Open Questions about Embedded Software

1. **What are the main features of Real-Time Operating Systems and how do they differ from regular OS's?**

   While a product is being designed, it includes different peripherals according to the purpose of use. These peripherals have different priorities according to their role in the product. Some peripherals definitely need to do their job in x time. One of the general examples for this is the airbags of vehicles. In the event of an accident, the airbag should deploy immediately. Another example is missile guidance systems. In this system, even a millisecond delay can cause a deviation in the missile's course. SpaceX even announced that it uses RTOS for Falcon launch vehicles and Dragon capsules. Since my previous field of study was particle physics and radiation, I had read that RTOS was used in the articles I read about CERN. It is used in systems controlling accelerators and data acquisition systems of experiments.

   In real-time operating systems, the important thing is to get the job done on time. In other words, it is important not to do this work in the fastest way, but to do it on time. The time interval when the task starts and ends is important. Because it is the answer to the question of whether the work that needs to be done has been done in the given time. There are two different applications in real time applications. One is hard real time and the other is soft real time. Car airbag app is hard real time. Delay cannot be tolerated. However, the delays that occur during the video call can be tolerated. This is soft real time. Real-time applications require a real-time operating system. The general logic is that there are jobs running in the background. But these things have a priority order. When a high priority event occurs while a low priority event occurs, the other event is interrupted and the high priority thread is executed. It works just like the CANBus protocol. Of course, one is the protocol and the other is the operating system. However, it works with priority logic in CANBus.

   General purpose operating systems also execute threads. However, the purpose of general purpose operating systems is to achieve high efficiency. Can handle high priority and low priority jobs. However, it is important how much work is done per unit time. So instead of spending x time on one high priority job, it spends x time on 3 low priority jobs. This provides high throughput but does not do the high priority work.

2. **When is it preferable to use STM32 over Arduino, ESP32, PIC or other comparable embedded system?**

   In embedded systems, microcontrollers are the brain of the system. It manages when the units do what work. In order to meet the requirements of the project, STM32, PIC microcontrollers, Arduino, ESP32, Raspberry Pi etc. development cards can be used. They are integrated circuit boards that contain different peripherals on the development boards. We can work within the limits of the datasheet set for these development boards. Arduino Uno, Nano, Mega and other models are boards with different GPIO pins and memory features that contain different microprocessors. In electronics hobby projects, we usually start by using the Arduino Uno model. ESP32, ESP8266 modules and development boards containing these modules are used for IOT projects. However, as I said before, we have to work with the features specified in the data sheet. The ESP32 includes Wifi and Bluetooth, but using both at the same time is not recommended. The convenience of programming environments was an advantage when I was

working with these development boards. But I had many problems because of the constraints. When I designed a radiation counter with ESP32, when I tried to read a 2.45 uS signal from the detector with an interrupt, I saw that there was saturation on the digital side at high radiation doses and I realized that the detector was working at a lower value than I expected. When I examined my analog circuit, it was still reading. As a result of my research, I found that gpio lags and wifi bluetooth usage are effective and other tasks are causing this issue. To solve this problem, I learned that just using a microcontroller to do the counting and transmit the data would give healthier results. This simple example was my own experience on why the STM32 should be used over others. In addition, it may be in peripheral units that we do not use on development boards. When making a product, the size of that product is important according to its intended use. The only component that makes up the dimension is the electronic circuit board inside.

With STM32, we can design products that meet the criteria we want. Most electronics we use today contain microcontroller-based cards. In short, if using a development card is sufficient in terms of the number of gpio, memory requirements and other features according to the requirements of the project, using the appropriate development card can be selected to use the time effectively. But it is always better to produce your own card.

3. **Let's assume that a few of the peripherals you use have the same hardcoded I2C address. What solution would you use to solve this problem?**

I had this problem. The MAX30101 sensor is a photodetector sensor. I needed to log the data obtained from this detector with time and date information. I used Arduino Uno as development board. I preferred the DS3231 module to get time information. Getting the time right was precise and more convenient because it contained the module crystal. By default, the DS3231 module has an I2C address of 0x68. The eeprom on this module has an I2C address of 0x57. The default I2C address of the photodetector sensor was 0x57. I have thought and researched many methods to solve this problem. At first, I thought of using ESP32 instead of Arduino and pulling the time and date information with NTP, but when I examined the DS3231 module, I saw that the address of the eeprom could be changed hardware-wise for this situation to occur. This is how I solved my own problem.
However, there are I2C address multiplexer modules as other solution suggestions. They can be used. The TCA9548A module supports up to 8 devices with the same address. After the hardware is connected with this multiplexer, software settings are also made..

4. **Let's say you want about 30 hardware modules to communicate with each other. There is one STM32 microcontroller on each module, and the modules can be removed and installed instantly. Which communication standard would you use for these modules to communicate effectively with each other? Why?**

Actually, data size and distance are important for us here. We can consider two different methods as wired or wireless solutions. For wireless solutions, we can communicate using zigbee or ble-mesh. Of these, I would prefer to use zigbee. Because it can work without mesh network and center device. But if we think for wired interface, we can use I2C (TWI) or SPI as distance dependent. However, for SPI, there will be a chip optional cable requirement for each device and main controller. I2C remains a sensible option in this case. Since we can use it with two wires, we will also save pins on the MCU side.

5. **3 devices with identical embedded software and hardware are required to communicate over I2C. What kind of solution would you develop so that devices can be dynamically addressed and recognize each other?**

   We have 3 identical hardware. These hardware also have the same software. But here the unique IDs of the microcontroller are different. I make a pre-identifier using these unique IDs for communication with I2C. At first, this ID data is transmitted. The I2C address is 7 or 10 bits long. STM32 addresses are 96 bits long. We can create a hash function that will return a result of 7 or 10 bits maybe less. We do this from the STM32's ID. Thus, it creates a hash of unique IDs for each device. If we hash with a lower bit value, we can fill in the blanks with 0. Thus, each fixed and unique address can be formed. We can use **“pearson hashing”** for this. With this method, we can use it for devices to recognize each other.