International Journal of Internet, Broadcasting and Communication Vol.14 No.1 162-169 (2022)
http://dx.doi.org/10.7236/IJIBC.2022.14.1.162

IJIBC 22-1-20

Framework design for efficient Arduino program development

Dong-Hwan Gong

Assistant Professor, Department of Computer Engineering, Hansei University, Korea
armyvision@hansei.ac.kr

Abstract

Arduino is used in various places such as education, experimentation, and industry. Due to the easy accessibility of Arduino, it is often used by non-majors, and it is also used in media art and toy programs. Although Arduino is relatively easy to use compared to other devices, it is not easy to control various IoT components at the same time. Some tasks run independently of other tasks, while others run dependently. In this paper, I proposed the Arduino Task Framework to efficiently execute many tasks in these various situations. The design framework of this paper is largely composed of two types: synchronous execution and asynchronous execution. These two execution methods can be combined to create several independent and dependent execution routines. Asynchronous tasks are independently executed tasks and are managed by AsyncTaskGroup, while synchronous tasks are dependently executed tasks and are managed by SyncTaskGroup. AsyncTaskGroup instance and SyncTaskGroup instance are instances of the same Task and can be used in combination with another task. The Arduino framework proposed in this paper simplifies the program structure and can easily compose various tasks.

Keywords: *Arduino, Framework, IoT Framework, Task, Multitasking, Synchronous, Asnchronous, IoT Device Framework, Open source Hardware.*

1. Introduction

Most IoT devices receive control signals from the IoT platform and communicate with various sensors. It receives data from sensors and sends it to the IoT cloud, or receives data from the IoT cloud and sends the data to an output device[8]. Arduino requires multitasking to perform these various tasks[2-3]. Multitasking consists of a combination of synchronous and asynchronous tasks. In addition, the Arduino executes several independent tasks to process the various types of data received[1].

If the IoT device input/output framework is configured incorrectly, input/output side effects that cause problems in data processing occur. These side effects lead to inefficiencies that shorten the lifespan of the device or prevent normal performance[4]. Therefore, it is important to configure the IoT device program with low specification performance simply and efficiently. The IoT device communicates with the remote device via Bluetooth and with the cloud IoT server via Wi-Fi. At the same time, when Bluetooth data transmission/reception is completed, the LED is turned on, and when Wi-Fi communication is completed, the completion information is output on the LCD. At this time, Bluetooth communication and Wi-Fi

Manuscript Received: December. 25, 2021 / Revised: December. 28, 2021 / Accepted: January. 3, 2022
Corresponding Author: armyvision@hansei.ac.kr
Tel: +82-31-450-5340, Fax: +82-31-450-5172
Assistant Professor, Department of Computer Engineering, Hansei University, Korea

communication are independent and asynchronous operations[6-7]. When Bluetooth communication is completed, the operation of turning on the LED is a sequential and synchronous operation[9]. When the Wi-Fi communication is completed, the operation of outputting the completion information to the LCD is also a sequential and synchronous operation. Therefore, it is important to systematically organize asynchronous and synchronous operations[5]. A common Arduino framework is ArduinoComponents. ArduinoComponents gives you base components and a framework for writing component and event based code for any Arduino supported microcontroller with one unified interface. ArduinoComponents gives your the base Component class and many usefull utilities[9]. But ArduinoComponents have a complex structure.

The design framework of this paper is largely composed of two types of execution methods: synchronous execution and asynchronous execution. These two execution methods can be combined to create several independent and dependent execution routines. Asynchronous tasks are independently executed tasks and are managed by AsyncTaskGroup, while synchronous tasks are dependently executed tasks and are managed by SyncTaskGroup. AsyncTaskGroup instance and SyncTaskGroup instance are instances of the same Task and can be used in combination with another task[10].

2. Proposed framework

Figure 1 shows the class diagram of the designed framework. Task is an abstract class of an asynchronous task, and it becomes a base class that repeats at a certain time and operates. The callback attribute of the Task class is a task for Task to execute, and the client registers as a callback function.The main functions of Task are available() and run() methods. The method available() checks the time (ms) value set in the interval attribute, and the method run() executes the task (callback) when the runnable state (the time set in the interval attribute) is reached. AsyncTaskGroup class has add() and remove() methods to register and remove Task instances to taskList collection. The AsyncTaskGroup class also inherits the Task class and operates as an independent task. The SyncTaskGroup class also inherits the Task class and operates as an independent task, and has a TaskQueue instance as a property for synchronous task execution. SyncTaskGroup also has add() method and remove() method for registering and removing ISyncTask common interface of synchronous task. Synchronous tasks registered in SyncTaskGroup guarantee sequential processing in TaskQueue. What is important in synchronous tasks is a method to waiting the execution of other tasks for a certain period of time without interfering with the execution of other tasks. Because if it operates like Arduino's delay() function to wait for a certain amount of time, it affects all other asynchronous tasks. It guarantees independence from other asynchronous tasks and defines DelayTask class for synchronous waiting. DelayTask class is a class that implements ISyncTask, a common interface for synchronous tasks, the start() method starts a synchronous task and waits while wait() is true. The SyncTask class has a callback property to save and call the client's synchronous task, and works the same as DelayTask.

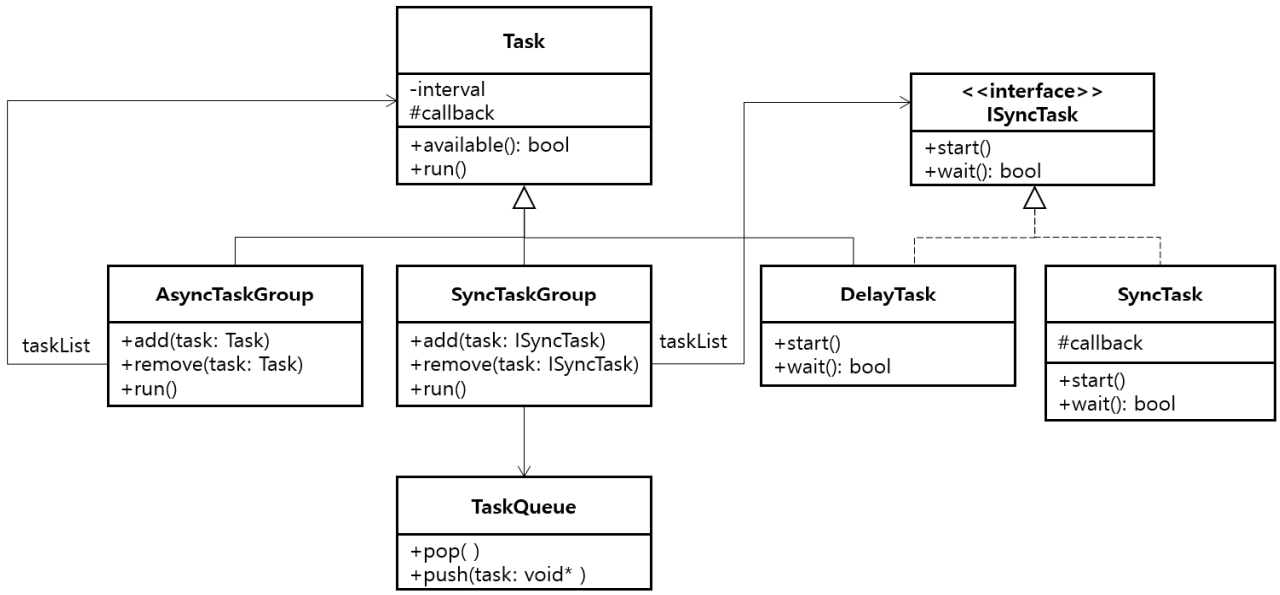

Figure 1. The class diagram of the designed framework

Figure 2 shows some codes of AsyncTaskGroup class and SyncTaskGroup class. Both classes are implemented by overriding the virtual method run() of Task. The run() method of the AsyncTaskGroup class calls all run() method of the taskList collection, and the run() method of the SyncTaskGroup class is stored in the TaskQueue and executed sequentially.

```
class AsyncTaskGroup : Task
{
    enum { TASK_MAX=20 };
    Task* taskList[TASK_MAX];
    int size;
public:
    AsyncTaskGroup(unsigned long iv = 0) {
        size = 0;
        setInterval(iv);
    }
    ~AsyncTaskGroup() {
        for (int i = 0; i < size; ++i) {
            delete taskList[i];
            taskList[i] = 0;
        }
    }
    void run()
    {
        Task::run();

        unsigned long time = millis();
        for (int i = 0; i < size; i++) {
            if (taskList[i]->available(time)) {
                taskList[i]->run();
            }
        }
        oldTime = millis();
    }
```

```
class SyncTaskGroup : public Task
{
    enum { TASK_MAX=100 };
    ISyncTask* taskList[TASK_MAX] = { 0 };
    ISyncTask* curTask;
    TaskQueue queue;
    int size;
public:
    SyncTaskGroup(unsigned long iv = 0) {
        curTask = NULL;
        size = 0;
        setInterval(iv);
    }
    ~SyncTaskGroup() {
        for (int i = 0; i < size; ++i) {
            delete taskList[i]; taskList[i] = 0;
        }
    }
    void run() {
        Task::run();
        if (queue.empty()) {
            for (int i = 0; i < size; ++i)
                queue.push(taskList[i]);
        }
        if( curTask == NULL ) {
            curTask = static_cast<ISyncTask*>(queue.pop());
            curTask->start();
        }
        if (!curTask->wait())
            curTask = NULL;
    }
```

Figure 2. Some codes of AsyncTaskGroup class and SyncTaskGroup class

3. Framework test

Figure 3 shows the code that executes asynchronous task on the PC. The cbAsyncTask1() function is a client-independent task for execution and is passed as the first argument of the Task object. The second argument of the Task object is the time (ms) to repeat this task, and the code is executed every 2 seconds.

```
#include "TaskFramework.h"
void cbAsyncTask1() {
    cout << " async task 1: " <<
        GetTickCount() << endl;
}
int main()
{
    Task task(cbAsyncTask1, 2000);

    while (1)
    {
        if(task.available())
            task.run();
    }
}
```

```
F:\taskframework\TaskFrameworkApp
async task 1: 172308359
async task 1: 172310359
async task 1: 172312359
async task 1: 172314359
async task 1: 172316359
async task 1: 172318359
async task 1: 172320359
async task 1: 172322359
async task 1: 172324359
async task 1: 172326359
async task 1: 172328359
async task 1: 172330359
async task 1: 172332359
async task 1: 172334359
async task 1: 172336359
```

Figure 3. The code that executes asynchronous task on the PC

Figure 4 shows three asynchronous tasks executed on PC. Independent client task cbAsyncTask1(), cbAsyncTask2(), cbAsyncTask3() functions are created as Task objects and registered as AsyncTaskGruop objects. The three tasks run independently, and task 1 runs every 1 second, task 2 runs every 2 seconds, task 3 runs every 3 seconds. If you check the fourth digit for the output time in milliseconds, you can check that it is independently executed at the same time.

```
#include "TaskFramework.h"
void cbAsyncTask1() {
    cout << " async task 1: " <<GetTickCount()
}
void cbAsyncTask2() {
    cout << " async task 2: " <<GetTickCount()
}
void cbAsyncTask3() {
    cout << " async task 3: " <<GetTickCount()
}
int main()
{
    AsyncTaskGroup atGroup;
    atGroup.add(new Task(cbAsyncTask1, 1000));
    atGroup.add(new Task(cbAsyncTask2, 2000));
    atGroup.add(new Task(cbAsyncTask3, 3000));

    while (1)
    {
        atGroup.run();
    }
}
```

```
F:\taskframework\TaskFrameworkApp
async task 2: 174329109
async task 1: 174329125
async task 1: 174330125
async task 2: 174331109
async task 1: 174331125
async task 3: 174331125
async task 1: 174332125
async task 2: 174333109
async task 1: 174333125
async task 1: 174334125
async task 3: 174334125
async task 2: 174335109
async task 1: 174335125
async task 1: 174336125
async task 2: 174337109
async task 1: 174337125
async task 3: 174337125
async task 1: 174338125
async task 2: 174339109
async task 1: 174339125
async task 3: 174340125
async task 1: 174340140
async task 2: 174341125
async task 1: 174341140
async task 1: 174342140
async task 2: 174343125
async task 3: 174343125
async task 1: 174343140
async task 1: 174344140
```

Figure 4. Three asynchronous tasks executed on PC

Figure 5 shows three synchronous tasks and three standby tasks performed on the PC. Task 1 is executed and task 2 is executed 1 second later. Task2 is executed and task3 is executed 2 seconds later. Since Task3 was executed and all tasks were executed 3 seconds later, repeat from task 1. In this case, since all three synchronous tasks (tasks 1, 2, 3) and three standby tasks (1 second, 2 seconds, 3 seconds) are executed sequentially, the order does not change. If the synchronous operation is performed below 1 ms, it takes a total of 6 seconds because it takes time only for the standby operation. Sequential tasks are registered as SyncTaskGroup objects to run synchronously. Synchronous group registration is registered by calling the add() method, and the order of synchronous operations is important, so it must be registered in the execution order. The run() method of the SyncTaskGroup object ensures that registered synchronous tasks are executed in order using TaskQueue.

```
#include "TaskFramework.h"
void cbSyncTask1() {
    cout << " sync task 1: " << GetTickCount()
}
void cbSyncTask2() {
    cout << " sync task 2: " << GetTickCount()
}
void cbSyncTask3() {
    cout << " sync task 3: " << GetTickCount()
}
int main()
{
    SyncTaskGroup stGroup;
    stGroup.add(new SyncTask(cbSyncTask1, 0));
    stGroup.add(new DelayTask(1000));
    stGroup.add(new SyncTask(cbSyncTask2, 0));
    stGroup.add(new DelayTask(2000));
    stGroup.add(new SyncTask(cbSyncTask3, 0));
    stGroup.add(new DelayTask(3000));

    while (1)
    {
        stGroup.run();
    }
}
```

```
F:\#taskframework\#TaskFrameworkApp
sync task 2: 175539375
sync task 3: 175541375
sync task 1: 175544375
sync task 2: 175545375
sync task 3: 175547375
sync task 1: 175550375
sync task 2: 175551375
sync task 3: 175553375
sync task 1: 175556375
sync task 2: 175557375
sync task 3: 175559375
sync task 1: 175562375
sync task 2: 175563375
sync task 3: 175565375
sync task 1: 175568375
sync task 2: 175569375
sync task 3: 175571375
sync task 1: 175574375
sync task 2: 175575375
sync task 3: 175577375
sync task 1: 175580375
sync task 2: 175581687
sync task 3: 175583781
sync task 1: 175586859
sync task 2: 175587921
sync task 3: 175590765
sync task 1: 175594078
sync task 2: 175595109
sync task 3: 175597140
```

Figure 5. Three synchronous tasks and three wait tasks

4. Framework Results

For checking whether the framework designed in the Arduino program works well, executed 3 independent tasks. The first task is to periodically output data to the serial device, the second task is to receive data when the user sends data to the Bluetooth in real time, and the third task is to turn the LED on and off. The third task again consists of four sequential tasks that turn the LED on and off. The four sequential tasks consist of turning on LED No. 1, waiting for No. 2 for 1 second, turning off No. 3 LED, and waiting for No. 4 for 1 second.

Figure 6 shows the code that executes three independent tasks using the designed framework. SoftwareWerial is used to receive real-time Bluetooth data, and AsyncTaskGroup is used to register three independent tasks as one group. Also, when the user sends the string “Hello” to Bluetooth in real time, the string is output to the serial monitor and the LED on/off is repeated every second. It can also be seen that the time is periodically

printed every 2 seconds on the serial monitor.

taskrameworkarduino-app

```
1  #include <TaskFramework.h>
2  #include <SoftwareSerial.h>
3
4  SoftwareSerial myBlue(10, 11);
5  AsyncTaskGroup atGroup;
6  SyncTaskGroup* ledGroup = new SyncTaskGroup();
7
8  void setup() {
9    Serial.begin(57600);
10   myBlue.begin(57600);
11   pinMode(13, OUTPUT);
12
13   ledGroup->add(new SyncTask([]()->void{
14     Serial.print("led  On:   ");
15     Serial.println( millis() );
16     digitalWrite(13,HIGH);}));
17   ledGroup->add(new DelayTask(1000));
18   ledGroup->add(new SyncTask([]()->void{
19     Serial.print("led Off:   ");
20     Serial.println( millis() );
21     digitalWrite(13,LOW);}));
22   ledGroup->add(new DelayTask(1000));
23
24   atGroup.add(new Task(printSerial,2000));
25   atGroup.add(new Task(recvBluetooth,0));
26   atGroup.add(ledGroup);
27 }
28 void loop() {
29   atGroup.run();
30 }
```

COM4

```
led  On:   82000
led Off:   83000
BLE: hello 83883
Serial:    84000
led  On:   84000
BLE: hello 84569
led Off:   85000
Serial:    86000
led  On:   86000
led Off:   87000
Serial:    88000
led  On:   88000
led Off:   89000
Serial:    90000
led  On:   90000
led Off:   91000
BLE: hello 91022
BLE: hello 91664
Serial:    92000
led  On:   92000
led Off:   93000
BLE: hello 93590
Serial:    94000
led  On:   94000
```

☑ 자동 스크롤 ☐ 타임스탬프

Figure 6. The code that executes three independent tasks using the designed framework

Figure 7 is a pictorial representation of three tasks that are executed asynchronously. These three asynchronous operations theoretically run at the same time. Task 3, which turns the LED on and off, uses a SyncTaskGroup object to group multiple sequential tasks into a single independent task.

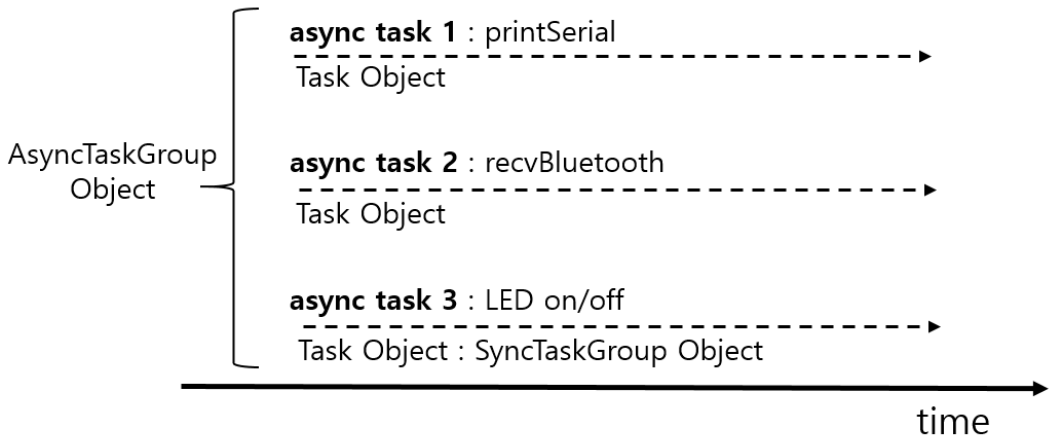

Figure 7. Execution of three asynchronous tasks

5. Conclusion

A good Arduino program should be able to systematically manage the execution order. Arduino was made for easy programming, but as the number of control parts increases, the complexity of the program continues to increase. By systematically managing the execution order, the complexity of the program can be reduced and easy-to-understand code can be written. Various tasks can be executed independently or as dependents. A framework provides a good design to manage these tasks systematically. The Arduino framework proposed in this paper is designed to simplify the program structure and to configure various tasks easily. The design framework of this paper is largely composed of two types: synchronous execution and asynchronous execution. These two execution methods can be combined to create several independent and dependent execution routines. Asynchronous tasks are independently executed tasks and are managed by AsyncTaskGroup, while synchronous tasks are dependently executed tasks and are managed by SyncTaskGroup. AsyncTaskGroup instance and SyncTaskGroup instance are instances of the same Task and can be used in combination with another task. DelayTask, which implements the ISyncTask interface, is designed to allow waiting for a certain period of time without interfering with the execution of other tasks.

References

[1] Hongyong Kim, Donggi Yoon and Seungjung Shin, “Development of Smart Multi-function Ground Resistivity Measuring Device using Arduino in Wind Farm”, The Journal of The Institute of Internet, Broadcasting and Communication(JIIBC), Vol. 19, No. 6, pp.65-71, Dec. 2019.
DOI: https://doi.org/10.7236/JIIBC.2019.19.6.65

[2] Donghwan Gong and Seungjung Shin, “Analysis of Arduino Timer Callback for IoT Devices”, The Journal of The Institute of Internet, Broadcasting and Communication (IIBC), Vol. 18, No. 6, pp.139-143, Dec. 2018.
DOI: https://doi.org/10.7236/JIIBC.2018.18.6.139

[3] Donghwan Gong, “IoT Device Testing for Efficient IoT Device Framework”, The Journal of The Institute of Internet, Broadcasting and Communication(JIIBC), Vol. 12, No. 2, pp. 77-82, Mar. 2020.
DOI: http://dx.doi.org/10.7236/IJIBC.2020.12.2.77

[4] Vera Suryani, Selo Sulistyo and Widyawan Widyawan, “Internet of Things (IoT) Framework for Granting Trust among Objects”, Journal of Information Processing Systems, Vol. 13, No. 6, pp. 1613-1627, Dec. 2017.
DOI: https://doi.org/10.3745/JIPS.03.0088

[5] Hongyong Kim and Seungjung Shin, "A Study on Smart Soil Resistance Measuring Device for Safety Characterized Ground Design in Converged Information Technology", The Journal of The Institute of Internet, Broadcasting and Communication (IIBC), Vol. 19, No. 1, pp.203-209, Feb. 2019.
DOI: https://doi.org/10.7236/JIIBC.2019.19.1.203

[6] Ohseok Kwon and Keehwan Kim, "Implementation of Smart Sensor Network System Based on Open Source Hardware", The Journal of The Institute of Internet, Broadcasting and Communication (IIBC), Vol. 17, No. 1, pp.123-128, Feb. 2017.
DOI: https://doi.org/10.7236/JIIBC.2017.17.1.123

[7] Seokjin Im and Heejoung Hwang, "Design and Development of Framework for Wireless Data Broadcast of XML-based CCR Documents", The Journal of The Institute of Internet, Broadcasting and Communication (IIBC), Vol. 15, No. 5, pp.169-175, Oct. 2015.
DOI: http://dx.doi.org/10.7236/JIIBC.2015.15.5.169

[8] Kyongdeck Jeon and Seungjung Shin, "Proposal for Safety Management of Formwork Construction Using IT Technology", The Journal of The Institute of Internet, Broadcasting and Communication (IIBC), Vol. 20, No. 6, pp.93-99, Dec. 2020.
DOI: https://doi.org/10.7236/JIIBC.2020.20.6.93

[9] Arduino, https://www.arduino.cc/

[10] Architecture framework, https://en.wikipedia.org/wiki/Architecture_framework