Regulated DC Power Supply to run an LED

Contents

[Problem Statement: 2](#_Toc515359143)

[Solution: 2](#_Toc515359144)

[Theory behind the circuit 3](#_Toc515359145)

[Steps in building the circuit 3](#_Toc515359146)

[Step 1: Circuit Designing 3](#_Toc515359147)

[Step2. Circuit Drawing and Simulation 5](#_Toc515359148)

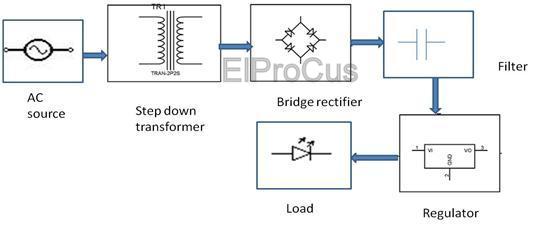
# **Problem Statement**:

Design a regulated DC power supply of 5V which can be used to run a LED, using AC voltage as the input.

# **Solution**:

You all must be aware of regulated DC power supply. If not, let me give a brief idea. Actually, most of the circuits or [electronic devices](http://www.edgefxkits.com/tv-remote-operated-domestic-appliances-control) require a DC voltage for their operation. We can use simple batteries to provide the voltage, but the major problem with batteries is their limited life time. For this reason, the only way we have, is to convert the AC voltage supply at our homes to the required DC voltage.

All we need is to convert this AC voltage into DC voltage. But it is not as simple as it seems. So, let us have a brief theoretical idea about how AC voltage is converted into regulated DC voltage.

[](https://www.elprocus.com/wp-content/uploads/2013/11/Bridge-Rectifier.jpg)Block Diagram by [ElProCus](https://www.elprocus.com/" \t "_blank)

# Theory behind the circuit

1. AC voltage from the supply at 230V is first stepped down to low voltage AC using a step down transformer. A transformer is a device with two windings –primary and secondary, wherein voltage applied across the primary winding , appears across the secondary winding by the virtue of inductive coupling. Since secondary coil has lesser number of turns, the voltage across the secondary is less than the voltage across the primary for a step down transformer.
2. This low AC voltage is converted to pulsating DC voltage using bridge rectifier. A bridge rectifier is an arrangement of 4 diodes placed in bridged form, such that anode of one diode and cathode of another diode are connected together to positive terminal of the voltage source and in the same way the anode and cathode of another two diodes are connected to the negative terminal of the voltage source. Also the cathodes of two diodes are connected together to the positive polarity of the voltage and the anode of two diodes is connected together to the negative polarity of the output voltage. For each half cycle, the opposite pair of diodes conducts and pulsating DC voltage is obtained across the bridge rectifiers.
3. The pulsating DC voltage thus obtained contains ripples in form of AC voltage. To remove these ripples a filter is needed which filters out the ripples from the DC voltage. A capacitor is placed in parallel to the output such that the capacitor (because of its impedance) allows high frequency AC signals to pass through get bypassed to ground and low frequency or DC signal is blocked. Thus the capacitor acts as a low pass filter.
4. The output produced from a capacitor filter is the unregulated DC voltage. To produce a regulated DC voltage a regulator is used which develops a constant DC voltage.

So let us now get into designing a simple AC-DC regulated power supply circuit to drive a LED.

# Steps in building the circuit

## Step 1: Circuit Designing

To design a circuit, we need to have idea about the values of each component required in the circuit. Let us now see how we are actually designing a regulated DC power supply circuit.

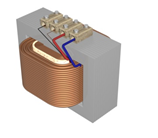
**1. Decide the regulator to be used and its input voltage.**

Here our requirement is to have a constant voltage of 5V at 20mA with positive polarity of the output voltage. For this reason we need a regulator which would provide a 5V output. An ideal and efficient choice would be the regulator IC LM7805. Our next requirement is to calculate the input voltage requirement for the regulator. For a regulator, the minimum input voltage should be the output voltage added by a value of three. In that case, here to have a voltage of 5V, we need a minimum input voltage of 8V. Let us settle down for an input of 12V.

[](https://www.elprocus.com/wp-content/uploads/2013/11/7805-regulator.png)7805 regulator by [Flickr](http://farm3.staticflickr.com/2680/4149945173_9ca4c43c13_z.jpg?zz=1)

**2. Decide the transformer to be used**

Now the unregulated voltage produced is a voltage of 12V. This is the RMS value of the secondary voltage required for a transformer. Since the primary voltage is 230V RMS, on calculating the turns ratio, we get a value of 19. Hence we have to get a transformer with 230V/12V , i.e. a 12V, 20mA transformer.

[](https://www.elprocus.com/wp-content/uploads/2013/11/Step-down-transformer.png)Step down transformer by [Wiki](http://upload.wikimedia.org/wikipedia/commons/7/7a/Transformer-hightolow_smaller.jpg)

**3. Decide the value of the filter capacitor**

The value of the filter capacitor depends on the amount of current drawn by the load, the quiescent current (ideal current) of the regulator, the amount of allowable ripple in the DC output and the time period.

For the peak voltage across the transformer primary to be 17V(12\*sqrt2) and the total drop across the diodes to be (2\*0.7V) 1.4V, the peak voltage across the capacitor is about 15V approx. We can calculate the amount of allowable ripple by the formula below:

∆V = VpeakCap- Vmin

As calculated, Vpeakcap = 15V and Vmin is the minimum voltage input for the regulator. Thus ∆V is (15-7)= 8V.

Now, Capacitance, C =( I\*∆t)/ ∆V,

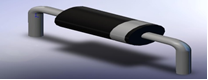
Now, I is the sum of the load current plus the quiescent current of the regulator and I = 24mA (Quiescent current is about 4mA and load current is 20mA). Also ∆t = 1/100Hz = 10ms. The value of ∆t depends upon the frequency of the input signal and here the input frequency is 50Hz.

Thus substituting all the values, the value of C comes to be around 30microFarad. So, let us select a value of 20microFarad.

[](https://www.elprocus.com/wp-content/uploads/2013/11/An-electrolyte-capacitor.png)An electrolyte capacitor by [Wiki](http://upload.wikimedia.org/wikipedia/commons/d/db/47uf_Electrolytic_Capacitor.jpg)

**4. Decide the PIV (peak inverse voltage) of the diodes to be used.**

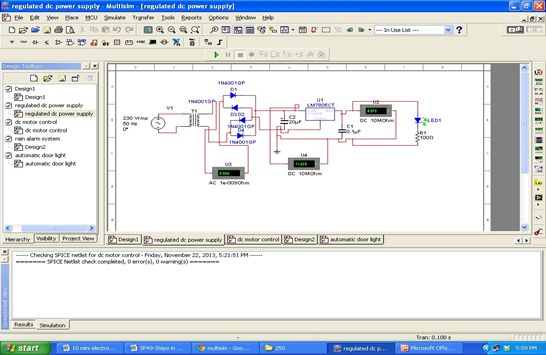
Since the peak voltage across the transformer secondary is 17V, the total PIV of the diode bridge is about (4\*17) i.e. 68V. So we have to settle down for diodes with PIV rating of 100V each. Remember PIV is the maximum voltage which can be applied to the diode in its reverse biased condition, without causing breakdown.

[](https://www.elprocus.com/wp-content/uploads/2013/11/PN-Junction-diode.png)PN Junction diode by [Nojavanha](http://www.nojavanha.com/media/2013/01/%D8%B1%D9%88%D8%A8%D8%A7%D8%AA%DB%8C%DA%A91.jpg" \t "_blank)

## Step2. Circuit Drawing and Simulation

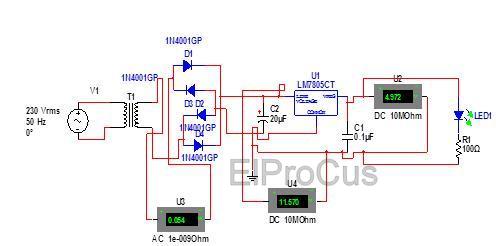
Now that you have the idea of the values for each component and the whole circuit diagram, let us get into drawing the circuit using circuit building software and simulate it.

Here our choice of the software is Multisim.

[](https://www.elprocus.com/wp-content/uploads/2013/11/Multisim-window.png)Multisim window

Below are the given steps to draw a circuit using Multisim and simulate it.

1. On your windows panel, click on the following link: Start >>> Programs –> National –> Instruments –> Circuit design suite 11.0 –> multisim 11.0.
2. A multisim software window appears with a menubar and blank space resembling a breadboard, to draw the circuit.
3. On the menu bar, select place –> components
4. A window appears with the title-‘select the components’
5. Under the heading ‘Database’ – select ‘Master Database’ from the drop down menu.
6. Under the heading ‘group’- select the required group. If you want to go for voltage or current source or ground. If you want to go for any basic component like a resistor, a capacitor etc. Here first we have to place the input AC supply source, hence select Source –>Power Sources –> AC\_power. After the component is placed (by clicking the ‘ok’ button), set the value of RMS voltage to 230 V and frequency to 50Hz.
7. Now again under the components window, select basic, then transformer, then select TS\_ideal. Since for an ideal transformer, the inductance of both coils is same, to achieve our output we have the change the secondary coil inductance. Now we know ratio of inductance of the transformer coils is equal to square of the ratio of turns. Since turns ratio required in this case is 19, therefore we have to set the secondary coil inductance to 0.27mH. (Primary coil inductance is at 100mH).
8. Under the components window, select basic, then diodes, and then select the diode IN4003. Select 4 such diodes and place them in a bridge rectifier arrangement.
9. Under the components windows, select basic, then Cap \_Electrolytic and select the value of capacitor to be 20microFarad.
10. Under the components window, select power, then Voltage\_ Regulator and then select ‘LM7805’ from the drop down menu.
11. Under the components window, select diodes, then select LED and from the drop down menu, select LED\_green.
12. Using the same procedure, select a resistor with the value of 100 Ohms.
13. Now that we have all the components and have an idea about the circuit diagram, let us get into drawing the circuit diagram on the multisim platform.
14. To draw the circuit, we have to make proper connections between the components using wires. To select wires, go to Place, then wire. Remember to connect the components only when a junction point appears. In multisim, the connecting wires are indicated by red color.
15. To get an indication of the voltage across the output, follow the given steps. Go to Place, then ‘Components’, then ‘indicator’, then ‘Voltmeter’, then select the first component.
16. Now your circuit is ready to be simulated.
17. Now click on ‘Simulate’ then select ‘Run’.
18. Now you can see the LED at the output blinks, which is indicated by the arrows going green in color.
19. You can verify whether you are getting correct value of voltage across each component by placing a Voltmeter in parallel.

[](https://www.elprocus.com/wp-content/uploads/2013/11/A-complete-Simulated-Circuit-Diagram.jpg)A complete Simulated Circuit Diagram by [ElProCus](https://www.elprocus.com/" \t "_blank)

Now you have an idea about designing a regulated power supply for loads which require a constant DC voltage, but what about loads which require variable DC voltage. I leave you with this task.Furthermore, any queries regarding this concept or electrical and [electronics projects](http://www.efxkits.com/)

[Link1](https://www.elprocus.com/steps-to-building-a-project-on-breadboard-circuit/)

<http://www.instructables.com/id/How-to-make-a-Clap-Clap-on-Clap-Clap-Off-switch-/>

<http://www.instructables.com/id/Color-Changing-Night-Light/>

<https://www.electronicshub.org/water-level-controller-using-8051-microcontroller/>

<http://www.instructables.com/id/Ten-Breadboard-Projects-For-Beginners/>

<https://www.electronicshub.org/adjustable-timer/>

<https://www.electronicshub.org/electronics-projects-ideas/>

<https://www.electronicshub.org/electronics-mini-project-circuits/>

<http://www.circuitstoday.com/simple-electronics-projects-and-circuits>

<https://circuitdigest.com/electronic-circuits>

<https://electronicsforu.com/category/electronics-projects/hardware-diy>

<http://www.instructables.com/id/Lets-Make-5-More-BreadBoard-Projects-for-Begginers/>

<https://www.treehugger.com/gadgets/17-best-diy-projects-2013.html>

<https://www.electronicshub.org/electronics-projects-kids/>

<https://www.elprocus.com/simple-electronic-projects-for-beginners-in-electronics-engineering/>

<http://www.ni.com/gate/gb/GB_EVALMULTISIM/US>

( [Khairul.Basar@LntTechservices.com](mailto:Khairul.Basar@LntTechservices.com), Ps120182)

# Book & Reference:

<http://rads.stackoverflow.com/amzn/click/0521370957>

<https://www.ibiblio.org/kuphaldt/electricCircuits/>

<https://www.allaboutcircuits.com/>

Books such as Electronics for Dummies

<https://www.electronicshub.org/electronics-books-beginners/>

<https://www.wiley.com/en-us/Electronics+All+in+One+For+Dummies-p-9780470147047>

<https://news.ycombinator.com/item?id=16775744>

<http://nptel.ac.in/courses/IIT-MADRAS/Basic_Electronics_Lab/LECTURE1.pdf>

<http://nptel.ac.in/courses/IIT-MADRAS/Basic_Electronics_Lab/LECTURE1.pdf>

<http://engineering.nyu.edu/gk12/amps-cbri/pdf/Basic%20Electronics.pdf>

<https://www.electronics-tutorials.ws/pdf/basic-electronics-tutorials.pdf>

<http://webwork.utleon.edu.mx/Paginas/Documentos/Robotica/electronica/(ebook)%20Gibilisco,%20Stan%20-%20Teach%20Yourself%20Electricity%20and%20Electronics.pdf>

<http://phy415.ahepl.org/Elecronics-Kybett.pdf>

<https://www.tutorialspoint.com/basic_electronics/basic_electronics_tutorial.pdf>

<http://www-mdp.eng.cam.ac.uk/web/library/enginfo/electrical/hong1.pdf>

<http://fmipa.umri.ac.id/wp-content/uploads/2016/03/Tooley_Electronic_Circuits_-_Fundamentals_and_ApBookZZ.org_.pdf>

<https://www.circuitlab.com/textbook/?from=editor_onload_promo>

# Design Software

<https://easyeda.com/>

<http://everycircuit.com/app/>

<http://www.ni.com/multisim/>

<https://www.tinkercad.com/>

<https://www.labcenter.com/>

<http://ngspice.sourceforge.net/>

<https://www.circuitlogix.com/student_version.php>

<https://www.systemvision.com/>

[https://www.circuitlab.com/editor/#](https://www.circuitlab.com/editor/)

<https://www.partsim.com/>

<http://www.analog.com/en/design-center/design-tools-and-calculators.html?domain=www.linear.com#LTspice>

<https://www.autodesk.com/products/eagle/overview>

# A ‘C’ Test: The 0x10 Best Questions for Would-be Embedded Programmers

<https://rmbconsulting.us/publications/a-c-test-the-0x10-best-questions-for-would-be-embedded-programmers/>

An obligatory and significant part of the recruitment process for embedded systems programmers seems to be the ‘C Test’. Over the years, I have had to both take and prepare such tests and in doing so have realized that these tests can be very informative for both the interviewer and interviewee. Furthermore, when given outside the pressure of an interview situation, they can also be quite entertaining (hence this article).

From the interviewee’s perspective, you can learn a lot about the person that has written or administered the test. Is the test designed to show off the writer’s knowledge of the minutiae of the ANSI standard rather than to test practical know-how? Does it test ludicrous knowledge, such as the ASCII values of certain characters? Are the questions heavily slanted towards your knowledge of system calls and memory allocation strategies, indicating that the writer may spend his time programming computers instead of embedded systems? If any of these are true, then I know I would seriously doubt whether I want the job in question.

From the interviewer’s perspective, a test can reveal several things about the candidate. Primarily, one can determine the level of the candidate’s knowledge of C. However, it is also very interesting to see how the person responds to questions to which they do not know the answers. Do they make intelligent choices, backed up with some good intuition, or do they just guess? Are they defensive when they are stumped, or do they exhibit a real curiosity about the problem, and see it as an opportunity to learn something? I find this information as useful as their raw performance on the test.

With these ideas in mind, I have attempted to construct a test that is heavily slanted towards the requirements of embedded systems. This is a lousy test to give to someone seeking a job writing compilers! The questions are almost all drawn from situations I have encountered over the years. Some of them are very tough; however, they should all be informative.

This test may be given to a wide range of candidates. Most entry-level applicants will do poorly on this test, while seasoned veterans should do very well. Points are not assigned to each question, as this tends to arbitrarily weight certain questions. However, if you choose to adapt this test for your own uses, feel free to assign scores.

**Preprocessor**

1. Using the #define statement, how would you declare a manifest constant that returns the number of seconds in a year? Disregard leap years in your answer.

#define SECONDS\_PER\_YEAR (60UL \* 60UL \* 24UL \* 365UL)

I’m looking for several things here:

(a) Basic knowledge of the #define syntax (i.e. no semi-colon at the end, the need to parenthesize etc.).

(b) A good choice of name, with capitalization and underscores.

(c) An understanding that the pre-processor will evaluate constant expressions for you. Thus, it is clearer, and penalty free to spell out how you are calculating the number of seconds in a year, rather than actually doing the calculation yourself.

(d) A realization that the expression will overflow an integer argument on a 16 bit machine – hence the need for the L, telling the compiler to treat the expression as a Long.

(e) As a bonus, if you modified the expression with a UL (indicating unsigned long), then you are off to a great start because you are showing that you are mindful of the perils of signed and unsigned types – and remember, first impressions count!

2. Write the ‘standard’ MIN macro. That is, a macro that takes two arguments and returns the smaller of the two arguments.

#define MIN(A,B) ((A) <= (B) ? (A) : (B))

The purpose of this question is to test the following:

(a) Basic knowledge of the #define directive as used in macros. This is important, because until the inline operator becomes part of standard C, macros are the only portable way of generating inline code. Inline code is often necessary in embedded systems in order to achieve the required performance level.

(b) Knowledge of the ternary conditional operator. This exists in C because it allows the compiler to potentially produce more optimal code than an if-then-else sequence. Given that performance is normally an issue in embedded systems, knowledge and use of this construct is important.

(c) Understanding of the need to very carefully parenthesize arguments to macros.

(d) I also use this question to start a discussion on the side effects of macros, e.g. what happens when you write code such as :

least = MIN(\*p++, b);

3. What is the purpose of the preprocessor directive #error?

Either you know the answer to this, or you don’t. If you don’t, then see reference 1. This question is very useful for differentiating between normal folks and the nerds. It’s only the nerds that actually read the appendices of C textbooks that find out about such things. Of course, if you aren’t looking for a nerd, the candidate better hope she doesn’t know the answer.

**Infinite Loops**

4. Infinite loops often arise in embedded systems. How does one code an infinite loop in C?

There are several solutions to this question. My preferred solution is:

while(1)

{

…

}

Another common construct is:

for(;;)

{

…

}

Personally, I dislike this construct because the syntax doesn’t exactly spell out what is going on. Thus, if a candidate gives this as a solution, I’ll use it as an opportunity to explore their rationale for doing so. If their answer is basically – ‘I was taught to do it this way and I have never thought about it since’ – then it tells me something (bad) about them. Conversely, if they state that it’s the K&R preferred method and the only way to get an infinite loop passed Lint, then they score bonus points.

A third solution is to use a goto:

Loop:

…

goto Loop;

Candidates that propose this are either assembly language programmers (which is probably good), or else they are closet BASIC / FORTRAN programmers looking to get into a new field.

**Data declarations**

5. Using the variable a, write down definitions for the following:

(a) An integer

(b) A pointer to an integer

(c) A pointer to a pointer to an integer

(d) An array of ten integers

(e) An array of ten pointers to integers

(f) A pointer to an array of ten integers

(g) A pointer to a function that takes an integer as an argument and returns an integer

(h) An array of ten pointers to functions that take an integer argument and return an integer.

The answers are:

(a) int a; // An integer

(b) int \*a; // A pointer to an integer

(c) int \*\*a; // A pointer to a pointer to an integer

(d) int a[10]; // An array of 10 integers

(e) int \*a[10]; // An array of 10 pointers to integers

(f) int (\*a)[10]; // A pointer to an array of 10 integers

(g) int (\*a)(int); // A pointer to a function a that takes an integer argument and returns an integer

(h) int (\*a[10])(int); // An array of 10 pointers to functions that take an integer argument and return an integer

People often claim that a couple of these are the sorts of thing that one looks up in textbooks – and I agree. While writing this article, I consulted textbooks to ensure the syntax was correct. However, I expect to be asked this question (or something close to it) when in an interview situation. Consequently, I make sure I know the answers – at least for the few hours of the interview. Candidates that don’t know the answers (or at least most of them) are simply unprepared for the interview. If they can’t be prepared for the interview, what will they be prepared for?

**Static**

6. What are the uses of the keyword static?

This simple question is rarely answered completely. Static has three distinct uses in C:

(a) A variable declared static within the body of a function maintains its value between function invocations.

(b) A variable declared static within a module [[1]](https://rmbconsulting.us/publications/a-c-test-the-0x10-best-questions-for-would-be-embedded-programmers/#footnote1), (but outside the body of a function) is accessible by all functions within that module. It is not accessible by functions within any other module. That is, it is a localized global.

(c) Functions declared static within a module may only be called by other functions within that module. That is, the scope of the function is localized to the module within which it is declared.

Most candidates get the first part correct. A reasonable number get the second part correct, while a pitiful number understand answer (c). This is a serious weakness in a candidate, since they obviously do not understand the importance and benefits of localizing the scope of both data and code.

**Const**

7. What does the keyword const mean?

As soon as the interviewee says ‘const means constant’, I know I’m dealing with an amateur. Dan Saks has exhaustively covered const in the last year, such that every reader of ESP should be extremely familiar with what const can and cannot do for you. If you haven’t been reading that column, suffice it to say that const means “read-only”. Although this answer doesn’t really do the subject justice, I’d accept it as a correct answer. (If you want the detailed answer, then read Saks’ columns – carefully!).

If the candidate gets the answer correct, then I’ll ask him these supplemental questions:

What do the following incomplete [[2]](https://rmbconsulting.us/publications/a-c-test-the-0x10-best-questions-for-would-be-embedded-programmers/#footnote2) declarations mean?

const int a;

int const a;

const int \*a;

int \* const a;

int const \* a const;

The first two mean the same thing, namely a is a const (read-only) integer. The third means a is a pointer to a const integer (i.e., the integer isn’t modifiable, but the pointer is). The fourth declares a to be a const pointer to an integer (i.e., the integer pointed to by a is modifiable, but the pointer is not). The final declaration declares a to be a const pointer to a const integer (i.e., neither the integer pointed to by a, nor the pointer itself may be modified).

If the candidate correctly answers these questions, I’ll be impressed.

Incidentally, one might wonder why I put so much emphasis on const, since it is very easy to write a correctly functioning program without ever using it. There are several reasons:

(a) The use of const conveys some very useful information to someone reading your code. In effect, declaring a parameter const tells the user about its intended usage. If you spend a lot of time cleaning up the mess left by other people, then you’ll quickly learn to appreciate this extra piece of information. (Of course, programmers that use const, rarely leave a mess for others to clean up…)

(b) const has the potential for generating tighter code by giving the optimizer some additional information.

(c) Code that uses const liberally is inherently protected by the compiler against inadvertent coding constructs that result in parameters being changed that should not be. In short, they tend to have fewer bugs.

**Volatile**

8. What does the keyword volatile mean? Give three different examples of its use.

A volatile variable is one that can change unexpectedly. Consequently, the compiler can make no assumptions about the value of the variable. In particular, the optimizer must be careful to reload the variable every time it is used instead of holding a copy in a register. Examples of volatile variables are:

(a) Hardware registers in peripherals (e.g., status registers)

(b) Non-stack variables referenced within an interrupt service routine.

(c) Variables shared by multiple tasks in a multi-threaded application.

If a candidate does not know the answer to this question, they aren’t hired. I consider this the most fundamental question that distinguishes between a ‘C programmer’ and an ‘embedded systems programmer’. Embedded folks deal with hardware, interrupts, RTOSes, and the like. All of these require volatile variables. Failure to understand the concept of volatile will lead to disaster.

On the (dubious) assumption that the interviewee gets this question correct, I like to probe a little deeper, to see if they really understand the full significance of volatile. In particular, I’ll ask them the following:

(a) Can a parameter be both const and volatile? Explain your answer.

(b) Can a pointer be volatile? Explain your answer.

(c) What is wrong with the following function?:

int square(volatile int \*ptr)

{

return \*ptr \* \*ptr;

}

The answers are as follows:

(a) Yes. An example is a read only status register. It is volatile because it can change unexpectedly. It is const because the program should not attempt to modify it.

(b) Yes. Although this is not very common. An example is when an interrupt service routine modifies a pointer to a buffer.

(c) This one is wicked. The intent of the code is to return the square of the value pointed to by \*ptr. However, since \*ptr points to a volatile parameter, the compiler will generate code that looks something like this:

int square(volatile int \*ptr)

{

int a,b;

a = \*ptr;

b = \*ptr;

return a \* b;

}

Since it is possible for the value of \*ptr to change unexpectedly, it is possible for a and b to be different. Consequently, this code could return a number that is not a square! The correct way to code this is:

long square(volatile int \*ptr)

{

int a;

a = \*ptr;

return a \* a;

}

**Bit Manipulation**

9. Embedded systems always require the user to manipulate bits in registers or variables. Given an integer variable a, write two code fragments. The first should set bit 3 of a. The second should clear bit 3 of a. In both cases, the remaining bits should be unmodified.

These are the three basic responses to this question:

(a) No idea. The interviewee cannot have done any embedded systems work.

(b) Use bit fields. Bit fields are right up there with trigraphs as the most brain-dead portion of C. Bit fields are inherently non-portable across compilers, and as such guarantee that your code is not reusable. I recently had the misfortune to look at a driver written by Infineon for one of their more complex communications chip. It used bit fields, and was completely useless because my compiler implemented the bit fields the other way around. The moral – never let a non-embedded person anywhere near a real piece of hardware! [[3]](https://rmbconsulting.us/publications/a-c-test-the-0x10-best-questions-for-would-be-embedded-programmers/#footnote3)

(c) Use #defines and bit masks. This is a highly portable method, and is the one that should be used. My optimal solution to this problem would be:

#define BIT3 (0x1 << 3)

static int a;

void set\_bit3(void) {

a |= BIT3;

}

void clear\_bit3(void) {

a &= ~BIT3;

}

Some people prefer to define a mask, together with manifest constants for the set & clear values. This is also acceptable. The important elements that I’m looking for are the use of manifest constants, together with the |= and &= ~ constructs.

**Accessing fixed memory locations**

10. Embedded systems are often characterized by requiring the programmer to access a specific memory location. On a certain project it is required to set an integer variable at the absolute address 0x67a9 to the value 0xaa55. The compiler is a pure ANSI compiler. Write code to accomplish this task.

This problem tests whether you know that it is legal to typecast an integer to a pointer in order to access an absolute location. The exact syntax varies depending upon one’s style. However, I would typically be looking for something like this:

int \*ptr;

ptr = (int \*)0x67a9;

\*ptr = 0xaa55;

A more obfuscated approach is:

\*(int \* const)(0x67a9) = 0xaa55;

Even if your taste runs more to the second solution, I suggest the first solution when you are in an interview situation.

**Interrupts**

11. Interrupts are an important part of embedded systems. Consequently, many compiler vendors offer an extension to standard C to support interrupts. Typically, this new key word is \_\_interrupt. The following code uses \_\_interrupt to define an interrupt service routine. Comment on the code.

\_\_interrupt double compute\_area(double radius) {

double area = PI \* radius \* radius;

printf(“nArea = %f”, area);

return area;

}

This function has so much wrong with it, it’s almost tough to know where to start.

(a) Interrupt service routines cannot return a value. If you don’t understand this, then you aren’t hired.

(b) ISR’s cannot be passed parameters. See item (a) for your employment prospects if you missed this.

(c) On many processors / compilers, floating point operations are not necessarily re-entrant. In some cases one needs to stack additional registers, in other cases, one simply cannot do floating point in an ISR. Furthermore, given that a general rule of thumb is that ISRs should be short and sweet, one wonders about the wisdom of doing floating point math here.

(d) In a similar vein to point (c), printf() often has problems with reentrancy and performance. If you missed points (c) & (d) then I wouldn’t be too hard on you. Needless to say, if you got these two points, then your employment prospects are looking better and better.

**Code Examples**

12. What does the following code output and why?

void foo(void)

{

unsigned int a = 6;

int b = -20;

(a+b > 6) ? puts(“> 6”) : puts(“<= 6”);

}

This question tests whether you understand the integer promotion rules in C – an area that I find is very poorly understood by many developers. Anyway, the answer is that this outputs “> 6”. The reason for this is that expressions involving signed and unsigned types have all operands promoted to unsigned types. Thus –20 becomes a very large positive integer and the expression evaluates to greater than 6. This is a very important point in embedded systems where unsigned data types should be used frequently (see reference 2). If you get this one wrong, then you are perilously close to not being hired.

13. Comment on the following code fragment?

unsigned int zero = 0;

unsigned int compzero = 0xFFFF; /\*1’s complement of zero \*/

On machines where an int is not 16 bits, this will be incorrect. It should be coded:

unsigned int compzero = ~0;

This question really gets to whether the candidate understands the importance of word length on a computer. In my experience, good embedded programmers are critically aware of the underlying hardware and its limitations, whereas computer programmers tend to dismiss the hardware as a necessary annoyance.

By this stage, candidates are either completely demoralized – or they are on a roll and having a good time. If it is obvious that the candidate isn’t very good, then the test is terminated at this point. However, if the candidate is doing well, then I throw in these supplemental questions. These questions are hard, and I expect that only the very best candidates will do well on them. In posing these questions, I’m looking more at the way the candidate tackles the problems, rather than the answers. Anyway, have fun…

**Dynamic Memory Allocation**

14. Although not as common as in non-embedded computers, embedded systems still do dynamically allocate memory from the heap. What are the problems with dynamic memory allocation in embedded systems?

Here, I expect the user to mention memory fragmentation, problems with garbage collection, variable execution time, etc. This topic has been covered extensively in ESP, mainly by Plauger. His explanations are far more insightful than anything I could offer here, so go and read those back issues! Having lulled the candidate into a sense of false security, I then offer up this tidbit:

What does the following code fragment output and why?

char \*ptr;

if ((ptr = (char \*)malloc(0)) == NULL) {

puts(“Got a null pointer”);

}

else {

puts(“Got a valid pointer”);

}

This is a fun question. I stumbled across this only recently, when a colleague of mine inadvertently passed a value of 0 to malloc, and got back a valid pointer! After doing some digging, I discovered that the result of malloc(0) is implementation defined, so that the correct answer is ‘it depends’. I use this to start a discussion on what the interviewee thinks is the correct thing for malloc to do. Getting the right answer here is nowhere near as important as the way you approach the problem and the rationale for your decision.

**Typedef**

15. Typedef is frequently used in C to declare synonyms for pre-existing data types. It is also possible to use the preprocessor to do something similar. For instance, consider the following code fragment:

#define dPS struct s \*

typedef struct s \* tPS;

The intent in both cases is to define dPS and tPS to be pointers to structure s. Which method (if any) is preferred and why?

This is a very subtle question, and anyone that gets it right (for the right reason) is to be congratulated or condemned (“get a life” springs to mind). The answer is the typedef is preferred. Consider the declarations:

dPS p1,p2;

tPS p3,p4;

The first expands to

struct s \* p1, p2;

which defines p1 to be a pointer to the structure and p2 to be an actual structure, which is probably not what you wanted. The second example correctly defines p3 & p4 to be pointers.

**Obfuscated syntax**

16. C allows some appalling constructs. Is this construct legal, and if so what does this code do?

int a = 5, b = 7, c;

c = a+++b;

This question is intended to be a lighthearted end to the quiz, as, believe it or not, this is perfectly legal syntax. The question is how does the compiler treat it? Those poor compiler writers actually debated this issue, and came up with the “maximum munch” rule, which stipulates that the compiler should bite off as big a (legal) chunk as it can. Hence, this code is treated as:

c = a++ + b;

Thus, after this code is executed, a = 6, b = 7 & c = 12;

If you knew the answer, or guessed correctly – then well done. If you didn’t know the answer then I would not consider this to be a problem. I find the biggest benefit of this question is that it is very good for stimulating questions on coding styles, the value of code reviews and the benefits of using lint.

Well folks, there you have it. That was my version of the C test. I hope you had as much fun doing it as I had writing it. If you think the test is a good test, then by all means use it in your recruitment. Who knows, I may get lucky in a year or two and end up being on the receiving end of my own work.

References:

In Praise of the #error directive. ESP September 1999.

Efficient C Code for Eight-Bit MCUs. ESP November 1988.

[1] Translation unit for the pedagogues out there.

[2] I’ve had complaints that these code fragments are incorrect syntax. This is correct. However, writing down syntactically correct code pretty much gives the game away.

[3] I’ve recently softened my stance on bit fields. At least one compiler vendor (IAR) now offers a compiler switch for specifying the bit field ordering. Furthermore the compiler generates optimal code with bit field defined registers – and as such I now do use bit fields in IAR applications.

# 5 Steps to Getting Started with Embedded Programing

I’ve been getting asked the question, “So how would I get started with embedded development?” more and more often lately.

This is actually a really tricky question. It’s not like, “How would I get started with Haskell?” or “How would I get started with Rust?” Embedded development is such a weird and diverse thing that it’s almost like asking, “How do I get started with programming?” except in an alternate universe where 128k is still a lot of RAM. I’m not sure where to even begin.

I think the people asking have one of two goals:

“I want to make my software affect physical things.”

“I want to learn what’s actually happening way down there at the bottom.”

If you’re mostly interested in the first goal, then I would recommend you check out the many good hobbyist embedded platforms available now ([Arduino](https://www.arduino.cc/), [Raspberry Pi](https://www.raspberrypi.org/), [Particle](https://www.particle.io/?redirected=true), etc…). They have lots of great communities to participate in and learn from.

However, if you’re more interested in the second goal, your task is a bit harder.

1. Learn C

For a variety of reasons, the vast majority of embedded toolchains are designed to support C as the primary language. If you want to write embedded software for more than just a few hobbyist platforms, your going to need to learn C (and hopefully maybe eventually [Rust](https://www.rust-lang.org/)).

2. Learn Some Basic Electronics

Don’t worry, you don’t need to take a class or anything. You just need a basic understanding of voltage, current, power, resistance, ohms law. You probably can get by with just a few online tutorials and some experimenting with online simulators and/or real circuits.

3. Get the Basic Equipment

Since this is embedded software and you’ll actually be interacting with the physical world, you’ll eventually need some physical equipment. You’ll at least need:

Soldering iron

Digital Multi-Meter (DMM)

A hardware debugger/ JTAG adapter (such as an ST-Link, or OLMEX adapter)

I also highly recommend getting a Logic Analyzer. My favorite is [from Saleae](https://www.saleae.com/), but they are many other cheaper ones.

4. Choose a Microcontroller and Toolchain

Okay, so now that we have the fundamentals, can we get to coding already?!

Almost. To actually get your programs running, you’ll need a microcontroller to run them, a compiler that can compile your programs for your target microcontroller, and other tools to load your programs onto your hardware and debug them.

I personally like the [STM32 family](http://www.st.com/web/en/catalog/mmc/FM141/SC1169) of microcontrollers. They are well supported by my favorite embedded toolchain: [arm-gcc](https://launchpad.net/gcc-arm-embedded) along with [openOCD](http://openocd.org/). This combination is not as user friendly as an Arduino, but it’s also suitable for many more real-world applications.

One good starter option is to get an [STM discovery kit](http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF259090); they are cheap, relatively accessible, and easy to get started with.

ARM is by far the most common architecture for embedded micros (especially 32bit micros), and arm-gcc can target pretty much all of them.

openOCD is an open source piece of software what will communicate with a hardware debugger and provide a gdb debugger server so you can load a program and step through your code running on target with gdb. You [don’t even need to run openOCD on the computer you’re developing and running gdb on](https://spin.atomicobject.com/2014/04/01/ethernet-adapter-jtag/).

5. Pick Components & Dig into Their Datasheets

Now you have enough to actually get started on something. All you have to do is pick out some components and then put them together! Some good places to look for components are [sparkfun](https://www.sparkfun.com/) and [adafruit](https://www.adafruit.com/). And for broader and cheaper selection, also [digikey](http://www.digikey.com/) and [mouser](http://www.mouser.com/).

Once you’ve found a few components that you think will do what you want, you’ll have to dig into their datasheets. Datasheets are essentially the manuals for electronic components. They are the key to figuring out how to use a component and to make sure it will, in fact, work for you application. Most of the questions you have about a component can be answered by its datasheets. But datasheets can be tricky. Tricky enough that I have my own 3 rules of embedded programming:

1st rule of embedded programming: **Read the datasheet.**

2nd rule of embedded programming: **Read the datasheet.**

3rd rule of embedded programming: **Don’t trust the datasheet.**

Datasheets are the source of all knowledge, but also not entirely intuitive or even accurate. I recommend reading [how to read a datasheet](http://www.egr.msu.edu/classes/ece480/capstone/read_datasheet.pdf) and [Sparkfun’s datasheet tutorial](https://www.sparkfun.com/tutorials/223) to help get started.

Whew, that’s quite a lot to go through, but will give you a pretty solid basis when you actually get through it. :)

<https://www.elprocus.com/basics-and-structure-of-embedded-c-program-with-examples-for-beginners/>

# Step by Step Learn Imbedded system

1. Learn C/C++. Practice it. Do some projects. Test yourself.
2. Learn Operating System Concepts. Work with Unix/Linux. Work on Multithreaded C programming using Pthreads. Learn Interprocess Communication. Work on Linux Kernel Programming. Work on device driver programming.
3. Learn Computer Architecture. Read 8086(x86), 8051, ARM architecture. As much as you can.
4. See how cross-compilation is done. Work with various IDEs and no IDE environment. Do some projects with arduino/ raspberry pi/embed board/STM32F4/PSoC4.
5. Go through your electronics knowledge. ADC, Sensors, relays, display interface, camera interface.,etc. Make use of oscilloscopes, multimeters, logic analyzers.,etc
6. Learn CADsoft eagle or OrCAD. Design your own PCB.
7. Read as many datasheets as possible. BLE module, WiFi module, ZigBee module and so on. Read the datasheet of mobile processors. And go through the schematics of development boards. Understand why some components are used. Revise your analog skills.
8. Learn any Hardware description Language. VHDL or Verilog.
9. Learn Web technologies- (HTML,JS),Databses-(Sqlite, Mysql), Java and Python too. It will be useful for IoT applications. Optional.
10. Learn simpleCV, OpenCV, fastCV to work on Image processing projects.
11. Work on Android Native Development Kit (NDK). Go through the various source codes of Kernel source releases of Motorola, Xiaomi or Oneplus.
12. Learn reverse engineering. Break into something. Debug using RX/TX-UART, JTAG. Debug processes in an OS.
13. Learn profiling/optimization of your code. Be good at data-structures and algorithms. Profiling tools - perf, Intel Vtune Amplifier.,etc.

# Learn from Scratch Microcontroller & Peripheral Driver Development for GPIO,I2C,SPI,USART using Embedded C Programming

**What Will You Learn?**

Understand Right ways of Handling and programming MCU Peripherals

Develop Peripheral drivers for your Microcontroller

Understand complete Driver Development steps right from scratch for GPIO,SPI,I2C and USART.

Learn Writing peripheral driver headers, prototyping APIs and implementation

Explore MCU data sheets, Reference manuals, start-up Codes to get things done

Learn Right ways of handling/configuring Interrupts for various peripherals

Learn about Peripheral IRQs/Vector table/NVIC interfaces and many

Learn about Configuration/status/Control registers of various Peripherals

Demystifying behind the scene working details of SPI,I2C,GPIOs,USART etc.

Explore hidden secretes of MCU bus interfaces, clock sources, MCU clock configurations, etc.

Understand right ways of enabling/configuring peripheral clocks/serial clocks/baud rates of various serial protocols

Learn about MCUs AHB, APB bus protocols

Learn about different MCU clocks like HCLK, PCLK, PLL,etc

Learn to capture/decode/analyze traces of serial protocols on Logic analyzer

Learn about Quick ways of debugging peripheral issues with case studies

**Requirements**

Basic knowledge of C and Micro controller could be added advantage but not mandatory.

**Brief Description By Course Instructor-**

This Course Demystifies the internal working of the Microcontroller and its Peripherals.

Coding for the Peripherals STEP-BY-STEP and Developing software drivers completely from scratch by extracting maximum information from Datasheets,Reference manuals, specs,etc

Protocol Decoding Using logic analyzers, Debugging, Testing along with Hints and Tips.

**Long Description:**

**Learning Embedded System Programming, can be a challenge**. Since it's a relatively little complex field, there's no real gold standard yet for how things are practiced, or taught, which can frustrate people who are just trying to learn new things and couldn’t able to connect the dots and this is the motivation behind creating this course to help engineers and students to learn different aspects of embedded systems by providing high quality advanced lectures with relatively low price.

Learn at your own pace using progressive method, with each of my short, info-packed lectures.

**Master Behind the Scene working !**I created this course because I believe your time is precious, and you shouldn't have to hunt around to get a practical foundation In Embedded System Programming. In this course, you are going to learn writing your own peripheral driver for most of the commonly used peripherals such as GPIOs, I2C, SPI, USART, etc. and interesting thing is that you are going to learn everything from scratch.

**No 3**

**rd**

**party libraries !**

**No blind coding !**

**Write your own Driver APIs by dealing with the Peripheral Registers of the MCU !**

Code and Implement APIs from scratch , diving into the datasheet and reference manual of the MCU. I will thoroughly explain how to extract the maximum information from datasheets, Technical Reference manuals to configure and handle peripherals. These techniques you can go and apply to any MCUs you have at your hand.

In this course I will walk you through step by step procedure how to Configure various Peripherals like GPIOs,SPI,USART,I2C by taking you into the reference manual and data sheet. We will develop fully working driver code, interrupt handlers, sample application everything from scratch to understand the big picture.

In each lecture, I assure you that, you will definitely learn something new that you can definitely use in your work or projects. You'll find yourself handling these peripherals with much more clarity and you will be able to quickly speculate and debug the problem and I’ll show you tricks and tips to debug the most common problems using debugging tools such as logic analyzers.

**This is not Arduino Style of programming!**

I believe **Arduino** is for quick prototyping of products but not for**Mastering Working of microcontrollers and its peripherals.**Unlike Arduino programming where you come up with quick solution and prototyping of products using third party libraries, this course is entirely different. In this course, no 3rd party libraries is used. Everything we will code by referring to the Technical reference manual of the MCU and create our own library. The Power of this approach is when things go wrong in your project work due to bugs, you can quickly able to speculate problem and debug like a pro. **If one thing me and my students are good at is "debugging". To achieve good debugging skills its very important you code by understanding how things works behind the scene but no by just blindly using some third party libraries and that’s the biggest TAKE away from this Course**.

The course is designed and explained in such a way that, it is generic across any kind of microcontroller. The code we develop can be used as templates to quickly come with peripheral driver for your MCUs on chip peripherals.

**Software/Hardware used:**In this course, the code is developed such a way that, It can be ported to any MCU you have at your hand. If you need any help in porting these codes to different MCUs you can always reach out to me! **The course is strictly not bound to any 1 type of MCU**. So, if you already have any Development board which runs with ARM-Cortex M3/M4 processor, then I recommend you to continue using it. But if you don’t have any Development board, then checkout the below Development boards.

1. **STM32F407xx based Discovery board ( This is the board used in this course )**

MCU Vendor : STMicroelectronics

2. STM32 Nucleo-64 development board( New in the market )

MCU Vendor : STMicroelectronic

3. FRDM-K64F: Freedom Development Platform

MCU Vendor : NXP

4. STM32F429IDISCOVERY with LCD

MCU Vendor : STMicroelectronics

**Who is the target audience?**

Professionals interested in exploring Embedded systems

Hobbyists and students who want to start their career in Embedded world

If you think about 'embedded' then think about taking this course. you will not be disappointe

This Course may not be suitable for those people who are looking for quick prototyping using boards such as Arduino

# Prototype Techniques

<https://www.capitaladvanced.com/uni-sip.htm>

<https://www.capitaladvanced.com/pdf%20documents/BULLETIN%20SI-116.pdf>

<https://www.capitaladvanced.com/products.htm>

<https://www.capitaladvanced.com/9000ser.htm>

<https://www.capitaladvanced.com/6000ser.htm>

<https://www.digikey.com/products/en#cat-45>

<https://www.arieselec.com/products/overview-correct-a-chip-sockets-adapters.htm>

<https://www.arieselec.com/products/overview-cable-assemblies.htm>

[Restriction of Hazardous Substances Directive](http://en.wikipedia.org/wiki/Restriction_of_Hazardous_Substances_Directive)(**RoHS**)

<https://learn.sparkfun.com/tutorials/how-to-solder-through-hole-soldering>

## How to Solder: Through-Hole Soldering

<https://learn.sparkfun.com/tutorials/how-to-solder-through-hole-soldering>

## Prototyping Methods

http://www.dartmouth.edu/~sullivan/prototyping.pdf