<https://www.tutorialspoint.com/embedded_systems/index.htm>

<https://career.guru99.com/top-18-embedded-systems-interview-questions/>

<https://www.wisdomjobs.com/e-university/embedded-systems-interview-questions.html>

<https://aticleworld.com/embedded-c-interview-questions-2/>

<https://www.toptal.com/embedded/interview-questions>

<https://www.monsterindia.com/career-advice/10-embedded-system-questions-and-answers-to-help-you-succeed-in-your-next-interview-7954.html>

<https://www.quora.com/What-are-some-good-embedded-interview-questions>

# Embedded Systems

## ISR

https://www.tutorialspoint.com/embedded\_systems/es\_interrupts.htm

http://thevolatilepointer.blogspot.com/2012/04/interrupt-service-routine-isr.html

http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C12\_Interrupts.htm

### What is ISR?

An ISR (Interrupt Service Routine) is an interrupt handler, a callback subroutine which is called when an interrupt is encountered.

### What is return type of ISR?

ISR does not return anything. An ISR returns nothing because there is no caller in the code to read the returned values.

### What is interrupt latency?

Interrupt latency is the time required for an ISR responds to an interrupt.

### How do you measure interrupt latency?

With the help of the oscilloscope, we can measure the interrupt latency. You need to take following steps.

* First takes two GPIOs.
* Configure one GPIO to generate the interrupt and second for the toggling (if you want you can attach an LED).
* Monitor the PIN (using the oscilloscope or analyzer) which you have configured to generate the interrupt.
* Also, monitor (using the oscilloscope or analyzer) the second pin which is toggled at the beginning of the interrupt service routine.
* When the interrupt is triggered, the signal of the both GPIOs will change.
* The interval between the two signals (interrupt latency) may be easily read from the instrument.

### How to reduce interrupt latency?

The interrupt latency depends on many factors. Some of them are:

* Platform and interrupt controller
* CPU clock speed
* Timer frequency
* Cache configuration
* Application program

So using the proper selection of platform and processor, we can easily reduce the interrupt latency.

We can also reduce the interrupt latency by making the ISR shorter and avoid calling a function within the ISR.

### Can we use any function inside ISR?

We can use function inside ISR as long as that function is not invoked from other portion of the code.

### Can we use printf inside ISR?

The printf function in ISR is not supported because printf function is not reentrant, thread safe and uses dynamic memory allocation which takes a lot of time and can affect the speed of an ISR up to a great extent.

### Can we put breakpoint inside ISR?

Putting a breakpoint inside ISR is not a good idea because debugging will take some time and a difference of half or more second will lead to different behavior of hardware. To debug ISR, logs are better.

### What is Top Half & Bottom Half of a kernel?

Sometimes to handle an interrupt, a substantial amount of work has to be done. But it conflicts with the speed need for an interrupt handler. To handle this situation, Linux splits the handler into two parts – Top half and Bottom half. The top half is the routine that actually responds to the interrupt. The bottom half, on the other hand, is a routine that is scheduled by the upper half to be executed later at a safer time.

All interrupts are enabled during execution of the bottom half. The top half saves the device data into the specific buffer, schedules bottom half and exits. The bottom half does the rest. This way the top half can service a new interrupt while the bottom half is working on the previous.

## Timer

### Significance of watchdog timer in embedded systems

The watchdog timer is a timing device with a predefined time interval. During that interval, some events may occur or else the device generates a time-out signal. It is used to reset to the original state whenever some inappropriate events take place which can result in system malfunction. It is usually operated by counter devices.

## RTOS

### What is RTOS?

In an operating system, there is a module called the scheduler, which schedules different tasks and determines when a process will execute on the processor. This way, the multi-tasking is achieved. The scheduler in a Real Time Operating System (RTOS) which is designed to provide a predictable execution pattern. In an embedded system, a certain event must be entertained in strictly defined time.

Main features of an RTOS are:

* Context switching latency should be short.
* Interrupt latency should be short.
* Interrupt dispatch latency should be short.
* Reliable and time bound inter process mechanisms
* Should support kernel preemption.

### What is the difference between hard real-time and soft real-time OS?

A Hard real-time system strictly adheres to the deadline associated with the task. If the system fails to meet the deadline, even once, the system is considered to have failed. In case of a soft real-time system, missing a deadline is acceptable. In this type of system, a critical real-time task gets priority over other tasks and retains that priority until it completes.

### What type of scheduling is there in RTOS?

RTOS uses pre-emptive scheduling. In pre-emptive scheduling, the higher priority task can interrupt a running process and the interrupted process will be resumed later.

### What is priority inversion?

If two tasks share a resource, the one with higher priority will run first. However, if the lower-priority task is using the shared resource when the higher-priority task becomes ready, then the higher-priority task must wait for the lower-priority task to finish. In this scenario, even though the task has higher priority it needs to wait for the completion of the lower-priority task with the shared resource. This is called priority inversion.

### What is priority inheritance?

Priority inheritance is a solution to the priority inversion problem. The process waiting for any resource, which has a resource lock, will have the maximum priority. This is priority inheritance. When one or more high priority jobs are blocked by a job, the original priority assignment is ignored and execution of critical section will be assigned to the job with the highest priority in this elevated scenario. The job returns to the original priority level soon after executing the critical section.

## IPC

Different types of IPC mechanism are:

* Pipes
* Named pipes or FIFO
* Semaphores
* Shared memory
* Message queue
* Socket

## Multithread

### What is spin lock?

If a resource is locked, a thread which wants to access that resource may repetitively check whether the resource is available. During that time, the thread may loop and check the resource without doing any useful work. Suck a lock is termed as spin lock.

### What is semaphore?

Semaphore is actually a variable or abstract data type which controls access to a common resource by multiple processes.

It is of two types:

* Binary semaphore: It can have only two values (0 and 1). The semaphore value is set to 1 by the process in charge, when the resource is available.
* Counting semaphore: It can have value greater than one. It is used to control access to a pool of resources.

### What is difference between binary semaphore and mutex?

The differences between binary semaphore and mutex are:

* Mutual exclusion and synchronization can be used by binary semaphore, while mutex is used only for mutual exclusion.
* A mutex can be released by the same thread which acquired it. Semaphore values can be changed by other thread.
* From an ISR, a mutex cannot be used.
* The advantage of semaphores is that, they can be used to synchronize two unrelated processes trying to access the same resource.
* Semaphores can act as mutex, but the opposite is not possible.

# C Embedded Programming

## Volatile

<https://www.geeksforgeeks.org/understanding-volatile-qualifier-in-c/>

<https://barrgroup.com/Embedded-Systems/How-To/C-Volatile-Keyword>

<https://www.embeddedrelated.com/thread/4749/when-and-how-to-use-the-volatile-keyword-embedded-c-programming>

### What is the use of volatile keyword?

The C's volatile keyword is a qualifier that tells the compiler not to optimize when applied to a variable. By declaring a variable volatile, we can tell the compiler that the **value of the variable may change any moment from outside of the scope of the program**. A variable should be declared volatile whenever its value could change unexpectedly and beyond the comprehension of the compiler.

In those cases it is required not to optimize the code, doing so may lead to incorrect result and load the variable every time it is used in the program. Volatile keyword is useful for memory-mapped peripheral registers, global variables modified by an **ISR** (Interrupt Service Routine), **global variables accessed by multiple tasks within a multi-threaded application**.

### Can a variable be both const and volatile?

The const keyword makes sure that the value of the variable can't be changed. This statement holds true in the scope of the program. The value can still be changed by outside intervention. So, the use of const volatile makes perfect sense.

### Can a variable be both static and volatile?

Yes, the use of static volatile is allowed.

### Can a volatile variable be passed to a function?

Yes.

### Can a pointer be volatile?

If we see the declaration volatile int \*p, it means that the pointer itself is not volatile and points to an integer that is volatile. This is to inform the compiler that pointer p is pointing to an integer and the value of that integer may change unexpectedly even if there is no code indicating so in the program.

### What is wrong with this code?

int square(volatile int \*p) {

return (\*p) \* (\*p);

}

The intention of the above code is to return the square of the integer pointed by the pointer p. Since it is volatile, the value of the integer may have changed suddenly and will result in something else which will looks like the result of the multiplication of two different integers. To work as expected, the code needs to be modified like this.

int square(volatile int \*p) {

int a = \*p;

return a\*a;

}

## Inline

### What are inline functions?

The ARM compilers support inline functions with the keyword \_\_inline. These functions have a small definition and the function body is substituted in each call to the inline function. The argument passing and stack maintenance is skipped and it results in faster code execution, but it increases code size, particularly if the inline function is large or one inline function is used often.

### Advantages and disadvantages of using macro and inline functions?

The advantage of the macro and inline function is that the overhead for argument passing and stuff is reduced as the function are inline. The advantage of macro function is that we can write type insensitive functions. It is also the disadvantage of macro function as macro functions can't do validation check. The macro and inline functions also increase the size of the executable.

### What happens when recursive functions are declared inline?

Inlining a recursive function reduces the overhead of saving context on stack. But, inline is merely a suggestion to the compiler and it does not guarantee that a function will be inline. Obviously, the compiler won't be able to inline a recursive function infinitely. It may not inline it at all or it may inline it, just a few levels deep.

## Struct

### Can structures be passed to the functions by value?

Passing structure by its value to a function is possible, but not a good practice. First of all, if we pass the structure by value and the function changes some of those values, then the value change is not reflected in caller function. We want that value changed because structure is used in global scope.

Also, if the structure is big, then passing the structure by value means copying the whole structure to the function argument stack. This slows down the program and consumes more memory.

### How can you define a structure with bit field members?

Bit field members can be declared as shown below

struct A

{

char c1 : 3;

char c2 : 4;

char c3 : 1;

};

Here c1, c2 and c3 are members of a structure with width 3, 4, and 1 bit respectively. The ':' indicates that they are bit fields and the following numbers indicates the width in bits.

## Data Types

### How is generic list manipulation function written which accepts elements of any kind?

It can be achieved using void pointer. A list may be expressed by a structure as shown below:

typedef struct

{

node \*next;

void \*data;

} node;

### Guess the output

#include<stdio.h>

main()

{

unsigned int a = 2;

int b = -10;

(a + b > 0) ? puts("greater than 0") : puts("less than 1");

}

Output - greater than 0

The a + b is -8, if we do the math. But here the addition is between different integral types - one is unsigned int and another is int. So, all the operands in this addition are promoted to unsigned integer type and b turns to a positive number and eventually a big one. The outcome of the result is obviously greater than 0.

## Static

### Can static variables be declared in a header file?

A static variable can be defined in the header file. But doing so, the result will be having a private copy of that variable in each source file which includes the header file. So it will be wise not to declare a static variable in header file, unless you are dealing with a different scenario.

## Preprocessor, Compiler, Linker

### Consider the two statements below and point out which one is preferred and why?

#define B struct A\*

typedef struct A\* C;

The typedef is preferred. Both statements declare pointer to struct A to something else and in one glance both looks fine. But there is one issue with the define statement. Consider a situation where we want to declare p1 and p2 as pointer to struct A. We can do this by: C p1, p2;

But doing B p1, p2 will expand to struct A \* p1, p2. It means that p1 is a pointer to struct A but p2 is a variable of struct A and not a pointer.

### What is the job of preprocessor, compiler, assembler and linker?

* The preprocessor commands are processed and expanded by the preprocessor before actual compilation.
* After preprocessing, the compiler takes the output of the preprocessor and the source code, and generates assembly code.
* Once compiler completes its work, the assembler takes the assembly code and produces an assembly listing with offsets and generate object files.
* The linker combines object files or libraries and produces a single executable file. It also resolves references to external symbols, assigns final addresses to functions and variables, and revises code and data to reflect new addresses.

### What is the difference between static linking and dynamic linking?

In static linking, all the library modules used in the program are placed in the final executable file making it larger in size. This is done by the linker. If the modules used in the program are modified after linking, then re-compilation is needed. The advantage of static linking is that the modules are present in an executable file. We don't want to worry about compatibility issues.

In case of dynamic linking, only the names of the module used are present in the executable file and the actual linking is done at run time when the program and the library modules both are present in the memory. That is why, the executables are smaller in size. Modification of the library modules used does not force re-compilation. But dynamic linking may face compatibility issues with the library modules used.

## Pointer

### Why cannot arrays be passed by value to functions?

In C, the array name itself represents the address of the first element. So, even if we pass the array name as argument, it will be passed as reference and not its address.

### What is wild pointer?

A pointer that is not initialized to any valid address or NULL is considered as wild pointer. Consider the following code fragment:

int \*p;

\*p = 20;

Here p is not initialized to any valid address and still we are trying to access the address. The p will get any garbage location and the next statement will corrupt that memory location.

### What is dangling pointer?

If a pointer is de-allocated (freed) and it’s not assigned to NULL, then it may still contain that address and accessing the pointer means that we are trying to access that location and it will give an error. This type of pointer is called dangling pointer.

### Write down the equivalent pointer expression for referring the same element a[i][j][k]?

We know that a[i] can be written as \*(a+i). Same way, the array elements can be written like pointer expression as follows:

a[i][j] == \*(\*(a+i)+j)

a[i][j][k] == \*(\*(\*(a+i)+j)+k)

### On a certain project it is required to set an integer variable at the absolute address 0x67a9 to the value 5. The compiler is a pure ANSI compiler. Write code to accomplish this task.

This can be achieved by the following code fragment:

int \*ptr;

ptr = (int\*)0x67a9;

\*ptr = 5;

## Bit Manipulation

### How do you write a function which takes 2 arguments - a byte and a field in the byte and returns the value of the field in that byte?

The function will look like this:

int GetFieldValue(int byte, int field )

{

byte = byte >> field;

byte = byte & 0x01;

return byte;

}

The byte is right shifted exactly n times where n is same as the field value. That way, our intended value ends up in the 0th bit position. "Bitwise And" with 1 can get the intended value. The function then returns the intended value.

### Which bit wise operator is suitable for checking whether a particular bit is on or off?

"Bitwise And" (&) is used to check if any particular bit is set or not. To check whether 5'th bit is set we can write like this:

bit = (byte >> 4) & 0x01;

Here, shifting byte by 4 position means taking 5'th bit to first position and "Bitwise And" will get the value in 0 or 1.

65) Why doesn't the following statement work?

char str[] = "Hello" ;

strcat (str, '!') ;

The string function strcat() concatenates two strings. But here the second argument is '!', a character and that is the reason why the code doesn't work. To make it work, the code should be changed like this:

strcat (str, "!") ;

### Write a code fragment to set and clear only bit 3 of an integer

#define BIT(n) (0x1 << n)

int a;

void ClearBit3() {

a &= ~BIT(3);

}

Explanation:

If 0x01 = 0000 0001, then 0x01 << 3 = 0000 1000

If a = 0010 **1**001, then a = a & 1111 0111 = 0010 **0**001

## Others

### Are count-down loops better than count-up-loops?

Count-down loops are better. Reason behind this is that at loop termination, comparison to zero can be optimized by the compiler. Most processors have instruction for comparing to zero. So they don't need to load the loop variable and the maximum value, subtract them and then compare to zero.

### Guess the output

main()

{

fork();

fork();

fork();

printf("hello world\n");

}

It will print 'hello world' 8 times. The main() will print one time and creates 3 children, let us say Child\_1, Child\_2, Child\_3. All of them printed once. The Child\_3 will not create any child. Child2 will create one child and that child will print once. Child\_1 will create two children, say Child\_4 and Child\_5 and each of them will print once. Child\_4 will again create another child and that child will print one time. A total of eight times the printf statement will be executed.

### Why ++n executes faster than n+1?

The expression ++n requires a single machine instruction such as INR to carry out the increment operation. In case of n+1, apart from INR, other instructions are required to load the value of n. That is why ++n is faster.