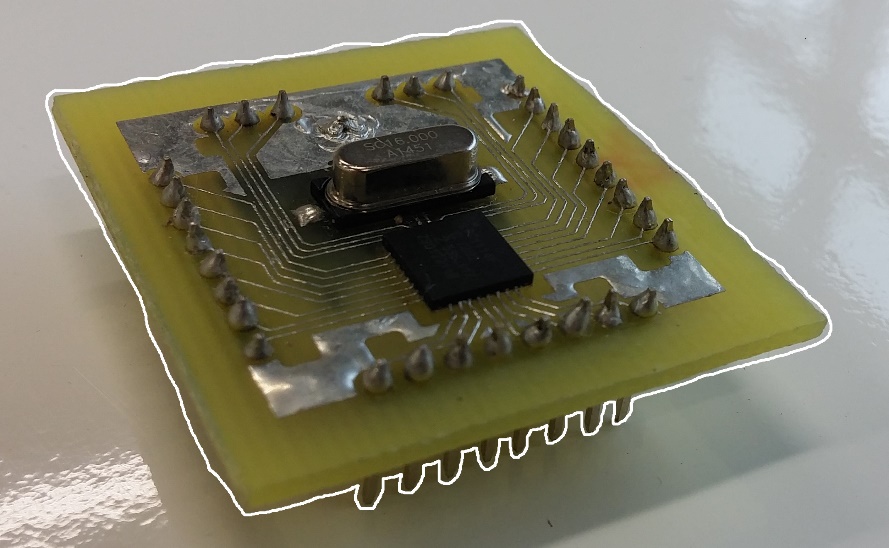
|  |
| --- |
| *Embedded OS with Korean based interpreter language* |

*2015-1st Semester*

*Interdisciplinary Project*



|  |  |
| --- | --- |
| Student ID | 20111235 |
| Name | Seongbeom Park |
| School | ECE |
| 1 track | CSE |
| 2 track | EE |
| Advisor 1 | Woongki Baek |
| Advisor 2 | Jingook Kim |

I. Introduction

1) Topic and Purpose of Research

Nowadays, everybody knows the importance of programming and IoT (Internet of Things) area is on fast growing. The government in Korea, not only the case in Korea but also other many countries, formulated an education policy that is all the students are going to take programming course in youth year at school. To successfully control this phenomenon, a platform which is easy to handle, low power consuming and small sized operating system is really needed.

Though the embedded system is very important part of IoT systems, most programmers doesn’t interested in the embedded programming because there are so many obstacles like too small amount of RAM, low speed, limited libraries, not sufficient debugging environment, etc.

There already exist lots of embedded boards but they have some limitations. In the Arduino case they have sufficient libraries but don’t providing operating system. The programmer should always think about the memory access or device controls so beginner of programming feels really hard on making. The case of NXT, LEGO is familiar platform for anybody but it also uses C language. Galileo made by Intel is able to run a Windows or modified Linux system but it needs too much of power.

In this situation, I think to make a simple operating system providing such as device management system, garbage collection and nice debugging environment can be an effective solution for handling the problems. During the project, I tried to make a design of small but powerful operating system and developing environment which includes Korean based interpreter language on a simple chipset.

II. Main Subject

1) Research Planning

At first I tried to make a hardware with LPC1114FHN33/202 chipsets which architecture is ARM-Cortex M0+. The reason why I choose this chipset is the chipset supporting sufficient functions but it is really cheap to make a board in personally; JTAG debugger, GPIOs, UART communication, SPI communications, I2C communication, etc. After making a decision which chipset to be used, I draw the circuit with Eagle CAD and then printed the circuit to a copper covered PCB. Printed board is etched and soldered.

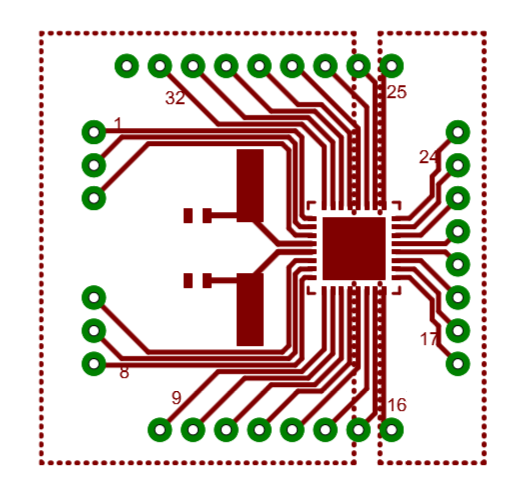
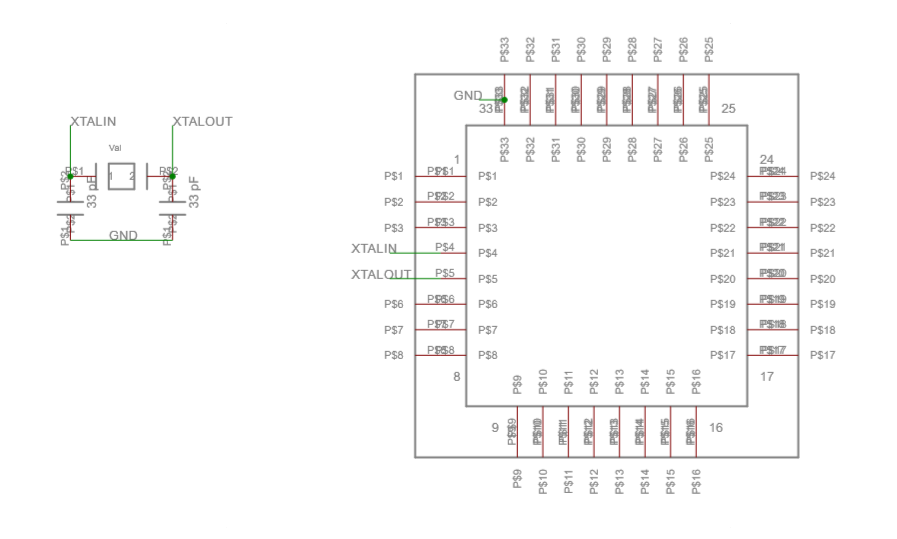


Figure . Schematics Figure . Board design

In the second step I designed operating system. The concept of system is based on the flowchart. The flowchart is a typically used way to describe how a process on going but using flowchart to implement the operating system have not been tried that much.

At the end of project, I tried to run the system design on the board. Due to the board has limited amount of RAM, only 2 KB, system is always considered how to minimize the operating system using memory.

III. Conclusion and Discussion

1) Research Results

The board has really simply design. It contains one micro control unit, one 16 MHz oscillator and two 39 µF capacitors but it providing 20 GPIO pins, 1 UART communication port, 1 SPI communication port which can be used for SD card interface and 1 I2C communication for user devices.

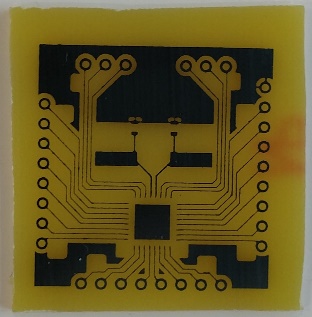
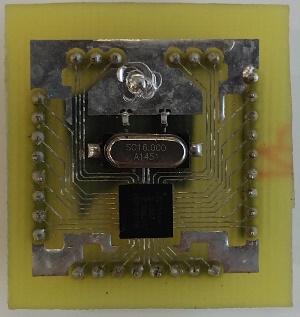
 

Figure . Etched board Figure . Soldered board

The operating system is constructed with 3 main parts; hardware manager, memory manager and thread manager. The hardware manager manages locks of devices which is preventing malfunctioning of a device interfered by more than one thread. Each thread will ask to hardware manager whether the thread can hold the device. While a thread holding a lock of the device, other threads cannot use the device until the thread release lock of the device.

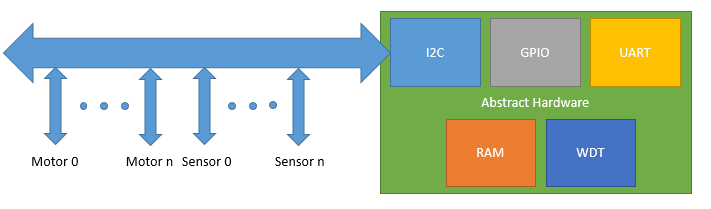


Figure . Devices can be connected with I2C communication

The second part, memory manager manages the heap area which supports dynamic memory allocation. In C++ programming, the programmers who are not familiar with dynamic allocation sometimes makes dangling pointer. The dangling pointer may not be a problem when the system has sufficient memory but the embedded system has limited memory. So the memory leak caused by dangling pointer may cause critical fault of the system even it may cause system down. To prevent this problem, a fast running garbage collection is really needed.

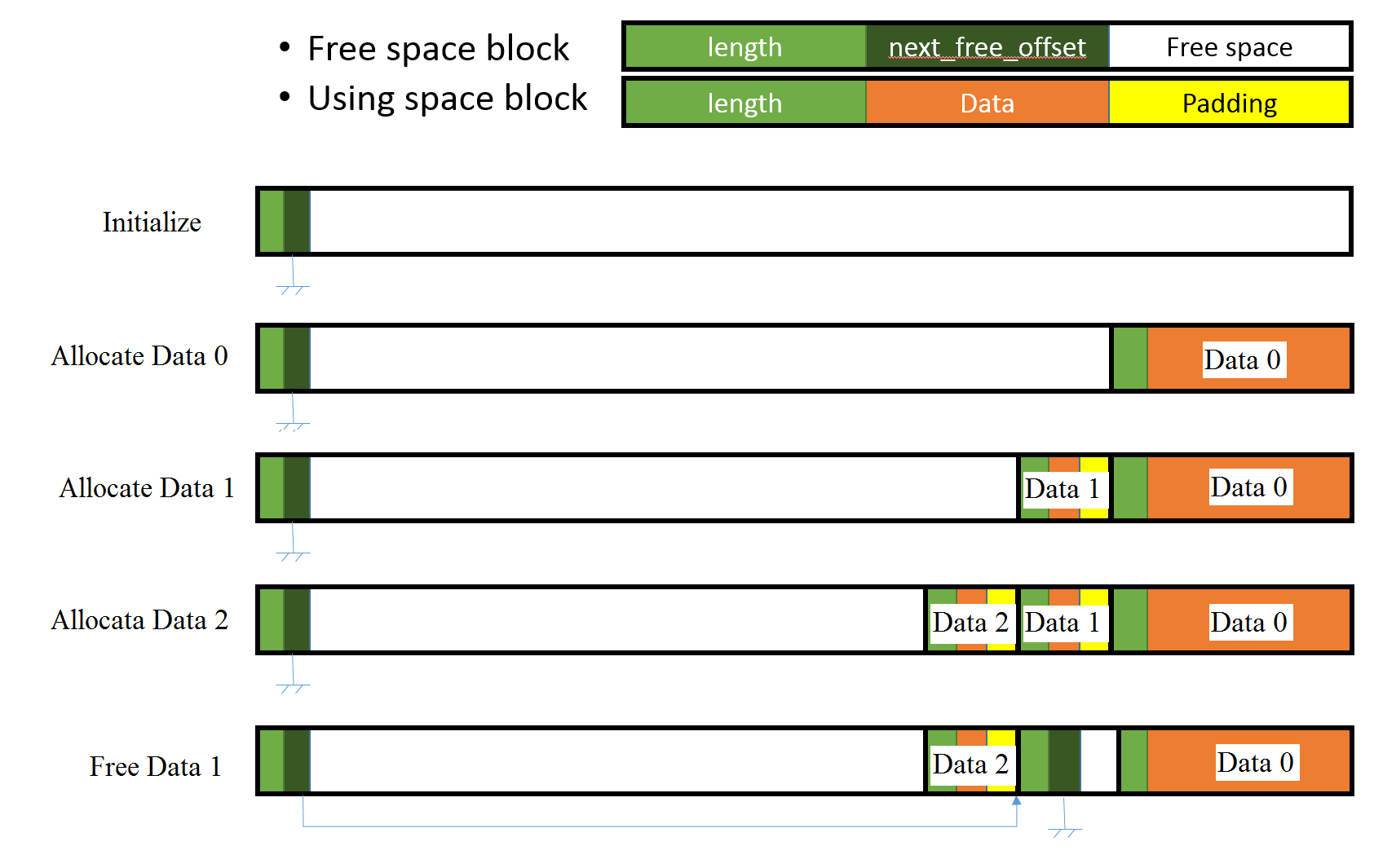


Figure . How the dynamic allocation works

The last main part of operation system is thread manager which has a queue of flowcharts. One block of flowchart has one function pointer or a parameter of the function. By the block flowing the program can be executed. Multi-threading is implemented by dequeue and enqueue of other flowchart’s blocks. To save RAM as much as possible, the blocks will be copied to flash memory which is usually larger than RAM in most embedded chipsets.

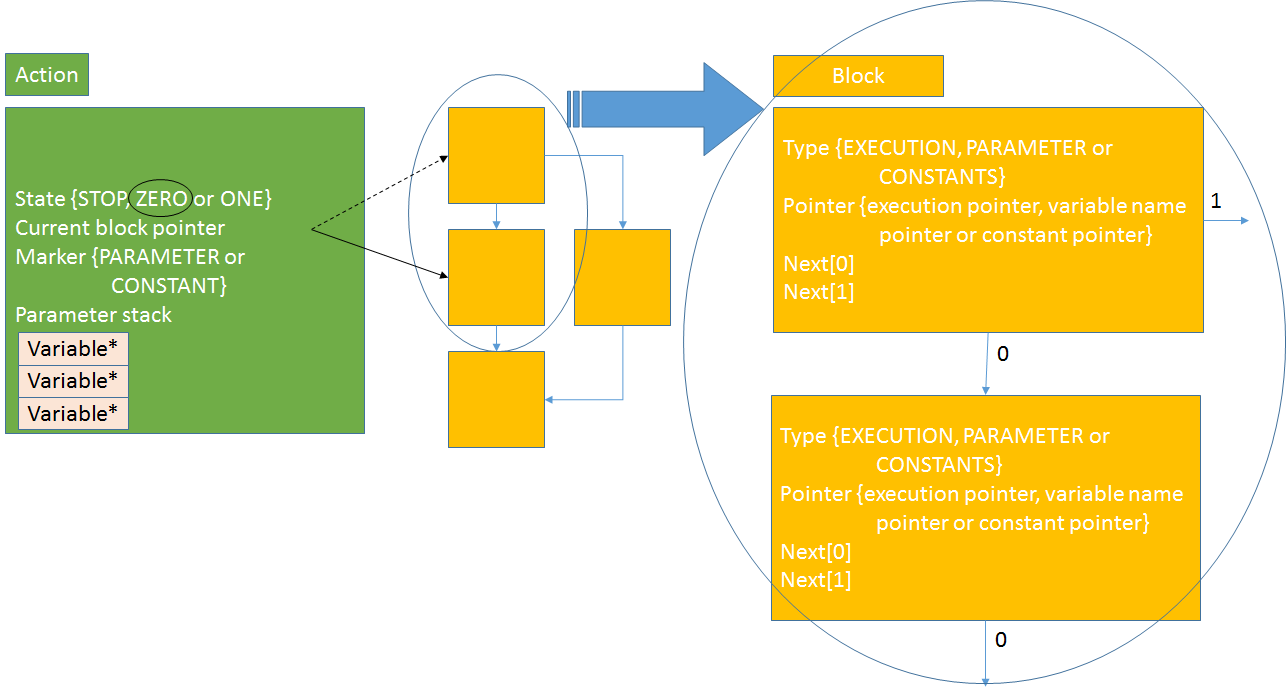


Figure . Structure of a flowchart

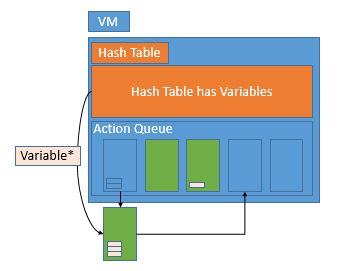


Figure . The case with running three flowcharts (dequeuer and enqueuer a block)

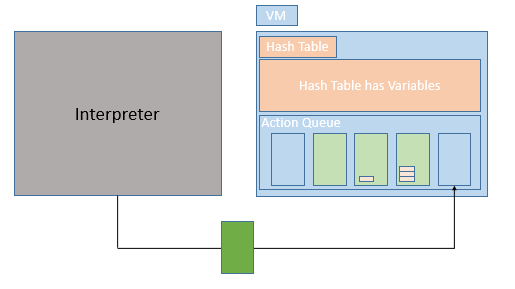


Figure . Interpreter creates a new thread

2) Discussion

Strength

Using the flowchart as a thread helps people easy to understand intuitively how the operating system running.

Using this design of system shows nice performance in memory saving compare to running thread on a RAM in most operating systems do because blocks of flowcharts use the flash memory.

Limitation

If the function is too large or it has infinite loop, the operating system cannot escape from that flowchart. This problem can be solved by using watch dog timer.

※ References

[1] Philip Levis, OSDI’12, Experiences from a Decade of TinyOS Development