to create the procedures for receiving and transmitting information between the board and other systems such as sensors, GPS, ground control station... In FlyMaple, there are only three communication channels, which is **SerialUSB**, **Serial1** and **Serial2** provided by the native drivers. As a Mapping Hardware layer, **AP_HAL_FLYMAPLE** provides a mapping for this situation (**fig 2-7**).

```
class HardwareSerial;
extern HardwareSerial Serial1; // Serial1 is labelled "COM1" on Flymaple pins 7 and 8
extern HardwareSerial Serial2; // Serial2 is Flymaple pins 0 and 1
extern HardwareSerial Serial3; // Serial3 is labelled "GPS" on Flymaple pins 29 and 30

static FLYMAPLEUARTDriver uartADriver(&Serial1); // AP Console and highspeed mavlink
static FLYMAPLEUARTDriver uartBDriver(&Serial2); // AP GPS connection
static FLYMAPLEUARTDriver uartCDriver(&Serial3); // Optional AP telemetry radio
```

Figure 2-7: Mapping in Harware Mapping Layer(HAL_FLYMAPLE_Class.cpp)

The code in **Fig 1-6** has two parts. The first part indicates the channels of the real hardware (FlyMaple), and the second part below it presents the ways connecting **uartA** to **uartC** with these channels. The **AP_HAL_FLYMAPLE** acts like glue code between native drivers and abstract hardware layer.

The Maple driver layer shown in **Fig 2-5** includes the driver files of FlyMaple, and these files are contained in specific libraries grouped in **libmaple**. In fact, **HardwareSerial** in **fig 1-6** is a class defined in this library.

Using the multi-layers structure, ArduPilot has the benefits as well as drawbacks of it. One of the most important advantages of the multi-layer is the portability. ArduPilot is suitable not only for many types of unmanned systems, but also for many different kinds of main board. The users do not have to change from a familiar programming language to another every time they change the

board, in other words, ArduPilot is hardware independent. This advantage can also help ArduPilot to be updated with any kind of modern board without changing so much.

Moreover, its multi layered structure makes ArduPilot easy to modify and change for the needs of a specific user. A programmer knowing the language can adjust it at the right level to achieve the desired targets without making many modifications.

However, for new users, ArduPilot is complex as the result of multi-layer structure. ArduPilot main code has more than seven hundred thousand code lines [12], and a new user cannot easily find the exact place to modify the code to do something simple without spending a lot of time. Additionally, the size of the constructed firmware, because it has to contain many related information, is large comparing with other one-target code. This is a major problem for old boards, or even modern complex control systems. The code efficiency is also a minus for multi-layer structure, as it cannot use **inline** functions in many cases in which it will minimize the size of the code and increase its performance. “*When you declare a function using the **inline** keyword, the compiler copies the body of the function into the calling function, making the copied instructions a part of the calling function as if it were written that way originally. The benefit to you is a slight improvement in performance.*” [13]. Although there are both pros and cons in using inline functions, for an autopilot systems, performance and code efficiency is very important, as they will affect the time the system need to do each tasks, and therefore affect the response time of each task in general.

2.1.4 ArduPilot threading in FlyMaple controller board.

ArduPilot is suitable for both old and new boards, this is also a disadvantage since it cannot use all of the advantages of the new one. A simple example for this is the threading/scheduling system in ArduPilot.

At the beginning, boards were not supporting multithreading, so as a default, ArduPilot supports these boards with a simple timer and callbacks called

a cyclic executive. The idea of this callback system is that any task that is registered in this timer system will be called at the rate of one kHz (one time each millisecond). In this one millisecond loop, each task will have a specific time to finish its mission, and then continue with the next task, sequentially.

Figure 2-8 is a description for this way of tasking.

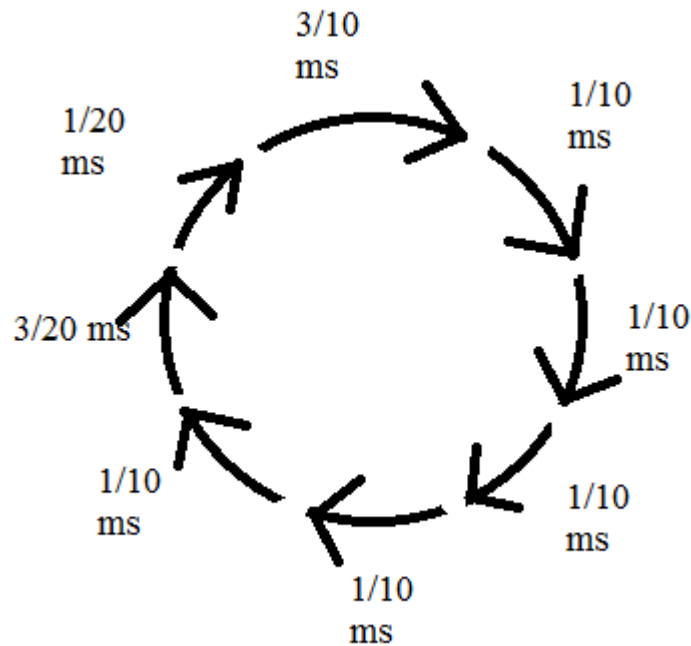
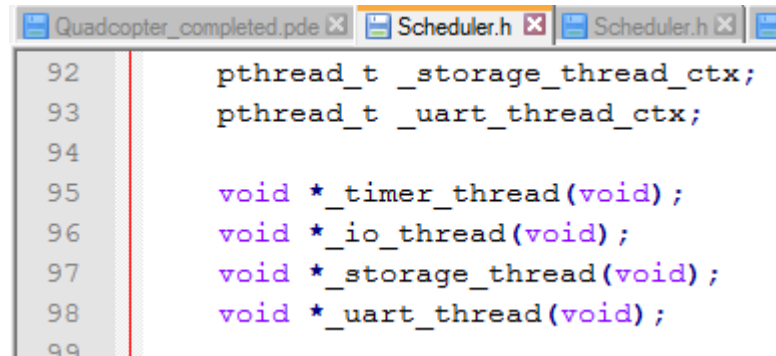


Figure 2-8: Tasking for a single thread processor in ArduPilot

Since controllers have to do a big amount of work, in fact no task is called at this one kHz rate. ArduPilot divides controller boards into two groups. The fast group (including PX4 and FlyMaple) has a MAIN_LOOP_RATE of 400 Hz, which means the fastest tasks can be called every 2.5 milliseconds. Meanwhile, slow group offers a 100Hz frequency.

For modern boards, like FlyMaple or Pixhawk, they not only can work at a much faster rate but also supports multi-threads. For these board, as described in [14], “*the threading approach in ArduPilot depends on the board it is build for*”. Besides the timer thread created for the single threading board as described above, at the AP-HAL abstract hardware layer, depends on the board, there are some more thread such as **UART thread** for communication and **I/O thread** for

tasks related to onboard memory. For example, in the **Scheduler.h** file for **AP_HAL_PX4**, as can be seen in **figure 1-8**, there are four separated threads.



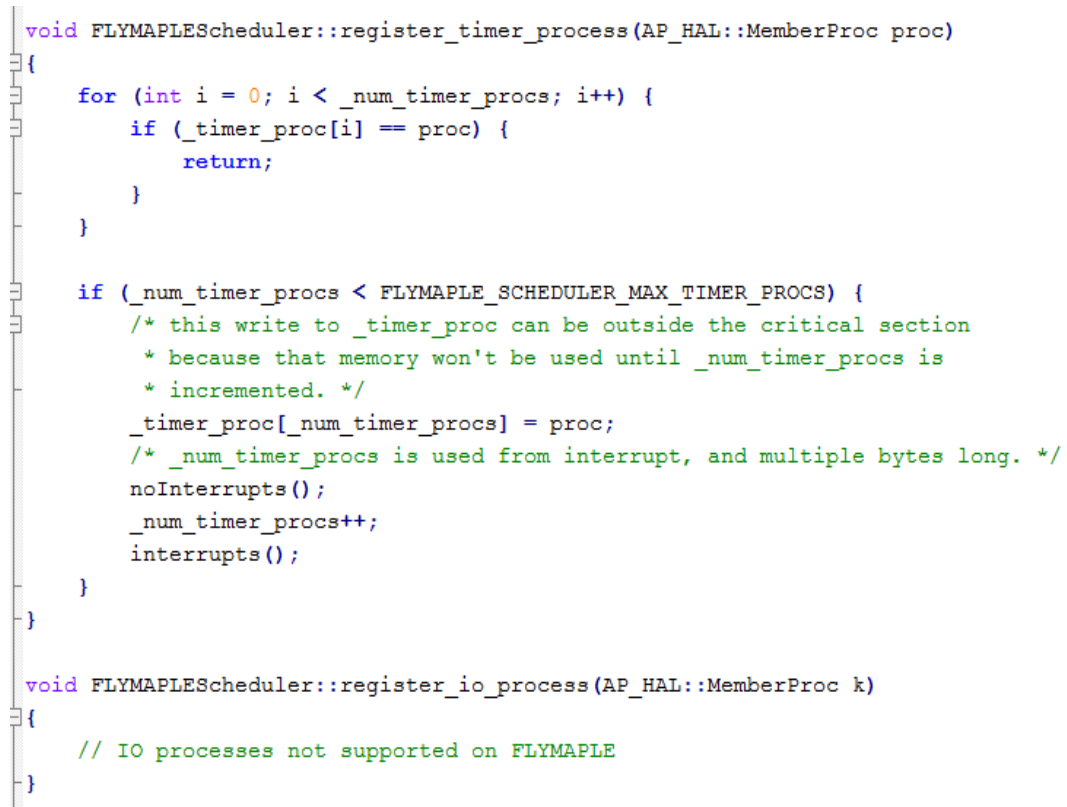
```

92 pthread_t _storage_thread_ctx;
93 pthread_t _uart_thread_ctx;
94
95 void *_timer_thread(void);
96 void *_io_thread(void);
97 void *_storage_thread(void);
98 void *_uart_thread(void);
99

```

Figure 2-9: Threading in PX4

In this case, four threads running and doing tasks concurrently, avoiding waste of time while polling I/O. For FlyMaple, however, although it is multithread supported and can use FreeRTOS for threading [10], ArduPilot does not fully provide these facilities. In fact, from the **Scheduler.h** file and the **Scheduler.cpp** file in **AP_HAL_FLYMAPLE**, there are no other threads besides the basic thread above.



```

void FLYMAPLEScheduler::register_timer_process(AP_HAL::MemberProc proc)
{
    for (int i = 0; i < _num_timer_procs; i++) {
        if (_timer_proc[i] == proc) {
            return;
        }
    }

    if (_num_timer_procs < FLYMAPLE_SCHEDULER_MAX_TIMER_PROCS) {
        /* this write to _timer_proc can be outside the critical section
        * because that memory won't be used until _num_timer_procs is
        * incremented. */
        _timer_proc[_num_timer_procs] = proc;
        /* _num_timer_procs is used from interrupt, and multiple bytes long. */
        noInterrupts();
        _num_timer_procs++;
        interrupts();
    }
}

void FLYMAPLEScheduler::register_io_process(AP_HAL::MemberProc k)
{
    // IO processes not supported on FLYMAPLE
}

```

Figure 2-10: FlyMaple threading in ArduPilot from Scheduler.cpp

Therefore, the only benefits of using ArduPilot for FlyMaple comparing to other older controller boards is that MAIN_LOOP_RATE of FlyMaple can run at the rate of 400Hz like PX4, which can control easily an autopilot system such as quadricopter. No multithreading is provided so far.

2.1.5 Building firmware for main board – Cross compiling

The code from the codebase, or the code created by the users, which can be read and understood by human, is not the kind of language a hardware, in this case a controller board, can understand. In order to change from programming language to the machine language, a process called “**build**” or “**compile**” must be done. In this case, it is a cross compiling, a process turning a source code (in this case C/C++ code) into executable code file, which can run in a different platform from the programming and the compile environment, called the host environment. For controller board, the output will be *.elf file, “*ELF (Executable and Linking Format) is a binary format originally developed by USL*” [15]. Cross-compiling is very important, in particularly for embedded systems, where the executive systems do not have enough resource to compile the running file itself, or programmers cannot write the particular code for that kind of language [16].

Step-by-step, the cross-compile from C/C++ source to executable file can be described in the figure below.

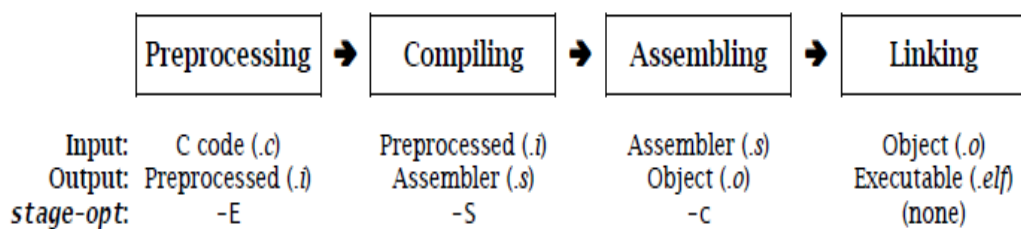


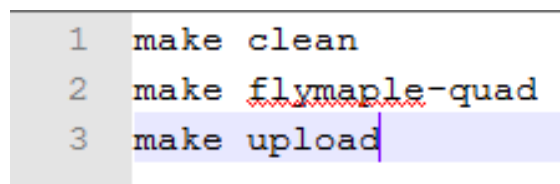
Figure 2-11: Stages of Compilation [17]

More detail of cross compiling along with the file types of ArduPilot will be discussed further in the next section of this report.

The result of this process is called **firmware**, “a software program or set of instructions programmed on a hardware device and provides the necessary instructions for how the device communicates with the other computer hardware” [18]. After being build, this firmware must be “**uploaded**” into the board.

There are several ways to build a firmware for ArduPilot code. One of them is using an **I**ntegrated **D**evelopment environment (IDE) such as **Eclipse** or **Emacs**. These programs not only provide the environment to build the firmware (the cross-compile tools), but also a working place to modify the code, find structure, etc. The modifications for these IDE to make them suitable for building ArduPilot code is in [19].

In general, making or building a firmware for any target needs three commands, as in **figure 1-10**.



```
1 make clean
2 make flymaple-quad
3 make upload
```

Figure 2-12: Commands to compile upload to main board in Terminal

The first command in **Figure 1-4** will clean any unneeded old firmware as well as the intermediate files created by a previous compilation. Meanwhile the second one will build a specific firmware for **FlyMaple** board using to control the Quadricopter. After finish building the firmware, the last command will upload it into the board. **Figure 2-13** below presents the process of uploading a built firmware indicated above. During uploading, instead of changing the board to perpetual boot loader mode every time as described in [10], we can simply quickly push the reset button at the right time after entering “make upload” command. During restart, the FlyMaple will have a few second in the boot loader mode and will receive the uploaded firmware [20].

A terminal window with a dark purple background and white text. The title bar shows 'quangnguyen@ubuntu: ~/ardupilot/ArduCopter'. The output of the 'dfu-util 0.5' command is displayed. It includes copyright information for Weston Schmidt, Harald Welte, OpenMoko Inc., and Tormod Volden. It states that the program is free software with absolutely no warranty. The output shows that dfu-util only supports DFU version 1.0. It then shows the process of opening a DFU USB device (ID 1eaf:0003), finding the DFU, claiming the USB interface, setting alternate settings, determining device status, and finally copying data from the PC to the DFU device. The process ends with 'Starting download: [#####]'.

```
quangnguyen@ubuntu: ~/ardupilot/ArduCopter
dfu-util 0.5

(C) 2005-2008 by Weston Schmidt, Harald Welte and OpenMoko Inc.
(C) 2010-2011 Tormod Volden (DfuSe support)
This program is Free Software and has ABSOLUTELY NO WARRANTY

dfu-util does currently only support DFU version 1.0

Filter on vendor = 0x1eaf product = 0x0003
Opening DFU USB device... ID 1eaf:0003
Run-time device DFU version 0001
Found DFU: [1eaf:0003] devnum=0, cfg=1, intf=0, alt=1, name="DFU Program FLASH 0
x08005000"
Claiming USB DFU Interface...
Setting Alternate Setting #1 ...
Determining device status: state = dfuIDLE, status = 0
dfuIDLE, continuing
DFU mode device DFU version 0001
Device returned transfer size 2048
No valid DFU suffix signature
Warning: File has no DFU suffix
bytes_per_hash=5181
Copying data from PC to DFU device
Starting download: [#####]
```

Figure 2-13: Build successfully and uploading firmware

2.1.6 GNU make

As discussed above, compiling the firmware is using “make”. Like any other open source program, which means thousands of code lines, many developers and hundred source files, ArduPilot uses “make” as a solution to build the target program, in this case a firmware. As mentioned in [21], one important mission of makefiles is to define the dependency between source files, so that the compiling process with “make” will build in an order that satisfies these dependencies. This process is very important in case of a change and need to recompile, “make” can just recompile the changed file and any other parts that are related to it. This report will mostly focus on GNU make, which was implemented by Richard Stallman and Roland McGrath [22], since ArduPilot uses this particular make process.

The files structure of ArduPilot includes three different types of file. All of the files in the library, which define the abstract hardware layer, mapping hardware layer as well as particular functions, are *.h files and *.cpp files. Meanwhile, in ArduCopter folder, there are *.h and *.pde files.

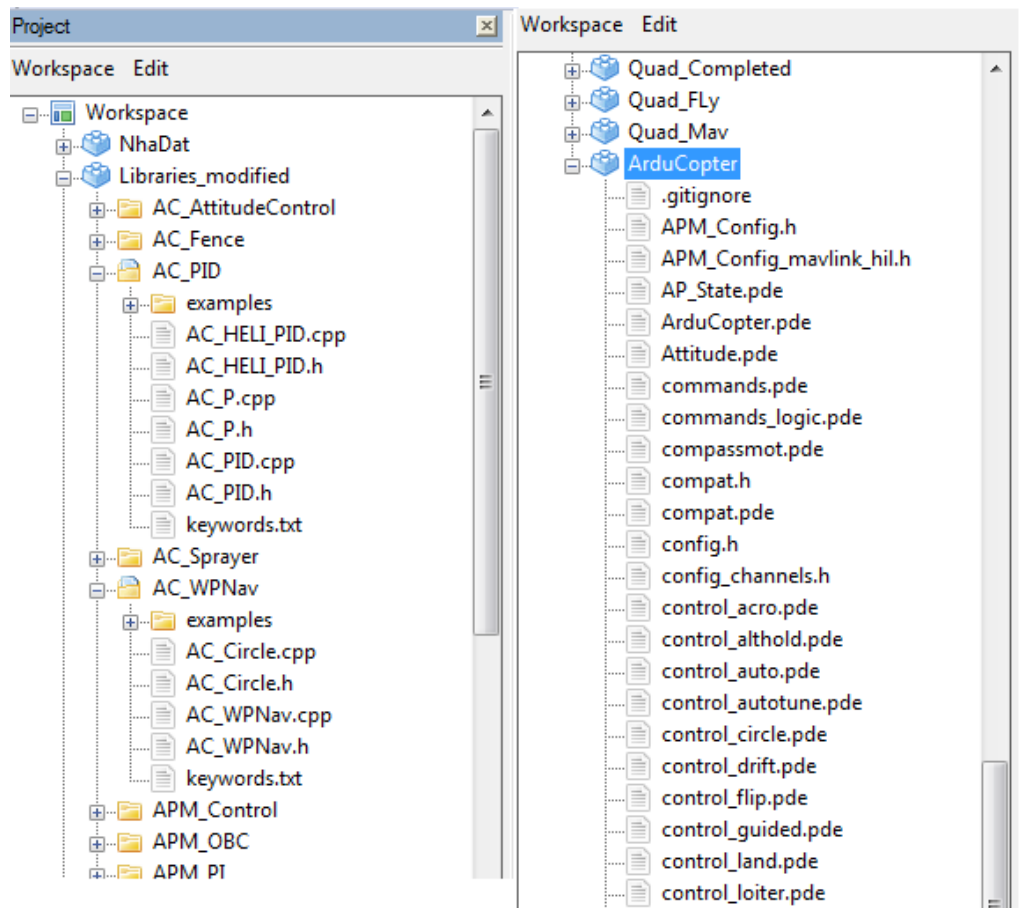


Figure 2-14: ArduPilot Files

*.h files and *.cpp files are standard files for C/C++ programming language. In short, *.h files is the header files, includes any class name, function name, variables and constants. *.cpp is where the classes and functions in *.h are implemented. *.pde is the file format of Arduino language, which is the base language of ArduPilot, these files will be preprocessed into *.cpp files.

In the Libmaple – a source code with drivers for FlyMaple - there are some other files types such as *.c and *.o. The *.c is format of C language, meanwhile, *.o files are the result of compilation from a source code. They will be added into compile process by a config.mk file with the structure as followed.

```

1  #config.mk START
2  # Select maple_RET6 for Flymaple
3  BOARD = maple_RET6
4  # HAL_BOARD determines default HAL target.
5  HAL_BOARD ?= HAL_BOARD_FLYMAPLE
6  # The communication port used to communicate with the Flymaple
7  PORT = /dev/ttyACM0
8  # You must provide the path to the libmaple library directory:
9  LIBMAPLE_PATH = $(HOME)/Learning_ArduPilot/libmaple
10 # Also, the ARM compiler tools MUST be in your current PATH like:
11 # export PATH=$PATH:~/Learning_ArduPilot/libmaple/arm/bin
12 #config.mk END
13

```

Figure 2-15: config.mk structure pointing out the necessary drivers for FlyMaple [23]

In order to cross compile, many rules, targets as well as building orders have to be defined. This structure is implemented into several *.mk files in mk directory of ArduPilot. From the definitions in these files, **figure 2-16** and **figure 2-17** can be used to present the “make” process in ArduPilot.

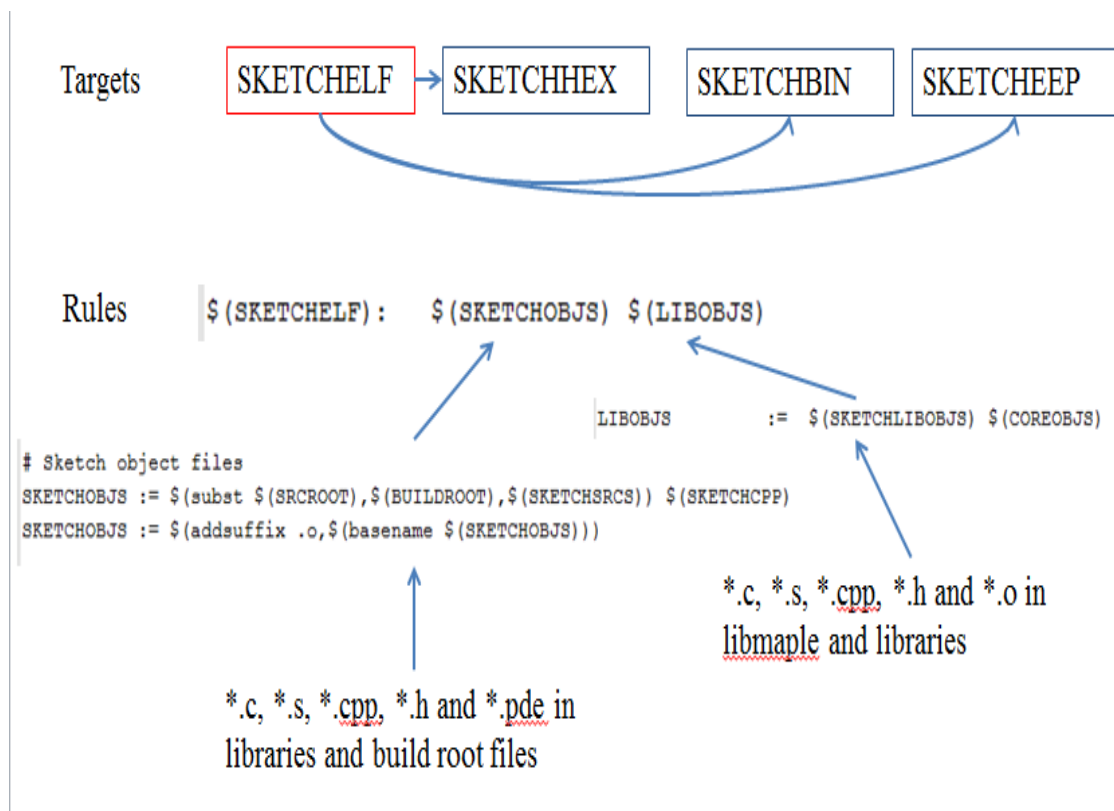


Figure 2-16: Summary of “make” process by targets and rules

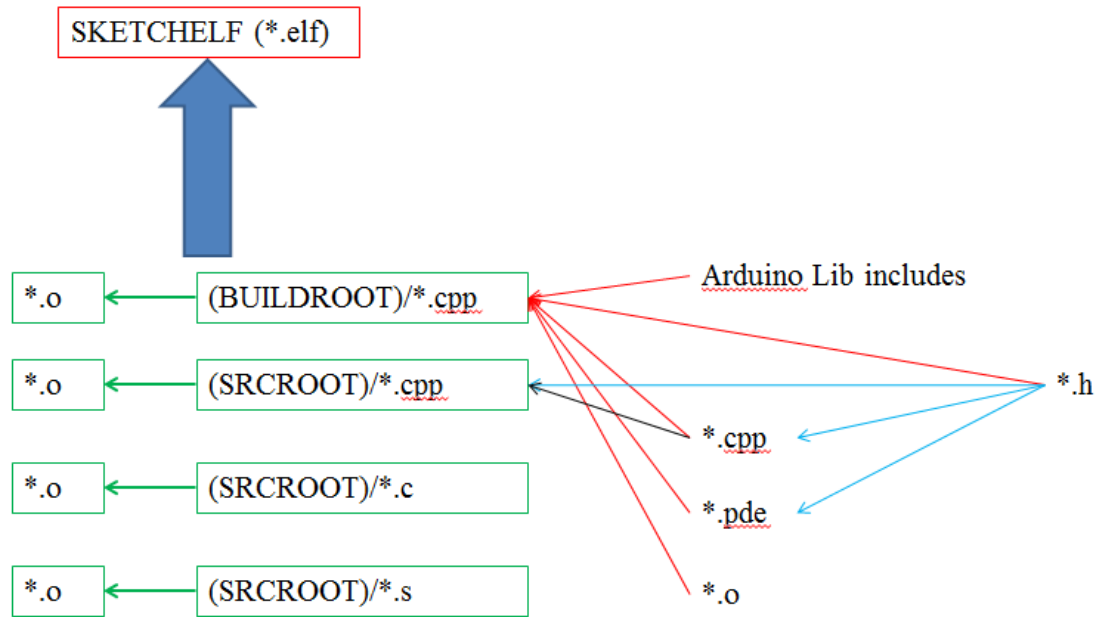


Figure 2-17: Summary of “make” process by objects and file formats

These figures, along with **figure 2-11**, give an overview look about GNU make process and the dependency of ArduPilot cross compile, how the firmware is made. Knowing the basics of ArduPilot as presented is very important to handle the problems of using ArduPilot in FlyMaple, which will be presented in the next part of this report.

2.2 ArduPilot problems and suggested solution

2.2.1 Problems

Although ArduPilot indicated an instruction to build ArduPilot code for FlyMaple [23], make an executable firmware for FlyMaple using ArduPilot structure, in fact, there are still some problems between the code and the hardware.

The first problem, as has been pointed out, is the threading and will not be discussed in this part. The next problem is more important, since it affect directly the way the embedded system communicate with other systems, in particularly the Ground Control Station, via wifi Connection in this project.

In a communication test between FlyMaple and Ground Control Station, strange phenomenon occurred, as described in figure 2-18. Usually, this error happens because of using a wrong version of MAVLink protocol, a communication standard for UAV which will discussed in another part of this report. However, in this case, either MAVLink Version or Baud Rate is not the problem.

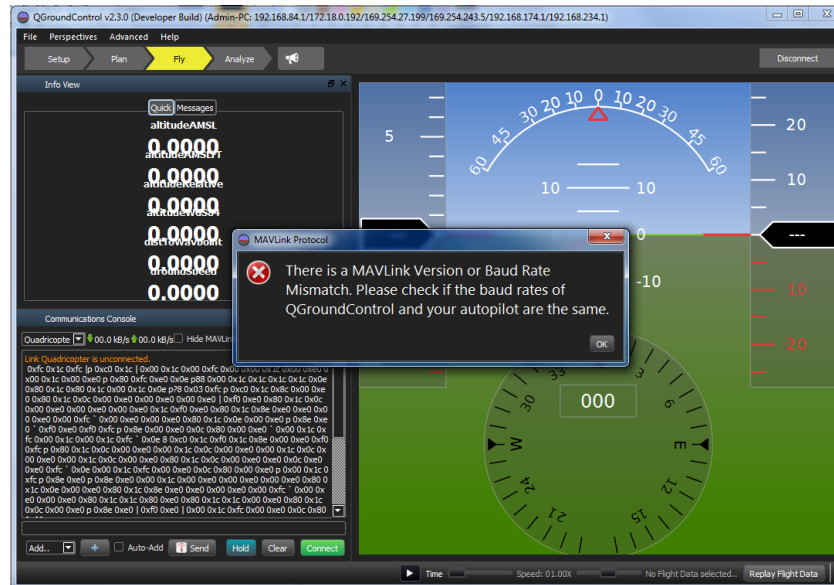


Figure 2-18: Problem with the original ArduPilot code

During tests, another problem was also noticed. As can be seen from **figure 2-19**, while communicating between two systems, many packets have been dropped, this phenomena lead to the problem of lost connection between ground control station and autopilot.



Figure 2-19: Lost packets problem

2.2.2 Solution

With the help of my supervisor, the problem has been located in the Mapping Hardware Layer, in particularly the file called **HAL_FLYMAPLE_Class.cpp** shown in **figure 2-7**. In the original code, the mapping can be described as follow

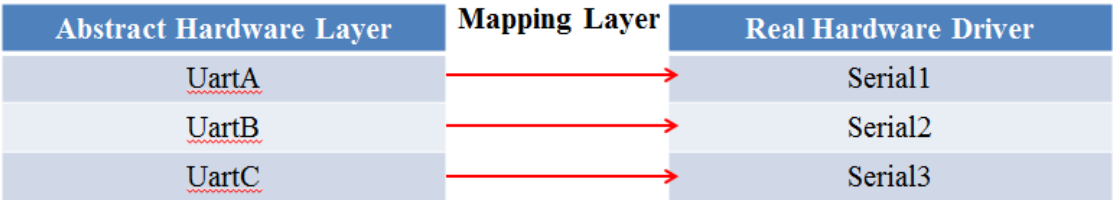


Figure 2-20: Original Mapping Layer

By default, ArduPilot defines **UartA** as the serial port for Mavlink communication and console, **UartB** is used for GPS. Meanwhile, as described in the work of Nhã Đạt, for the embedded system in this project, **Serial1** of FlyMaple is for GPS and **Serial2** is used in wifi communication [10]. In order to fix the problems showed above, a simple modification will be made in this file. By changing two simple numbers as can be seen in **figure 2-21**, a new mapping layer has been created for the FlyMaple using in this project.

```
static FLYMAPLEUARTDriver uartADriver(&Serial2); // AP Console and highspeed mavlink
static FLYMAPLEUARTDriver uartBDriver(&Serial1); // AP GPS connection
static FLYMAPLEUARTDriver uartCDriver(&Serial3); // Optional AP telemetry radio
```

Figure 2-21: New mapping layer for this project

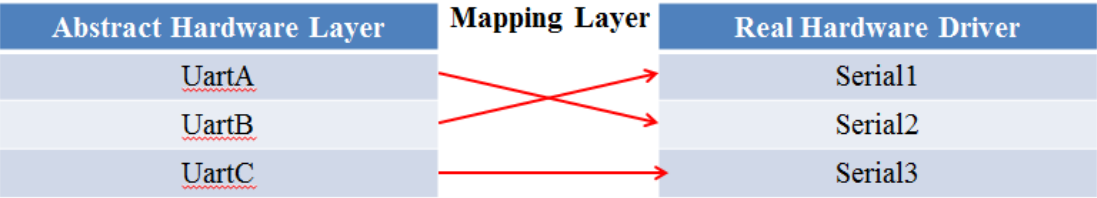


Figure 2-22: New mapping for the FlyMaple in this project

In a multi layered architecture description, the modification which has just been done is the change in the Hardware Mapping Layer. As mentioned above, this change will not affect the default settings or any other parts of ArduPilot. This is

the crucial point since this modification can be done with any version of ArduPilot, provide the embedded system used in this project to be carry on with the newest version of ArduPilot.

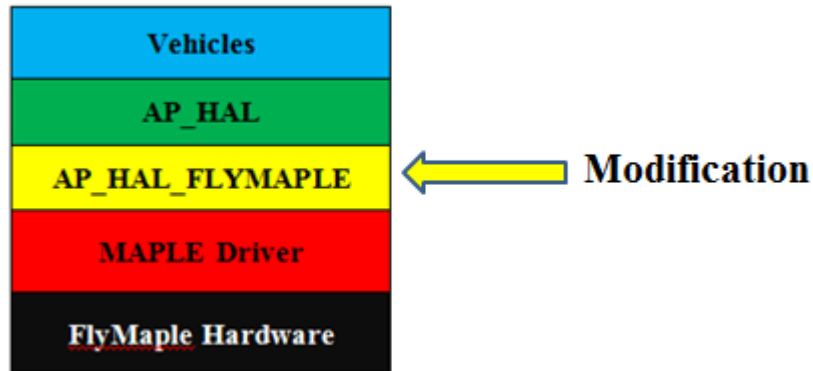


Figure 2-23: Modification for FlyMaple in this project

More importantly, this modification is suitable for the idea of a multi layered architecture application, which is adapting the mapping layer for specific hardware.

References

- [1] US Department of Defense, *Dictionary of Military and Associated Terms.*, 2005. [Online].
<http://www.thefreedictionary.com/Unmanned+Aerial+Vehicle>
- [2] H.McDaid et al. (2013, Feb) Remote Piloted Aerial Vehicles : An Anthology. [Online]. http://www.ctie.monash.edu/hargrave/rpav_home.html#Beginnings
- [3] J.F Kean and S.S.Carr, "A brief History of Early Unmanned Aircraft ," *Johns Hopkins APL Technical Digest*, vol. 32, pp. 558-571, 2013.
- [4] (2005, July) The Radioplane Target Drone. [Online].
http://www.ctie.monash.edu.au/hargrave/rpav_radioplane4.html
- [5] Â Frost and Sullivan, "Study Analysing The Current Activities in The Field of UAV," European CommisionÂ Enterprise and Industry Directorate-General, Technical Report 2007. [Online].
<https://engineering.purdue.edu/HSL/index.php?page=numerical-analysis-of-cyberattacks>
- [6] THE NEW YORK TIMES. (2010) Elmer Sperry Dies; Famous Inventor (Jun 1930). [Online].
<http://www.nytimes.com/learning/general/onthisday/bday/1012.html>
- [7] William Scheck. (2006, Dec) Lawrence Sperry: Autopilot Inventor and Aviation Innovator. [Online]. <http://www.historynet.com/lawrence-sperry-autopilot-inventor-and-aviation-innovator.htm>

- [8] "Now - The Automatic Pilot," *Popular Science Monthly*, p. 22, Feb 1930.
[Online].
http://books.google.fr/books?id=4ykDAAAAMBAJ&pg=PA22&redir_esc=y#v=onepage&q&f=false
- [9] William Scheck. Development of Aviation Technology. [Online].
<http://www.century-of-flight.net/Aviation%20history/evolution%20of%20technology/autopilot.htm>
- [1 N.Đạt BUI, "Embedded System for Quadricopter," Ho Chi Minh City
0] University of Technology, Internship Report 2014.
- [1 A.Castillo Benito, "Flight Control and Navigation of a Quadcopter," ISAE-
1] ENSMA, Poitiers, Internship Report 2014.
- [1 (2015) Welcome to the ArduPilot/APM development site. [Online].
2] <http://dev.ardupilot.com/>
- [1 (2015) Learning the ArduPilot Codebase. [Online].
3] <http://dev.ardupilot.com/wiki/learning-the-ardupilot-codebase/>
- [1 Daniel Petri. (2009, Aug) OSI Model Concepts. [Online].
4] http://www.petri.com/osi_concepts.htm
- [1 A.D. Marshall, "New Feature of C++," in *Hands on: C++ Programming.*,
5] 1998-2004, p. 24.
- [1 (2015) Learning ArduPilot - Threading. [Online].
6] <http://dev.ardupilot.com/wiki/learning-the-ardupilot-codebase/learning-ardupilot-threading/>
- [1 Daniel Barlow. (1996) What is ELF? An introduction. [Online].
7] http://cs.mipt.ru/docs/comp/eng/os/linux/howto/howto_english/elf/elf-howto-1.html
- [1 Introduction to cross-compiling for Linux. [Online].

- 8] <http://landley.net/writing/docs/cross-compiling.html#footnote1>
- [1 EmSys:An Introduction to the GNU Compiler. [Online].
- 9] http://shukra.cedt.iisc.ernet.in/edwiki/EmSys:An_Introduction_to_the_GNU_Compiler
- [2 (2015) The Tech Terms Computer Dictionary. [Online].
- 0] <http://techterms.com/definition/firmware>
- [2 (2015) Building the code. [Online]. <http://dev.ardupilot.com/wiki/building-the-code/>
- [2 (2015) Maple cant find dfu device. [Online].
- 2] <https://www.olimex.com/forum/index.php?topic=465.0>
- [2 (2015) GNU make in detail for beginnners. [Online].
- 3] <http://www.opensourceforu.com/2012/06/gnu-make-in-detail-for-beginners/>
- [2 R.McGrath, P.D.Smith R.M.Stallman. (2014, September) GNU Make.
- 4]
- [2 Building ArduPilot for FlyMaple in Linux. [Online].
- 5] <http://dev.ardupilot.com/wiki/building-the-code/building-apm-for-flymaple/>
- [2 (2015) Learning ArduPilot UARTs and the console. [Online].
- 6] <http://dev.ardupilot.com/wiki/learning-the-ardupilot-codebase/learning-ardupilot-uarts-and-the-console/>
- [2 GitHub ArduPilot. [Online].
- 7] https://github.com/diydrones/ardupilot/blob/master/libraries/AP_HAL/examples/UART_test/UART_test.pde
- [2 Joint Education and Doctrine Division, *DOD Dictionary of Military Term.*, 8] 2015.