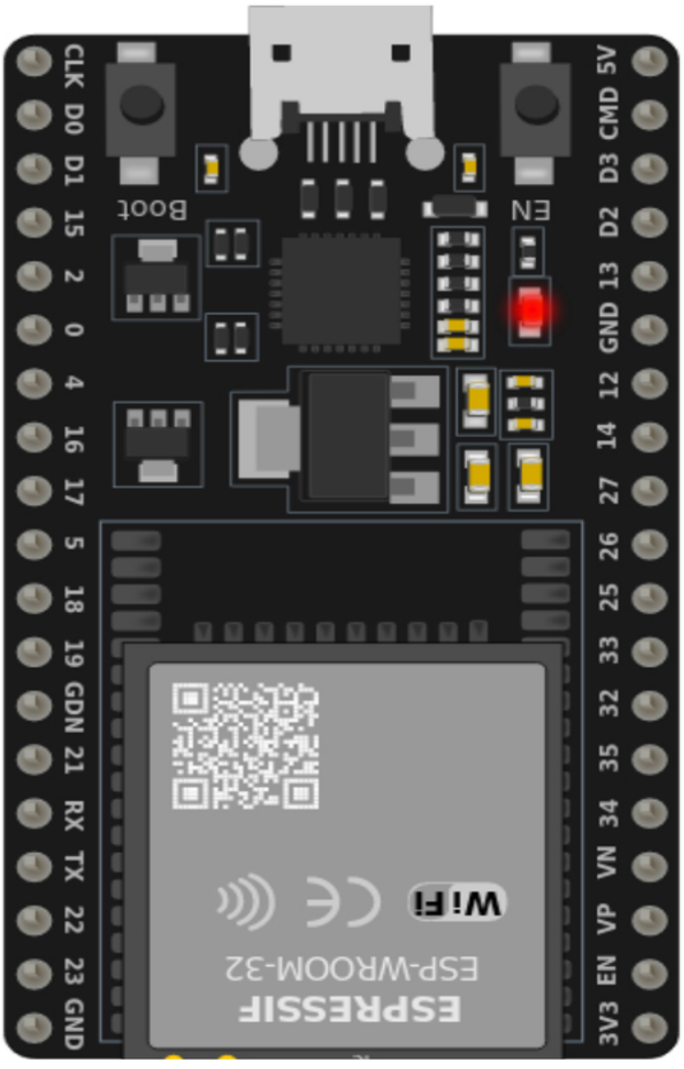
ESP32 HOME AUTOMATION WITH ARDUINO

Building a Smart Home with ESP32, Arduino, FreeRTOS, and Twilio SMS Messaging Real-Time Notifications for Your Smart Home



ESP32 HOME AUTOMATION WITH ARDUINO

Building a Smart Home with ESP32, Arduino, FreeRTOS, and Twilio SMS Messaging Real-Time Notifications for Your Smart Home

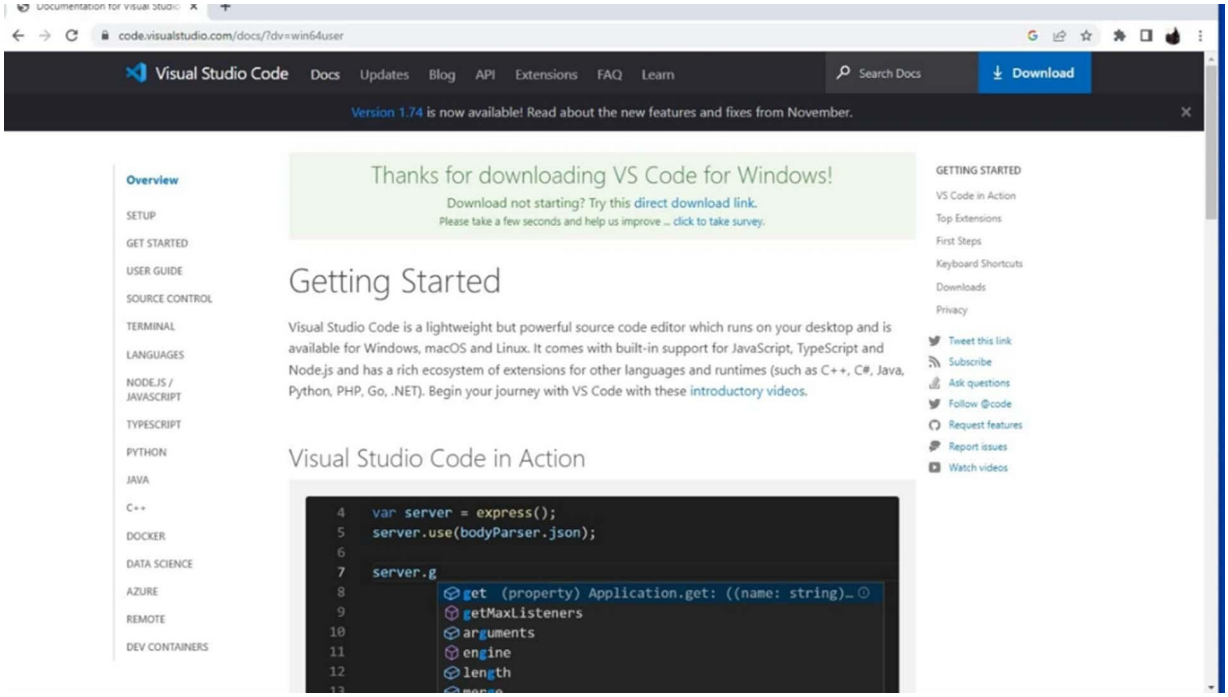
By

Janani Selvam

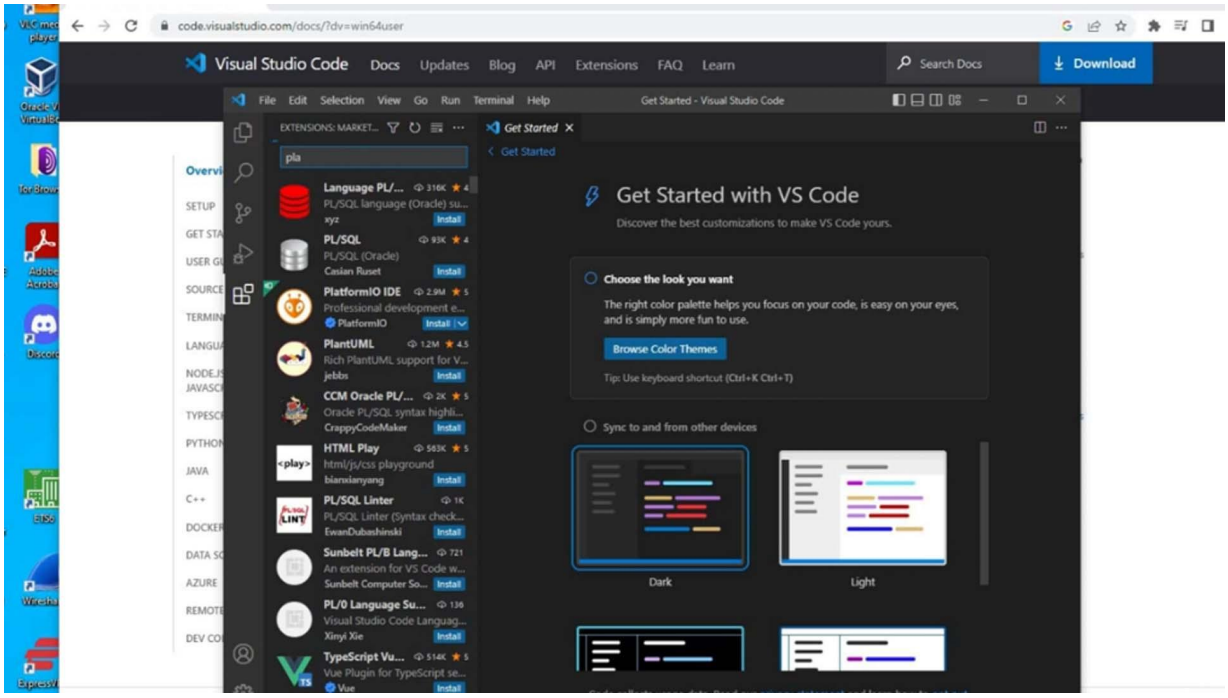
TABLE OF CONTENTS

INTRODUCTION INSTALLING VSCODE AND PLATFORMIO, LAUNCHING AN EXAMPLE PROJECT

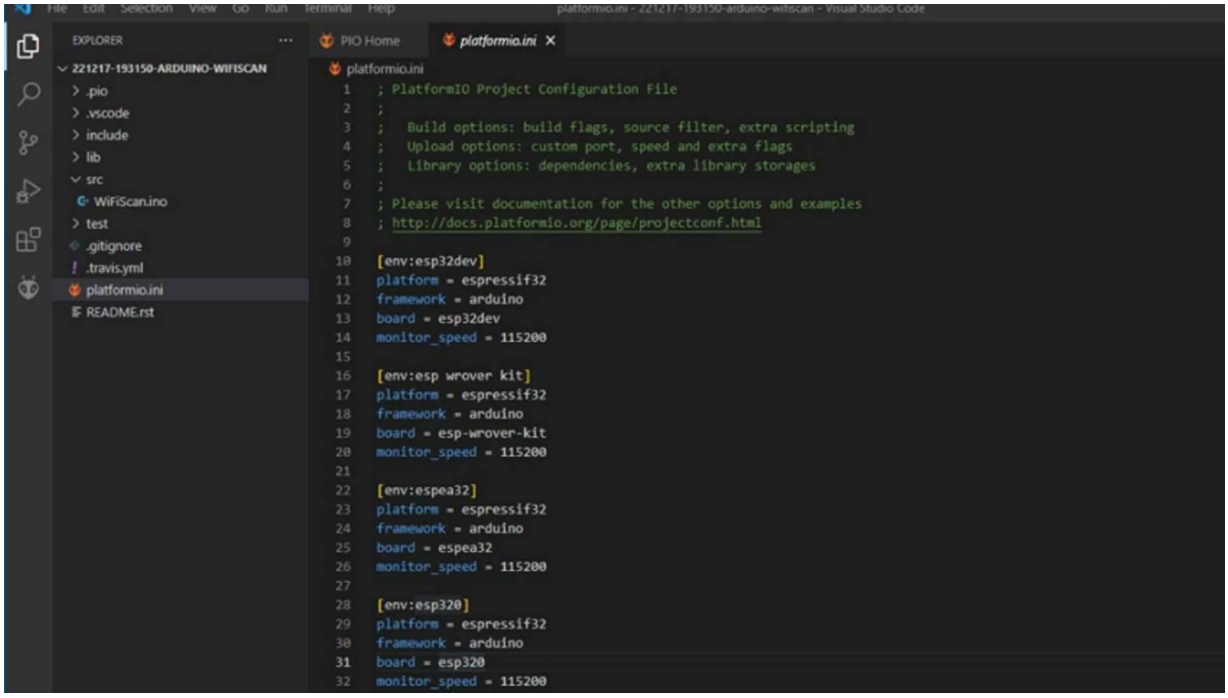
I will tell you how to write Arduino programs for ESP32 using the FreeRTOS real-time operating system. To develop programs, I suggest you use VS Code with the PlatformIO extension, which has more advanced functionality compared to the original Arduino IDE.



At the same time, all sketches made in VS Code are fully compatible with the Arduino IDE and can be easily transferred from one IDE to another. The same goes for libraries. All Arduino. PlatformIO extension. PlatformIO is installed. Let's restart VS Code.

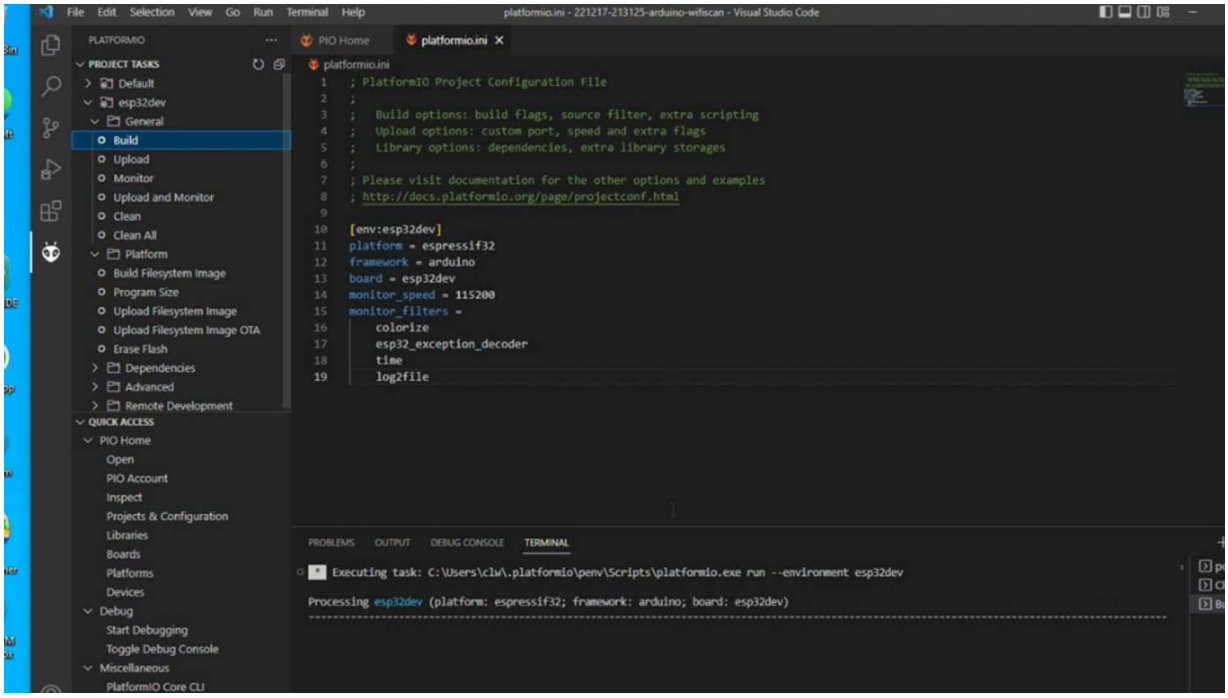


Here we can see the PlatformIO extension. Next, we'll open some Arduino projects. To do this, navigate to Open and here select Project Examples. And here we can select some examples. Let it be Arduino Wi-Fi Scan. Import. Yes, I trust the author. Let's enlarge the window a bit. Let's take a look at the PlatformIO project. Of... Arduino.ino file. This one. And a PlatformIO.ini file. This one. We can see the IntelliSense warning that it does not support the .ino file. So we can rename it to CPP.

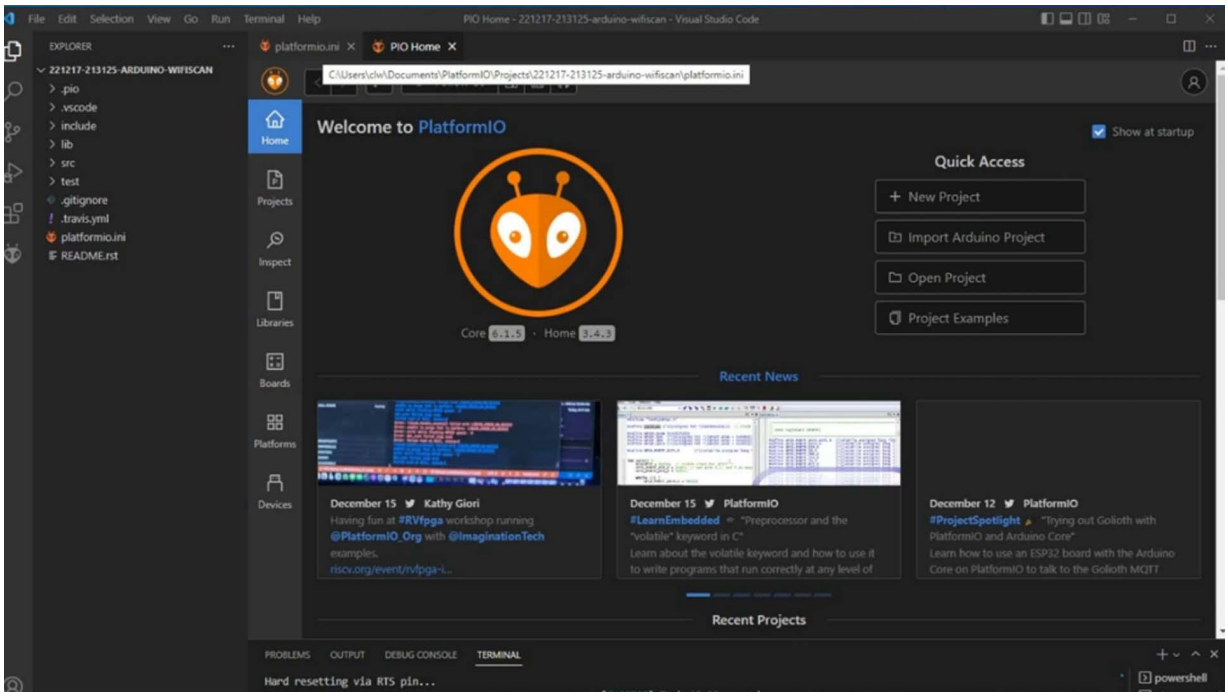


Now IntelliSense will work. Let's look at platformio.ini file. We can see 4 different configurations of 4 different boards in this file. I'm using the first version of ESP32. This is the first configuration. We don't need the others, I can just delete them. Let's add another parameter to this configuration. With this parameter we'll enable logging of the monitored text information with time indication and decoding of ESP32 exceptions. Further I'll show how to use it. Let's compile this project and see how it works.

To do this, I go to platform.io icon and first I choose the client and then build. Build is successful. Cleaning is necessary only at the first compilation. Then it's not needed. Next we can upload the program to ESP32. Loading. Let's open the terminal or serial monitor. Restart ESP. Scan start. This is my Wi-Fi net. Again the scan starts. As we can see, everything is working. Let's stop monitoring and look at the log file. Log file is here. I can open it. Here is everything that was written to monitor.



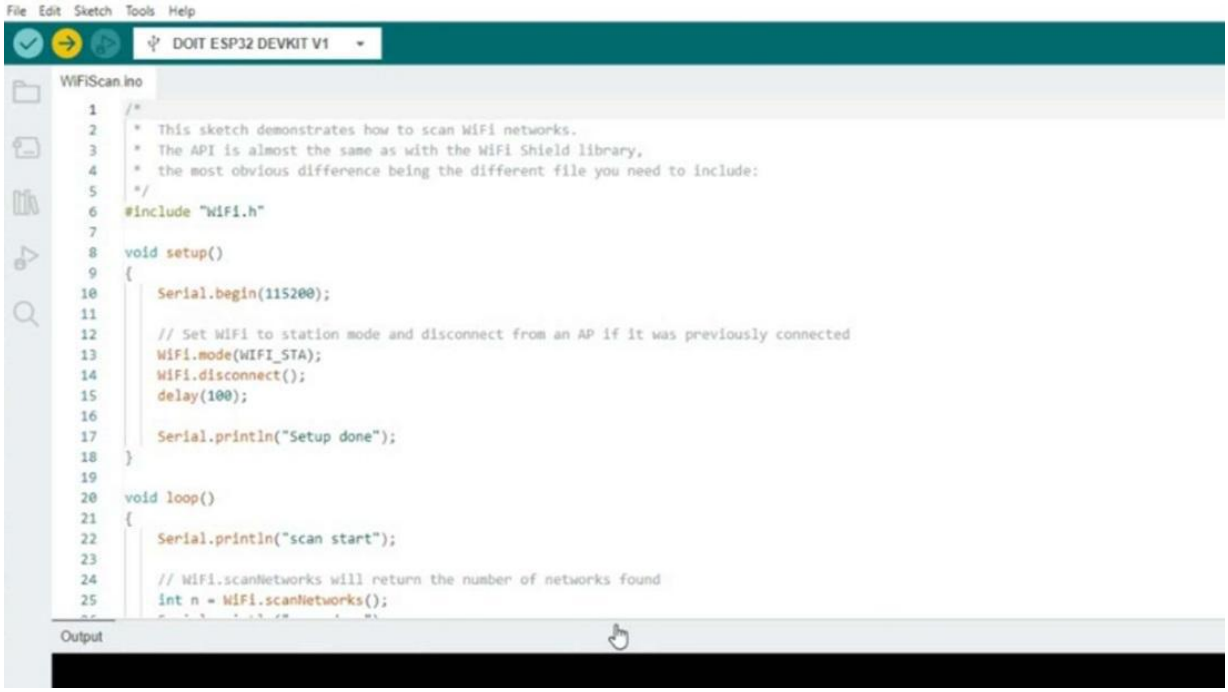
Let's move this project to the original Arduino IDE. First let's see where platform.io projects are located. Hover the cursor over the file header and see the path to it. This path to it. Open this folder and copy the CPP file. This is my platform.io project. This is a Wi-Fi scan. SRC. Wi-Fi scan. CPP. Copy. And paste it to any other folder.



I paste it here. Let's rename it back to Inno. Ok, now we have an Arduino file. Double click it. I click OK and Arduino creates a folder with the name of our file. Wi-Fi scan. Wi-Fi scan. We can compile now.

Ah, we have to select a board. I have to do it. Yes, P32, DevKit version 1. Ok, again compile.

Okay, it's compiled. Let's upload it to ESP32. First, select the port and upload. And then we start the serial monitor and restart ESP. As we can see, it's working. Scan done. This is my Wi-Fi net. Let's open any example program in Arduino IDE and import it in platform IO. Here are many examples. I'll open something simple.



I think timer. Repeat timer. Let's save it. Arduino. Okay, repeat timer. Documents. Arduino. The path. Repeat timer. Okay, and close this all. This is also close. And then we import this project. Platform IO. Open. Import Arduino project.

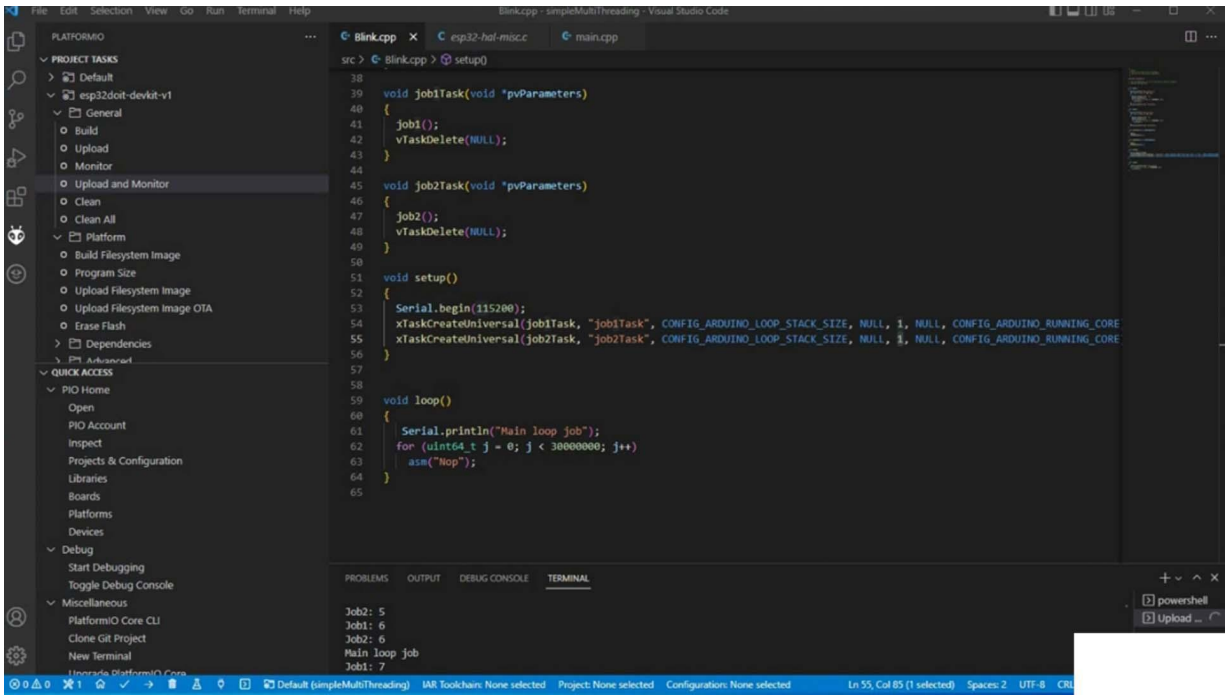
Let's find the path to our project. Users. Name of PC. Documents. Arduino. And our repeat timer. Import. Ah, no, no, no. Select the board. My board is a DUET. This one. And set this. Use libraries installed by Arduino IDE. If we set this, we can use all libraries

installed in Arduino. And now import. Now we can see the repeat timer. Let's look at the platform IO.ini file. Here we can see that the path of the Arduino libraries has been added. We also need to specify the monitor settings. Now we can compile and upload. And then upload and monitor.

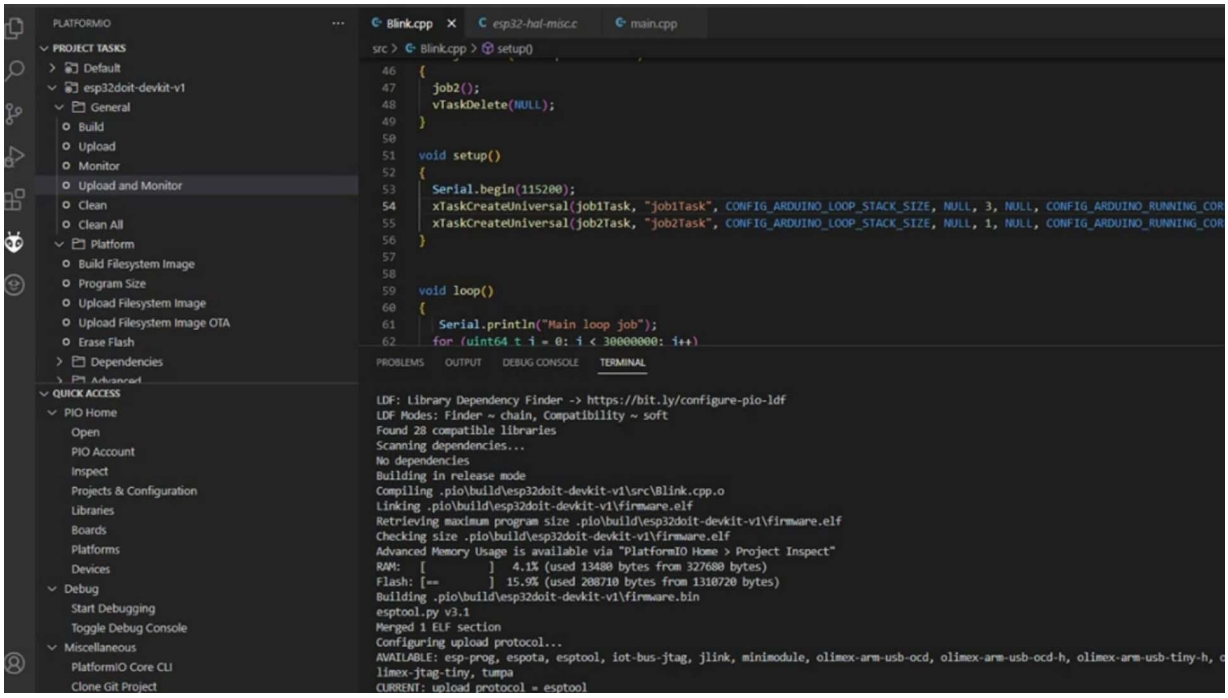
First compiling, then uploading, and then starting to monitor. Reset ESP32. Here we can see that the timer is working. In the next lecture we will write a simple program with Arduino and FreeRTOS.

PRIORITIES OF TASKS

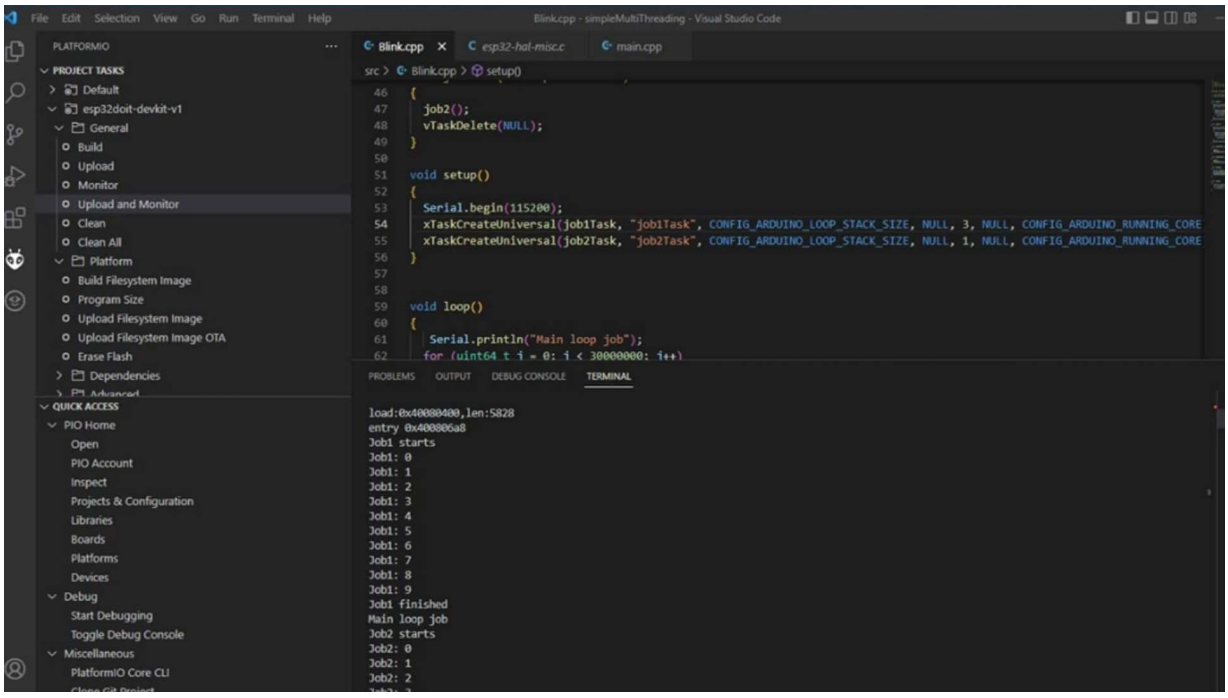
Now let's take a closer look at the tasks in this program. As mentioned in the previous lecture, three tasks are launched in the program. The first starting task is the Arduino task. It's started in the file main.cpp. Let's look at this parameter. This is the priority of the task. The lower this value, the lower the priority in FreeRTOS. Now let's look at the priority of the tasks we create. It's the same as the Arduino task. Here is also one.



Therefore, all three running tasks will be executed sequentially, switching with each clock cycle of the task scheduler. If you look at the output of the program in the terminal, we will see that all three of our tasks are performed simultaneously. This one. Job 1, Job 2, Job 2. Main loop job. Job 1, Job 2, Job 1, Job 2. And so on. Now let's raise the priority of one of the tasks and see what changes. I will set the priority for Job 1 equal to 3. This one. 3. Update and monitor. Sorry, I disconnected the ESP. I connect it and upload and monitor again.



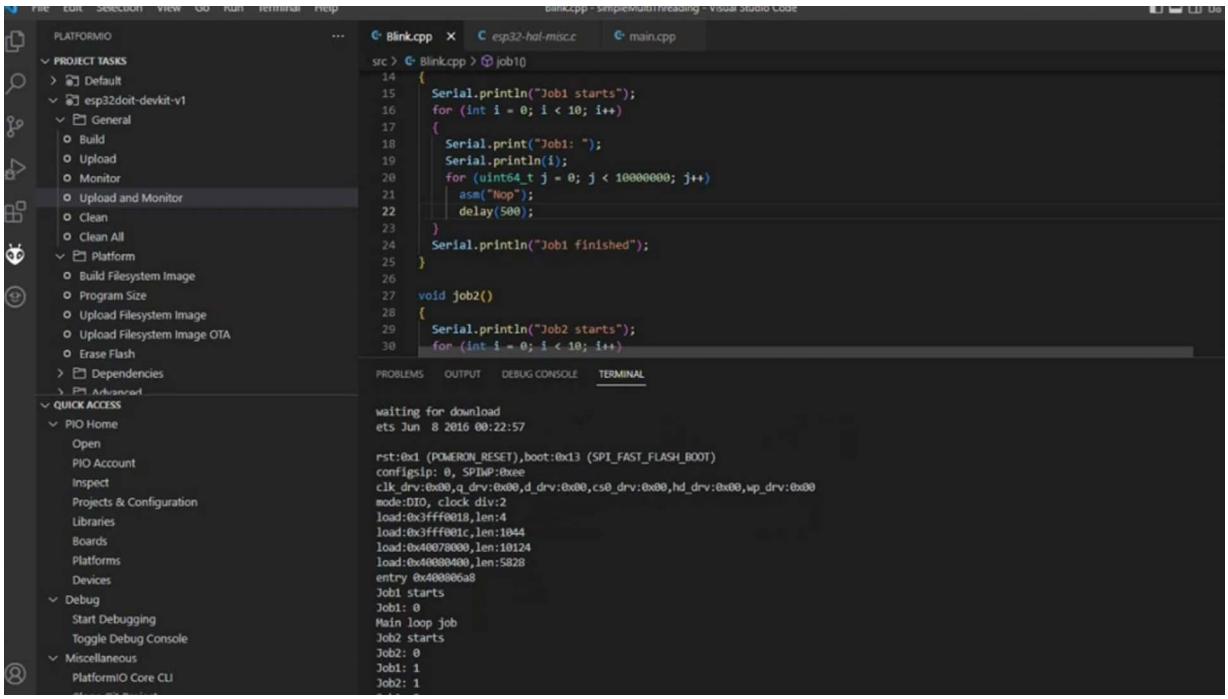
Restart ESP. As we can see, at first only Job 1 is executed. And then, after it finishes, the Job 2 and Arduino loop tasks begin to work. Thus, it turns out that a task with a high priority completely displaces tasks with a low priority. Thus, in this case, tasks with a low priority will not work until the task with a high priority finishes working.



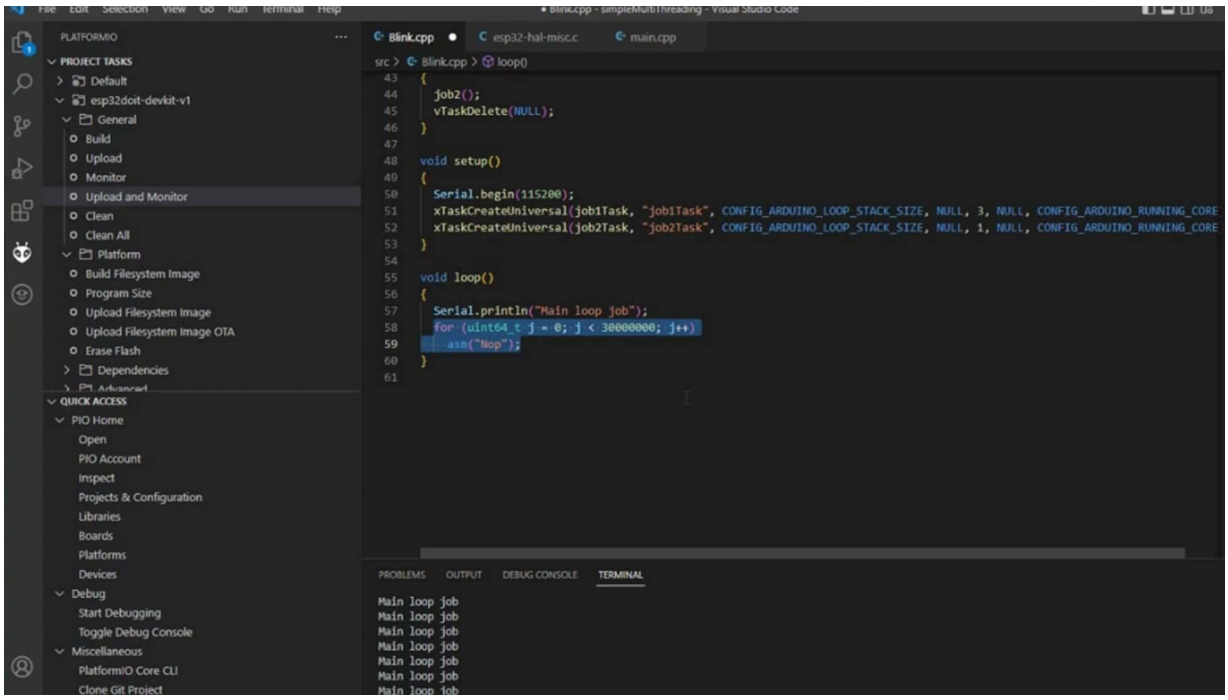
But what if tasks with a low priority should also work when tasks with a high priority are working? In this case, the work of a task with a high priority can sometimes be blocked.

To do this, there are various functions in FreeRTOS, the simplest of which is a delay for the time specified in milliseconds.

500 ms delay to the task of the first job. We can use the Arduino delay function here. Upload and monitor. Restart ESP. As we can see, all three tasks have started working, despite the fact that one of them has a higher priority. Thus, we blocked job 1 by delay, allowing the work of other tasks.



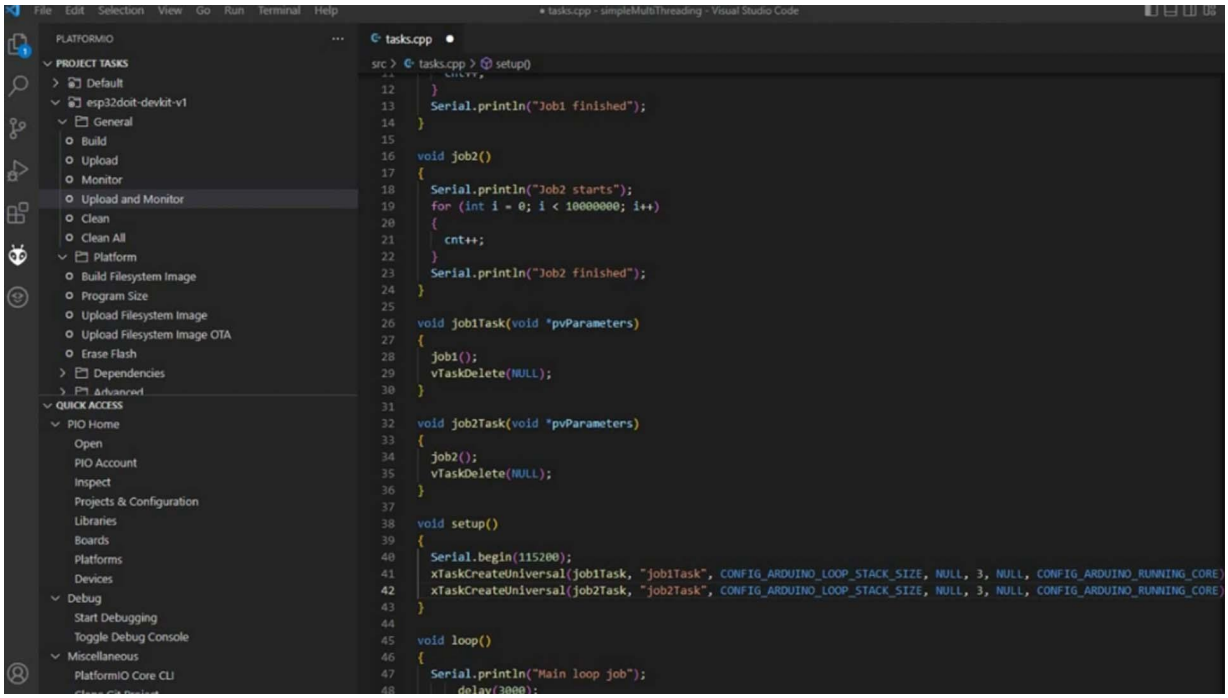
As I said, we used the usual Arduino delay function. Let's see what it has inside. To do this, press the left control and click on it.



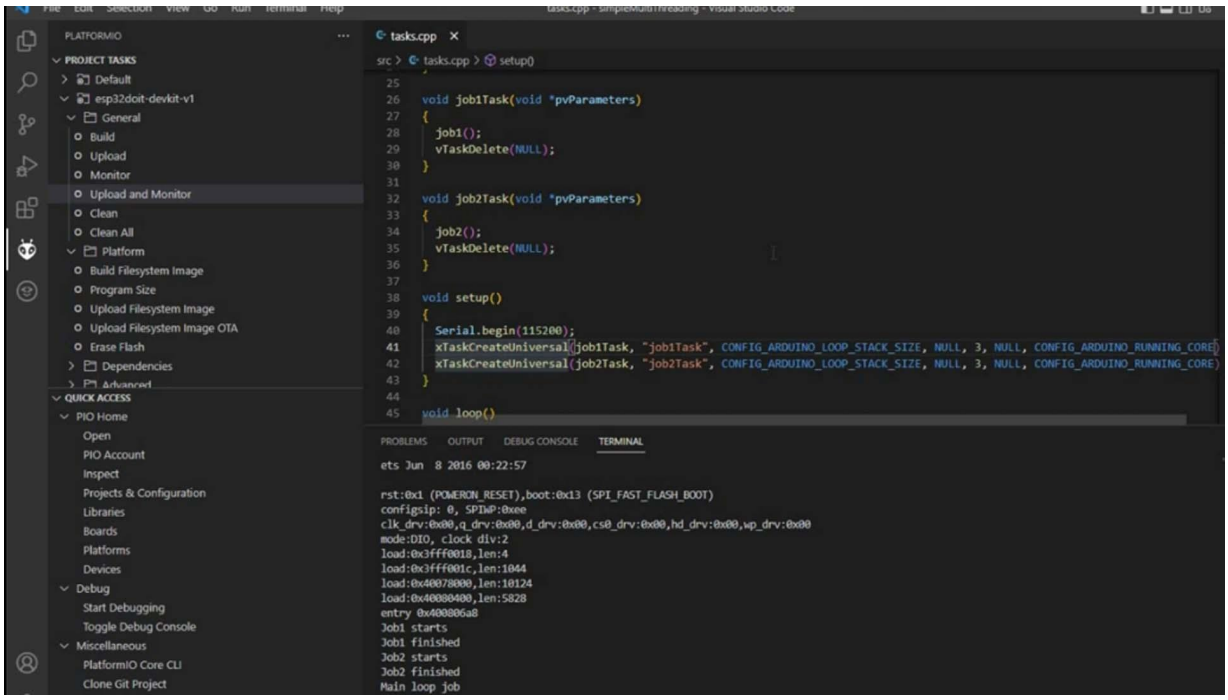
And here we see the FreeRTOS delay function, which performs the delay not through the for loop, like this delay for loop, but by blocking the task for a given number of ticks. This blocking allows tasks with a lower priority to work. Now let's replace the delays on the for loops with a delay function. Let it be 1 second delay. And here. Let it be here for 3 seconds. Upload and monitor. Restart ESP. As we can see, all three tasks are executed simultaneously, despite the fact that one of them has a higher priority.

MUTEXES

Let's change our program now. Let the tasks in it increment the same variable. But first I'll rename the file blink.cpp because we no longer use the red. Let it be called tasks.cpp. Rename tasks.cpp. Also we don't need this. And we don't need this. Let's create one variable of type int and call it cnt. That means counter. This one. And let our program increment this counter 10 million times in each task. 10 million. And here 10 millions. Here cnt++. Let's give these tasks the same priority, equal to 3. At the same time the Arduino... 1.

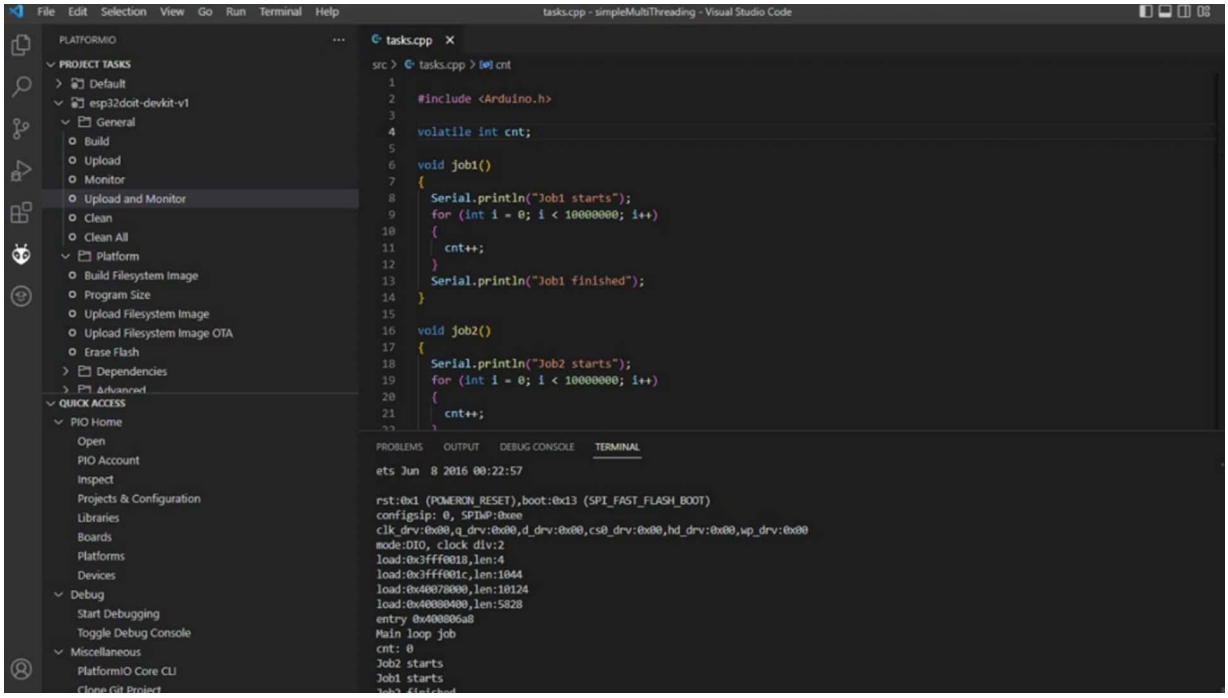


Therefore our tasks will work first and then the Arduino loop will start. Also in the Arduino loop let's output the resulting value of the counter cnt to the monitor. And then stop the program. Compile the program and see how it works. Restart ESP. As we can see both tasks did not start at the same time. But the result of the increment of the counter is correct, 20 million. This one. Why did not the tasks start simultaneously? Let's look at the sequence of launching tasks. First, the Arduino setup task starts with a low priority of 1.



Then it creates a job 1 task with a higher priority of 3. This one. This task displaces the Arduino task. In this case the job 2 task will not be created until the job 1 task finishes working. Therefore in order for the second task to start, the first one needs to be blocked for a short period of time. By inserting a delay of 1 ms at its beginning. I insert a delay here. And in order for the tasks to start at the same time, the same delay must be inserted into the second task. I'll insert it here.

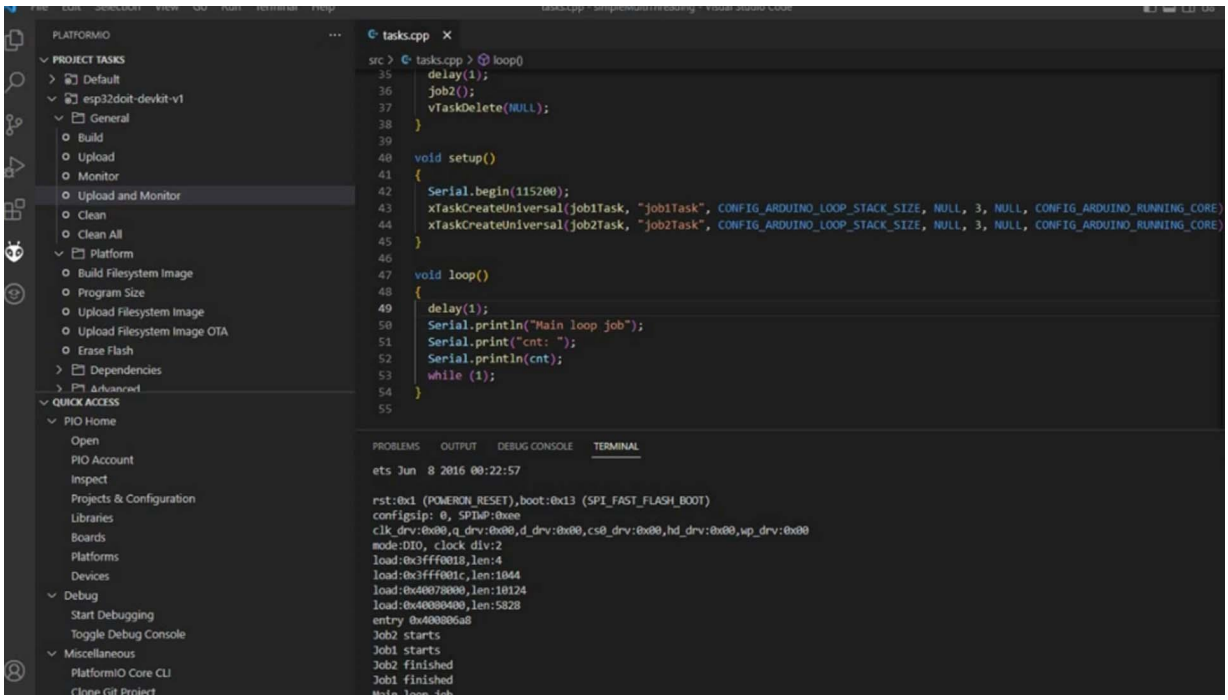
Also, in order for the compiler not to optimize the counter, we need to put it with the volatile keyword. It's here. Ok, now we can compile, upload and monitor.



Restart ESP. Now we see that the tasks are running simultaneously. Starts both and finishes both. But an Arduino task is started before them, which outputs the counter value to the monitor before it has changed. Here. Therefore we also need to insert a delay in the Arduino loop. Here.

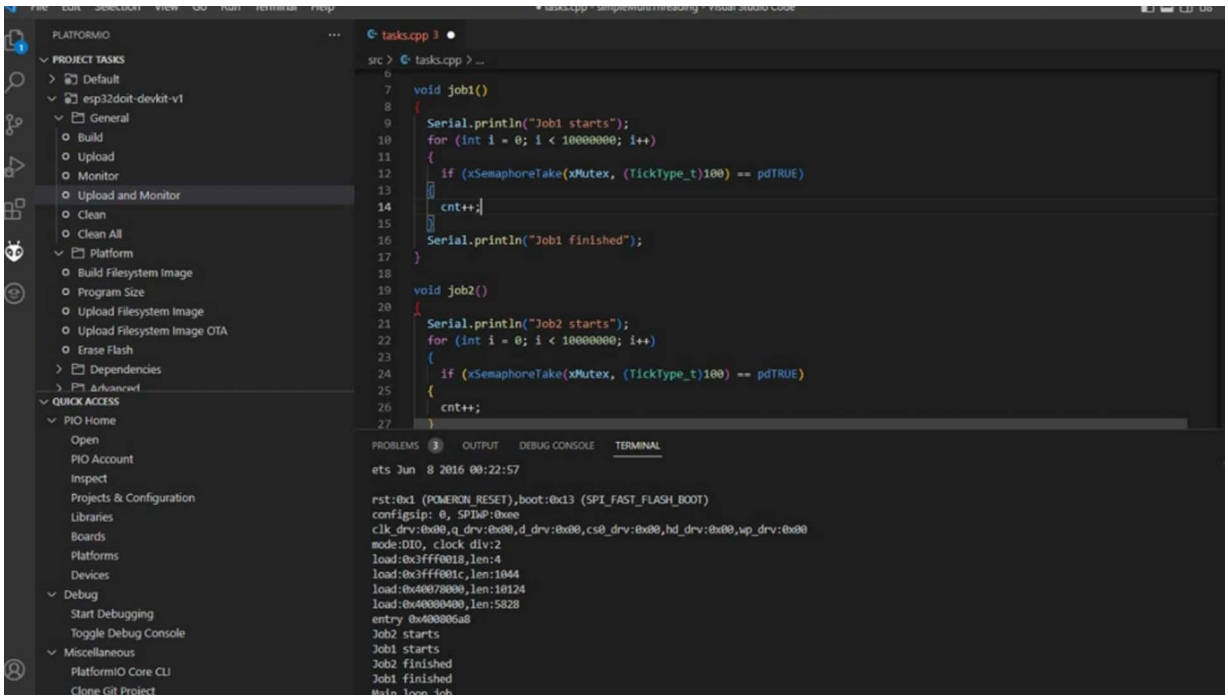
Upload and monitor. Restart ESP. Now our program works correctly. Both tasks start at the same time. At the end the result is displayed on the monitor. Only the result itself does not match the expected 20 million. Why did this happen? This happens because both tasks access the same cell in the memory. At the same time they can interrupt each other. In such a way that one task starts the operation but does not have time to finish because another task starts working with this variable. Thus, as a result, an incorrect value is written to the variable.

To solve this problem, tasks need to organize access to this memory cell in turn. That is, when one task is working with this variable, the second one should go into standby mode so that it does not conflict with the first one.



Mutexes are used for this purpose. Let's create a mutex and test how it works. To define a mutex in FreeRTOS, the semaphore handle T type is used. Let's call our mutex xMutex. Now we have to create it. This is done by the xSemaphore Create Mutex function. It should be done in an Arduino setup. And now we can use it in our tasks.

Now, when a task needs to access a shared resource, in our case a memory cell, it must first take the mutex. And if the mutex is taken by another task, then the current task will be blocked until the mutex is released. At the end of working with the resource, the task must release the mutex. Taking a mutex is performed by the xSemaphoreTake function in FreeRTOS. To do this, I insert this line here. Here and here. With this function we take a mutex. Here. We also need to give it away. The xSemaphore Give function is used for this.



I insert it here and here. Let's look at the function of taking mutex. Two parameters are passed to it. This is the mutex itself and the timeout in milliseconds. If the mutex is taken by another task, the task will be blocked until the mutex is released or for the time passed to it in this parameter. Therefore, in the case of a successful mutex taking, the function returns pdTrue. And in the case of a timeout exit, pdFalse. Now compile, upload and monitor. GSP starts. And it's been working for a long time because we have mutex. Let's wait.

The program finished its work with the correct counter value. 20 millions. Only it worked for almost 2 minutes. As we can see, using a mutex allows sharing access to a resource. But at the same time slows down the program. This should be taken into account when using it. In the next project we'll run the same tasks on different ESP32 cores.

EXAMPLE DUMMY CODE

Mutexes (short for "mutual exclusion") are synchronization primitives used in multi-threaded programming to prevent multiple threads from accessing shared resources simultaneously. Here's an example in C++ that demonstrates how to use mutexes to protect a shared resource (a simple counter) from concurrent access:

Cpp

```
#include <iostream>
#include <thread>
#include <mutex>

std::mutex mtx;  // Declare a mutex

int sharedCounter = 0;

void incrementCounter(int id, int iterations) {
    for (int i = 0; i < iterations; ++i) {
        mtx.lock();  // Lock the mutex to protect the shared
resource
        sharedCounter++;
        mtx.unlock();  // Unlock the mutex when done
    }
}

int main() {
    const int numThreads = 4;
    const int iterations = 10000;
```

```
    std::thread threads[numThreads];

    for (int i = 0; i < numThreads; ++i) {
        threads[i] = std::thread(incrementCounter, i,
iterations);
    }

    for (int i = 0; i < numThreads; ++i) {
        threads[i].join();
    }

    std::cout << "Final Counter Value: " << sharedCounter
<< std::endl;

    return 0;
}
```

In this example, we have a shared integer sharedCounter that multiple threads will try to increment. To ensure that the counter is updated safely, we use a std::mutex named mtx to lock and unlock access to the shared resource. Each thread locks the mutex before accessing the counter, increments it, and then unlocks the mutex when it's done.

By using the mutex, we guarantee that only one thread at a time can access and modify the sharedCounter, preventing data races and ensuring thread safety.

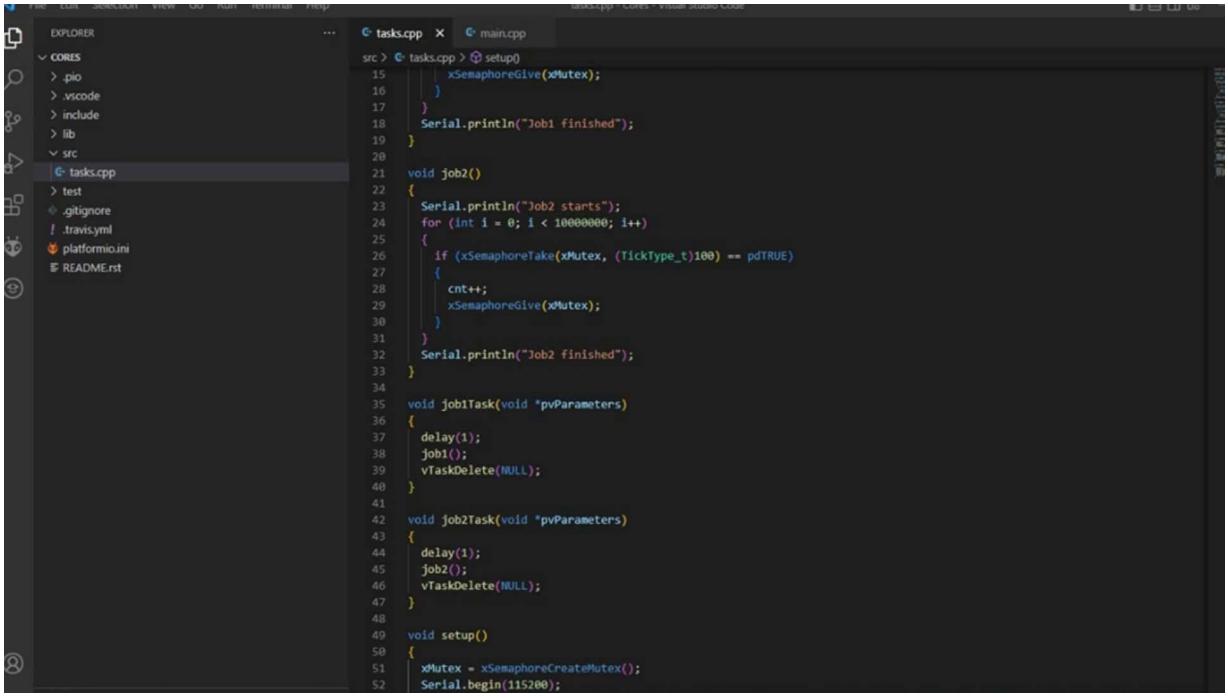
Keep in mind that using mutexes can introduce potential bottlenecks and performance issues, and it's important to use them judiciously in your multithreaded applications. In practice, you might also consider using higher-level abstractions like std::lock_guard for safer and more convenient mutex management.

SPINLOCK, CRITICAL SECTION, MULTICORE

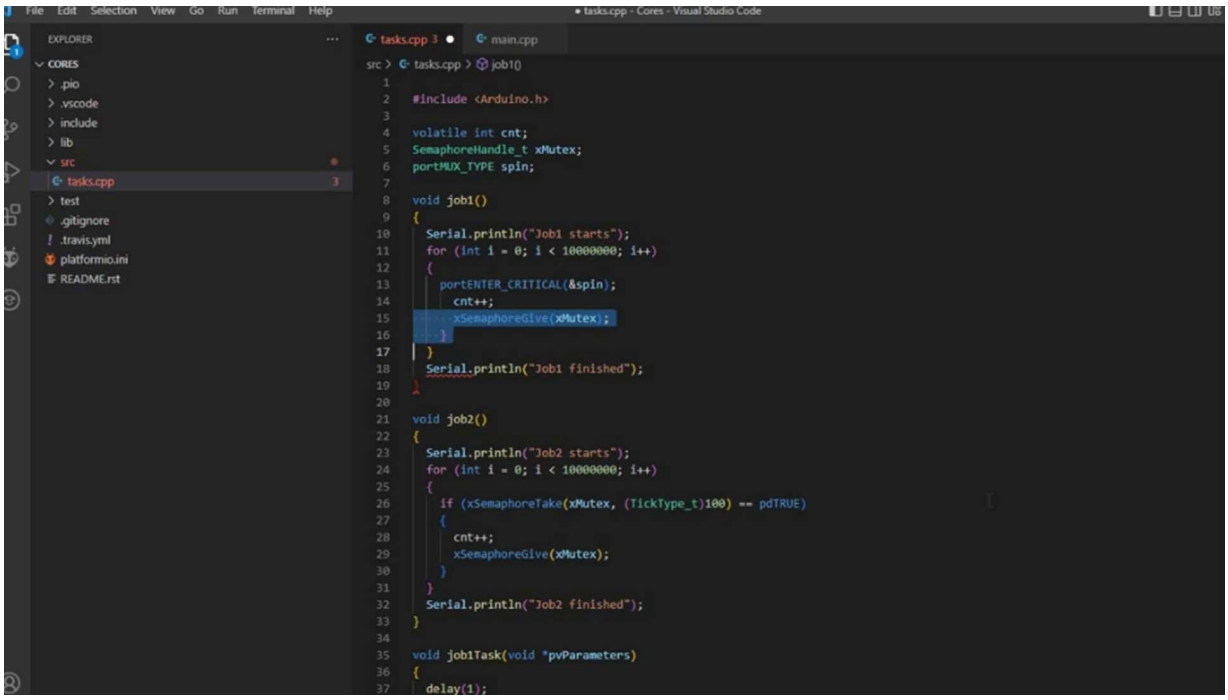
The previously discussed synchronization of tasks using mutexes slows down the program when they are called frequently. This is because the process of blocking and unblocking the thread is quite expensive. And if I do it often, the performance of the program will decrease. Let's now look at another synchronization method based on critical sections and spinlocks. A critical section is a section of the test code for which other tasks are stopped and most hardware interrupts are disabled. I will talk about interrupts in the following lectures.

On a processor with one core, it is enough to disable interrupts and the task schedule. To enter the critical section. But in ESP32, interrupts can be disabled from one core only on the same core. And they will remain enabled on the other. Therefore, a spinlock is added to the entrance to the critical section. Spinlock is similar in functionality to a mutex. It also needs to be taken and given. The only difference is that when the mutex could not be taken, the task is blocked.

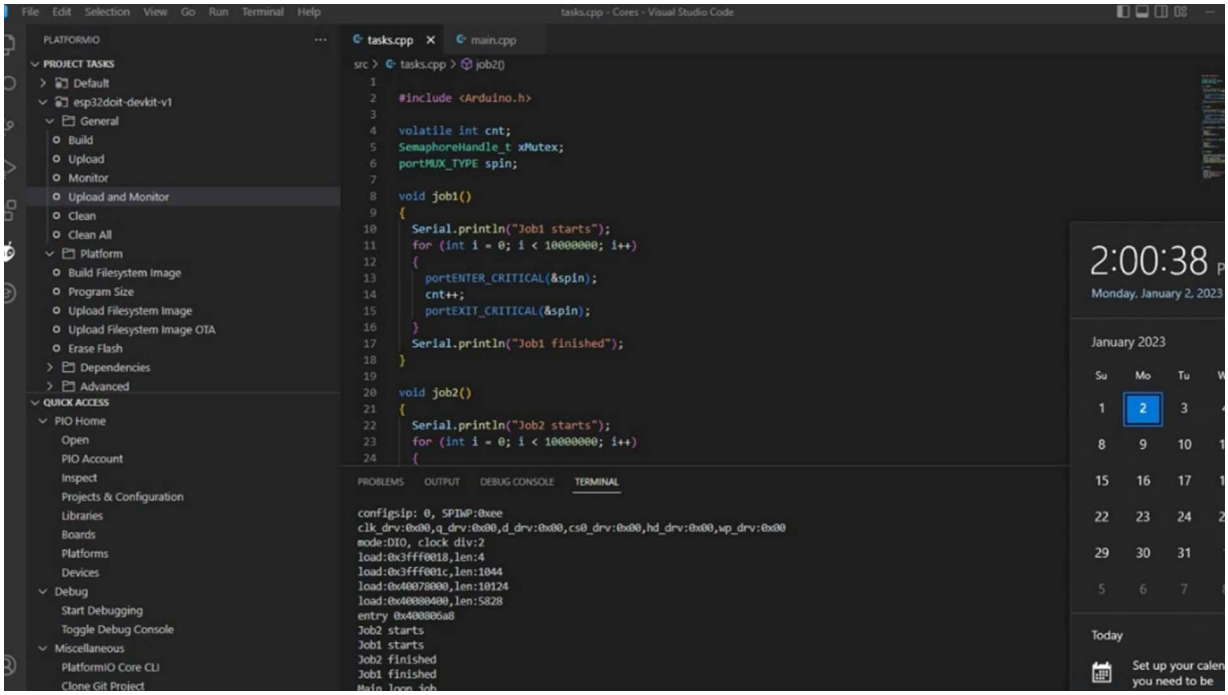
And if it was not possible to take the spinlock, then the blocking does not occur. And the program continues to check whether it has been released.



Therefore, for short sections of code with frequent access to a shared resource, it is preferable to use spinlock. Let's create a spinlock and replace the mutex with a critical section. The port mux type is used for the spinlock. I'll call it simply spin. Then we have to create a spinlock. This creates a spinlock function. And then we replace mutex by a critical section. I removed this. And here enter critical.

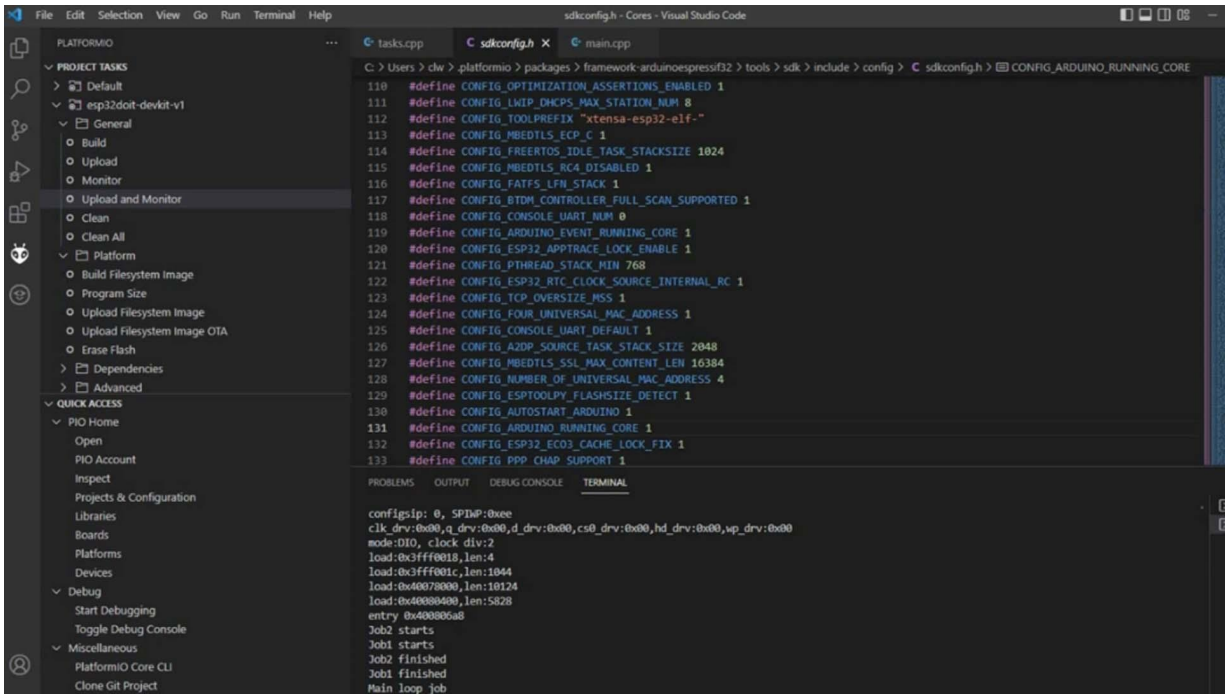


And here exit critical. And same for job 2. Let's compile it. Restart ESP. Waiting for the job to finish. It's finished. As we can see, the program performance has increased significantly. Let's approximately measure the time of its execution. The time can be detected by the Windows clock. I restart the ESP32.



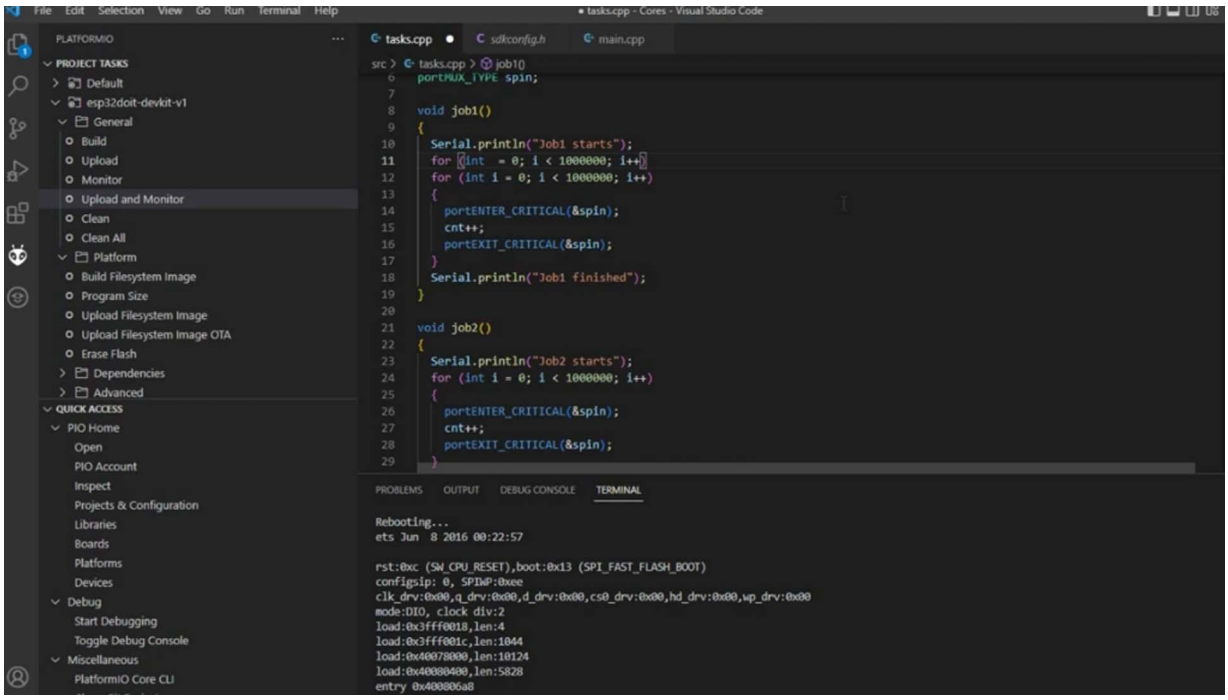
In 40 seconds it's restarted. Let's look when it's finished. The program completed its work within 20 seconds, which is much faster than the previous version. Now let's run one of the tasks on another core.

Let's see which core the Arduino is running on. To do this, press CTRL and click on the last parameter in the task creation function. We can see that Arduino runs on core number 1.



And we will launch Job 2 on core 0, another core. Upload and monitor. Restart ESP. As we can see, the program is already restarting. This happened because both of our tasks have a higher priority than the system task that should restart the watchdog timer. Therefore, our task should sometimes give time for tasks with a low priority. Therefore, let's call the delay for 1 millisecond 10 times in each of our tasks. Let's modify our code and insert a delay of 1 millisecond 10 times. First, I'll make 1 million here instead of 10 million. And here.

And I'll add 1 for the cycle. And here delay for 1 millisecond. And same for Job 2. And here should be 10, not 1 million. This one and here. Now we can compile and monitor. Job starting. Now we see that ESP32 is no longer rebooting.



But the result appeared on the monitor before the end of our tasks. So let's add a delay of 15 seconds before the output of the result. Here 15 seconds. Upload and monitor again. It's working. Let's measure the time of its execution. I'll start in 20 seconds. Restart ESP. And we are waiting for our jobs to finish.

Ok, it was 14 seconds. Thus, our program runs even faster on two cores. This is because no time is wasted switching between tasks.

EXAMPLE DUMMY CODE

provide you with an example that combines spinlocks, critical sections, and demonstrates their use on a multi-core system. In this example, we'll use C++ and the C++11 standard library to create a simple program with two threads running on multiple CPU cores, contending for a shared resource within a critical section using a spinlock.

Cpp

```
#include <iostream>
#include <thread>
#include <atomic>

// Define a simple spinlock using std::atomic_flag
std::atomic_flag spinlock = ATOMIC_FLAG_INIT;

const int iterations = 10000;
int sharedResource = 0;

void incrementSharedResource(int threadID) {
    for (int i = 0; i < iterations; ++i) {
        // Acquire the spinlock
        while
(spinlock.test_and_set(std::memory_order_acquire)) { }

        // Critical Section
        sharedResource++;

        // Release the spinlock
        spinlock.clear(std::memory_order_release);
    }
```

```
}

int main() {
    std::thread thread1(incrementSharedResource, 1);
    std::thread thread2(incrementSharedResource, 2);

    thread1.join();
    thread.join();

    std::cout << "Final sharedResource value: " <<
sharedResource << std::endl;

    return 0;
}
```

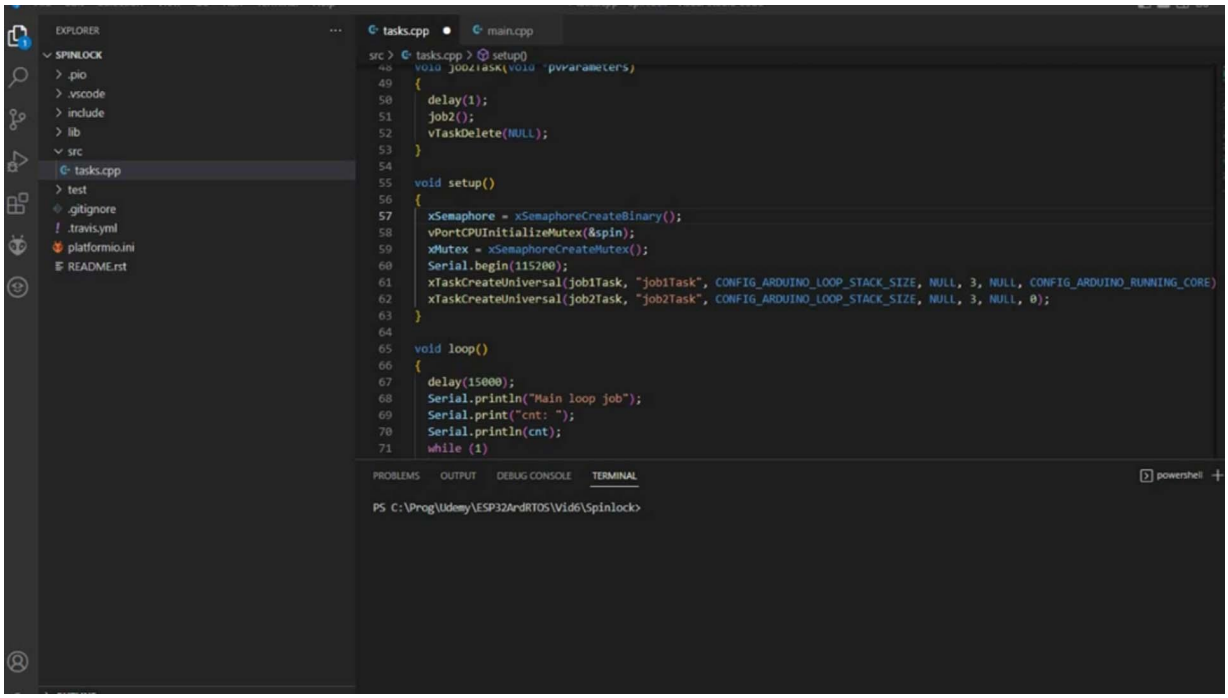
In this example, we use a simple spinlock implemented with std::atomic_flag to protect access to the sharedResource. The spinlock is used to prevent concurrent access to the critical section. Each thread tries to acquire the spinlock with spinlock.test_and_set() (spin-waiting) before entering the critical section and releases it with spinlock.clear() when it's done.

Both threads increment the sharedResource within the critical section, and due to the spinlock, only one thread can execute the critical section at any given time. The iterations variable controls how many times each thread accesses the critical section.

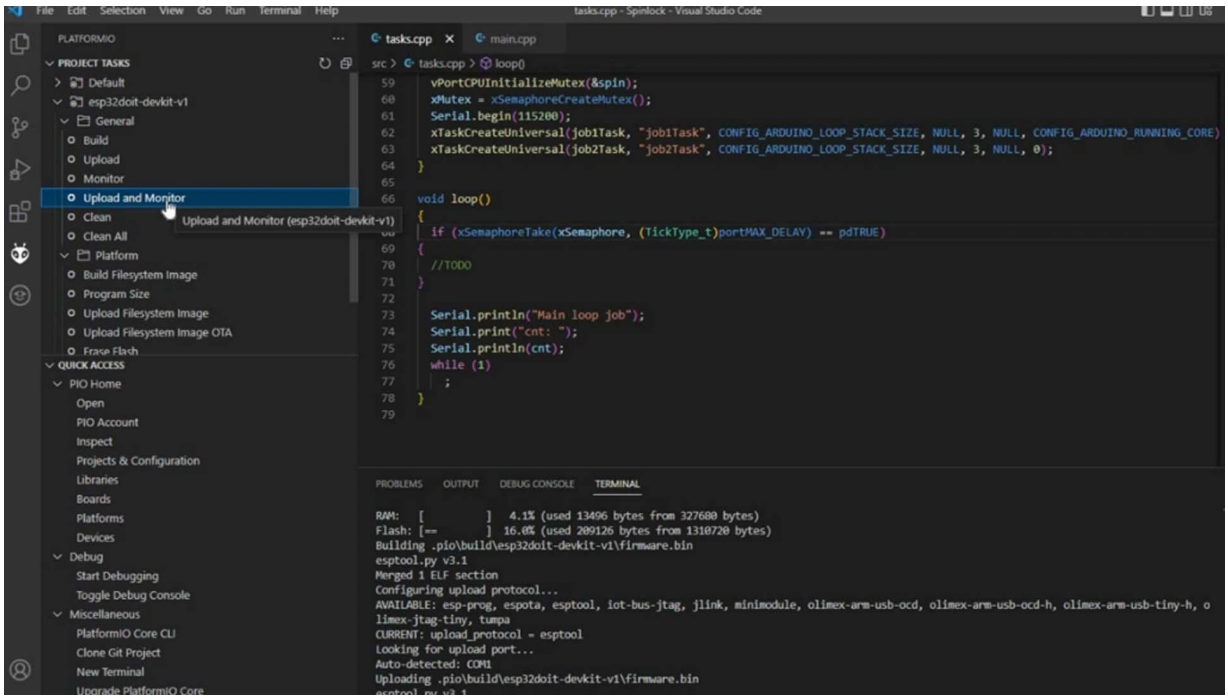
This code demonstrates the use of spinlocks to protect a shared resource while running on multiple CPU cores (multi-core). The program simulates contention between threads, highlighting the importance of synchronization mechanisms like spinlocks when dealing with shared resources in multi-threaded applications.

SEMAPHORES AND QUEUES

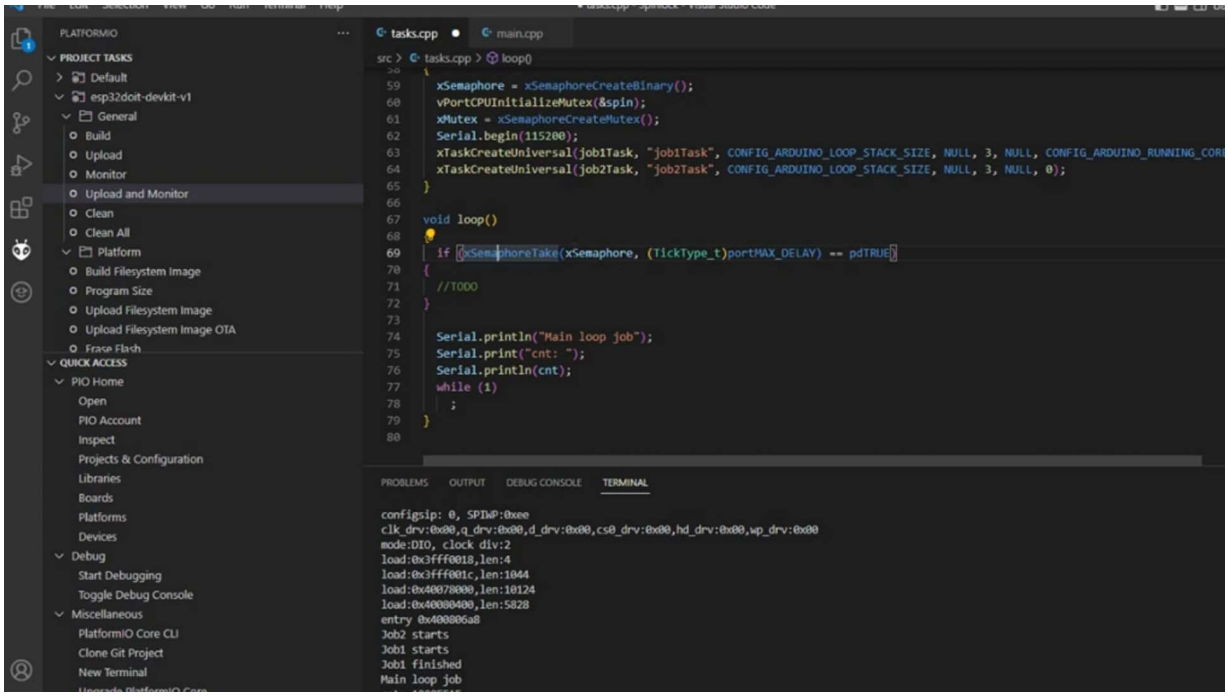
We used a delay of 15 seconds to output the counter value after the end of our tasks. But is there a way to know exactly when our tasks will finish working? Yes, for this we can use a semaphore. A semaphore is similar to a mutex. But unlike a mutex, a semaphore does not need to be given away. Therefore, with the help of a semaphore, one task can signal another about the occurrence of an event. semaphore handle type. This one is semaphore. And then we can create it. Now we can use it. Let's send it at the end of the job1 task.



This timeout in milliseconds. If the timeout has expired, then pd false. If the timeout is not needed, then it can be replaced with the maximum possible value of port max delay. This one. Now we can compile and test. Sorry, ESP is disconnected. Restart again. Restart ESP. Waiting for it to finish its job.

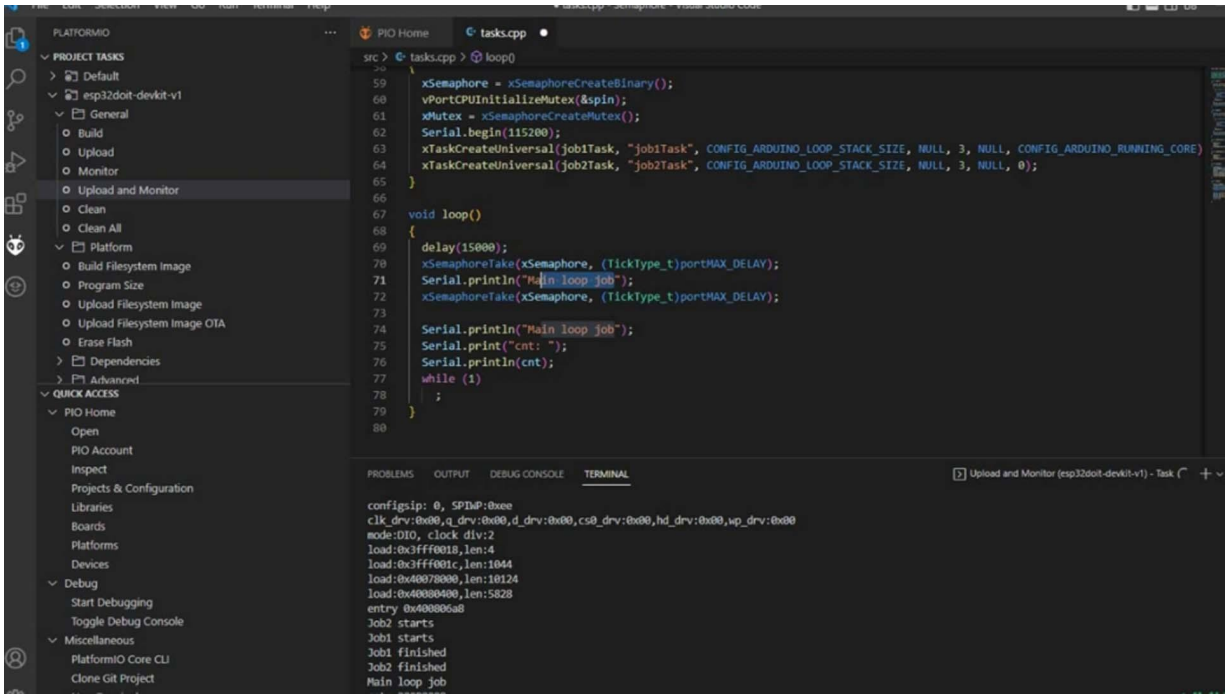


As we can see, at the end of task1, a semaphore was received. At the same time, task2 has not finished its work yet. So the counter value is not yet equal to 20 million. Let's add semaphore sending to the second task. We can simply copy it from here.



Also, we have to accept this semaphore. In our case, there is no need to check the value returned by this function. Therefore, this record can be simplified.

I'll copy and paste this line. Here we will receive both semaphores one by one, regardless of the order in which they arrive.

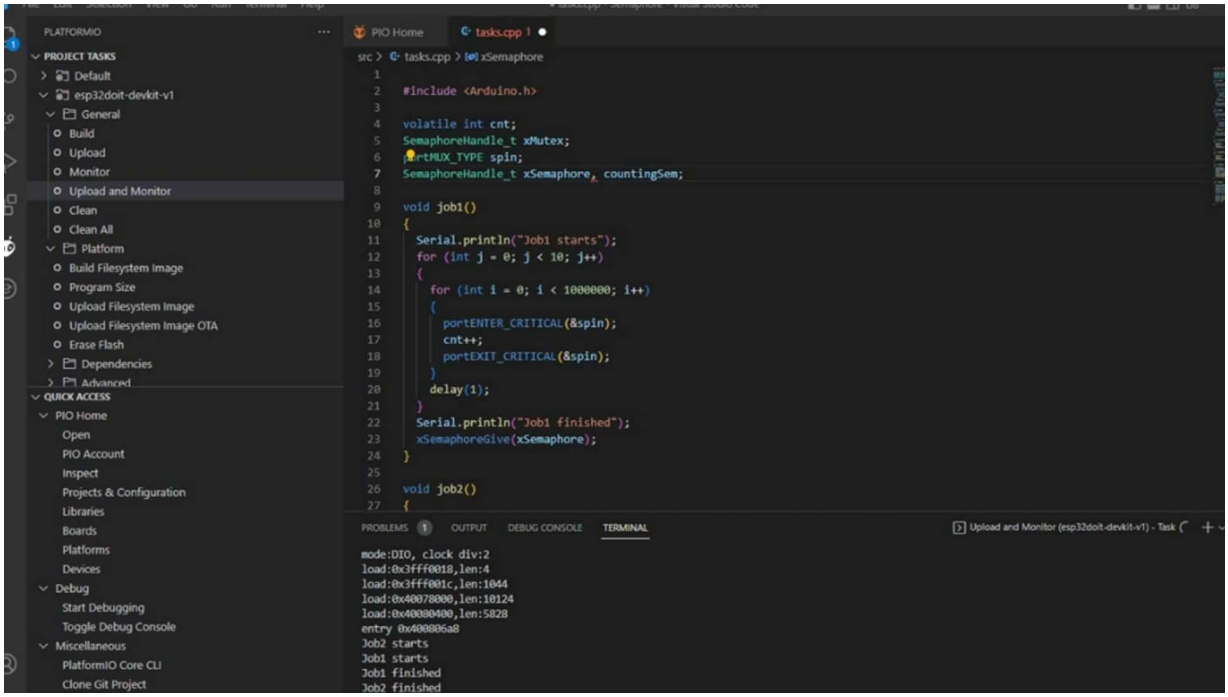


Let's upload and monitor. Job started. As we can see, the counter value is displayed correctly. Thus, the semaphore helped us to synchronize the end of the work of both tasks running on different cores with the Arduino task. Now let's consider the case when our tasks send both semaphores, while the Arduino task has not yet received any. 15 seconds in the Arduino task before receiving semaphores. Here.

And then output messages about receiving semaphores. Here is a delay. And here are messages. Semaphore 1. And semaphore 2. Upload and monitor. Restart. Waiting for my job to be finished. As we can see, the program has received only the first semaphore, and is waiting for the second one. This happened because we used a binary semaphore that can only be set once. But we need to send two semaphores. And we also need to accept two. To do this, we

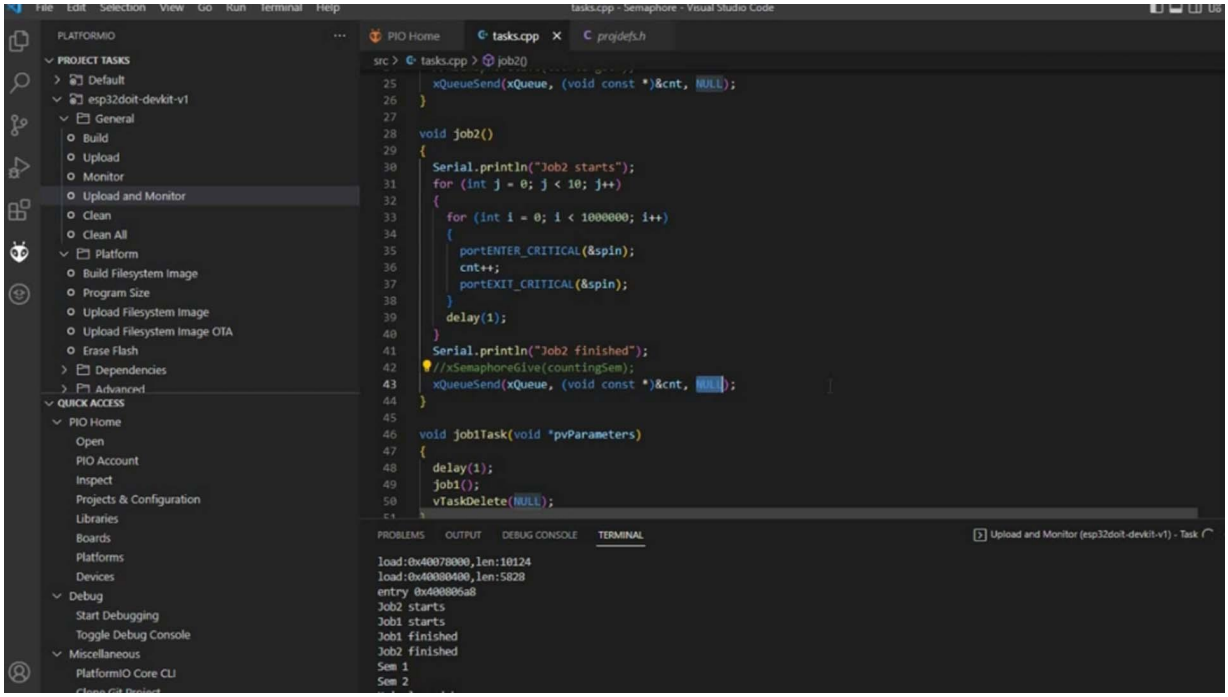
need to use a counting semaphore. Let's change the binary semaphore to a counting one.

A counting semaphore is almost the same as a binary one. It uses the same data type. Let's define it. I'll call it counting sem. It differs only in the function of creating a semaphore.



When creating it, we must specify the maximum number of semaphores and the initial value of its counter. I'll create it here. This is the maximum number of semaphores, and 0 is the initial value of the counter. Now we can simply change our binary semaphore to a counting semaphore. Here, here and sending it. Here and here. Upload and monitor. ESP. Waiting for our semaphores. As we can see, the semaphores with the semaphore we need to transmit data with some kind of work result. For this purpose a queue is used, which allows us to exchange data between tasks. Let's replace our semaphore with a queue. Type of queue is the queue handle. Let's define a queue. I'll call it xQueue. To create a queue we need to specify the type of data being stored in it and the size of the queue. The queue size is the maximum amount of data of the specified type that it can store. The queue is created by the function xQueueCreate.

I specify the queue size of 10 and data type UN32. Now we can transmit and receive data using this queue. Let's pass the counter value at the end of each of our tasks. Next we'll receive it and output to the monitor. The sending will be performed by the function xQueueSend. I'll insert it here and here.



I pass to this function name of the queue, reference to our counter and timeout. We don't need timeout, it's null. Now we can accept the passed counter value by the function xQueueReceive. We don't need this delay and we'll receive it here.

I have created a received counter variable here in which the received value will be written. The queue handler and timeout are also passed to this function. Next we'll output this value to the monitor. And the same for the second job. Also we don't need these entities. Let's upload and monitor. Restart ESP. We see that the counter value is passed from the first task because it finishes its work faster. At the same time the counter value is not yet equal to 20 million. Then we get a message from the second task. And now we see that the calculation is finished.

EXAMPLE DUMMY CODE

Semaphores and queues are often used in concurrent programming to coordinate the execution of multiple threads or processes. Here's an example in Python that demonstrates how to use semaphores and a simple queue:

Python

```
import threading
import time
import queue

# Define a semaphore with an initial value of 2
semaphore = threading.Semaphore(2)

# Define a thread-safe queue
my_queue = queue.Queue()

# Function to simulate a task that consumes an item from the queue
define consumer(thread_id):
    while True:
        item = my_queue.get()
        if item is None:
            break

        with semaphore:
            print(f"Thread {thread_id} is consuming item: {item}")
            time.sleep(1)
```

```
# Function to simulate a task that produces items and adds them to the queue
def producer():
    for item in range(1, 6):
        my_queue.put(item)
        print(f"Produced item: {item}")
        time.sleep(0.5)

# Create consumer threads
consumers = []
for i in range(3):
    t = threading.Thread(target=consumer, args=(i,))
    consumers.append(t)
    t.start()

# Create a producer thread
producer_thread = threading.Thread(target=producer)

# Start the producer
producer_thread.start()

# Wait for the producer to finish
producer_thread.join()

# Signal consumers to exit by adding None to the queue
for _ in range(3):
    my_queue.put(None)

# Wait for consumer threads to finish
for t in consumers:
    t.join()

print("All threads have finished.")
```

In this example, we have three consumer threads and one producer thread. The producer produces items and adds them to the queue, while the consumers consume items

from the queue. We use a semaphore with an initial value of 2 to limit the number of consumers that can access the queue concurrently. The producer produces items at a faster rate than consumers consume them, which demonstrates how semaphores can control access to a shared resource.

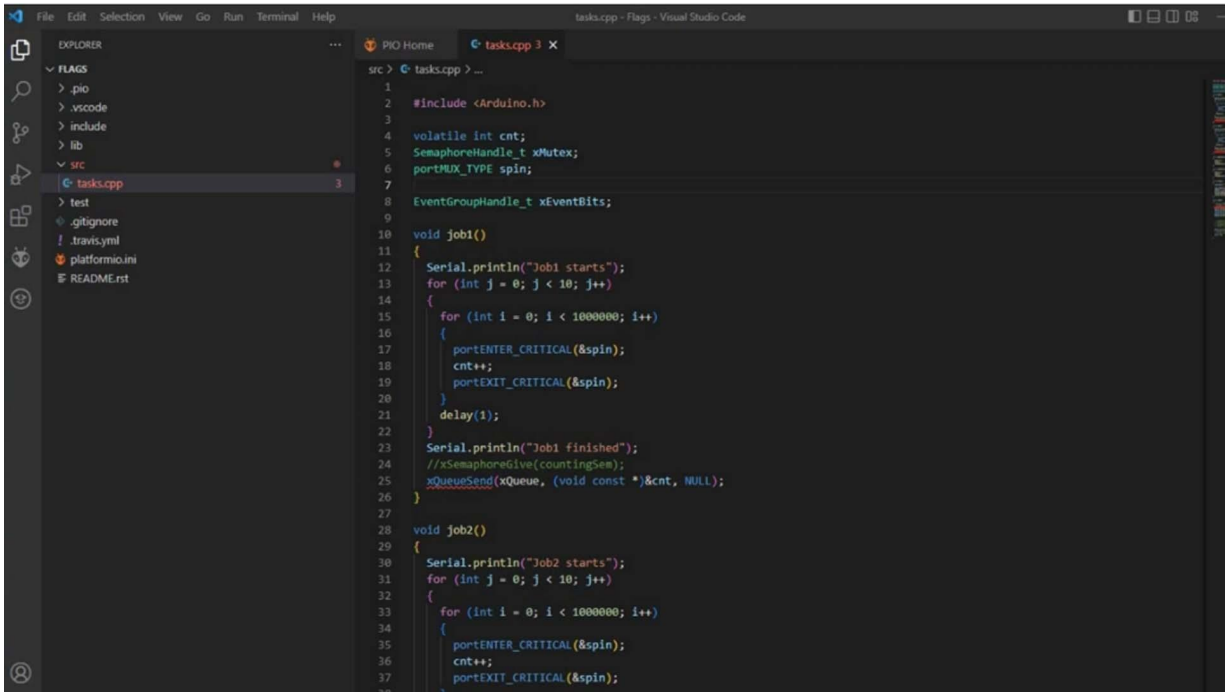
The queue (my_queue) is thread-safe and ensures that items are added and removed in a synchronized manner.

The program also uses None as a signal to tell the consumer threads to exit once all items have been consumed. This is a common pattern for gracefully terminating consumer threads in a producer-consumer scenario.

This code showcases the use of semaphores and queues for coordinating multiple threads in a concurrent program.

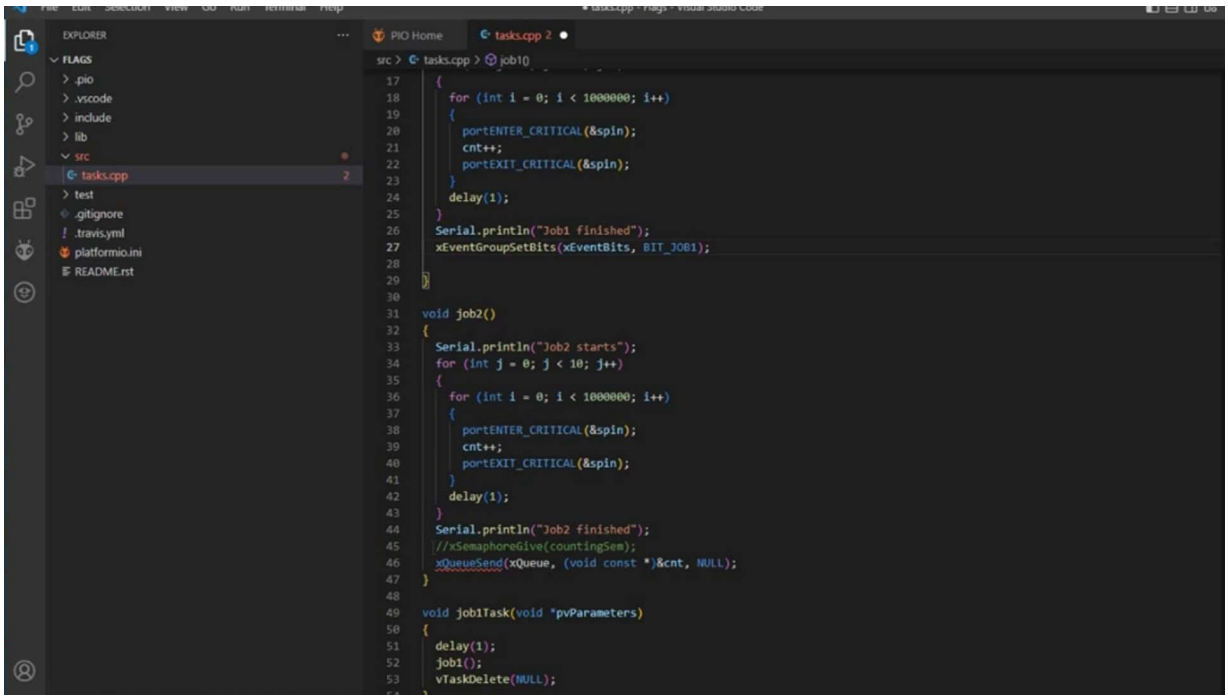
EVENT FLAGS

Earlier we looked at synchronization between three RTOS tasks using semaphores and queues. There is another possibility of synchronization using flags. The flag is a single bit that is set by the program when an event occurs. Then this flag can be checked elsewhere in the program. The flags are combined into a group called event group. In ESP32 there can be 24 flags in such a group. Using flags is similar to using semaphores or queues. First we need to define a handle for a group of flags. This event group handle.

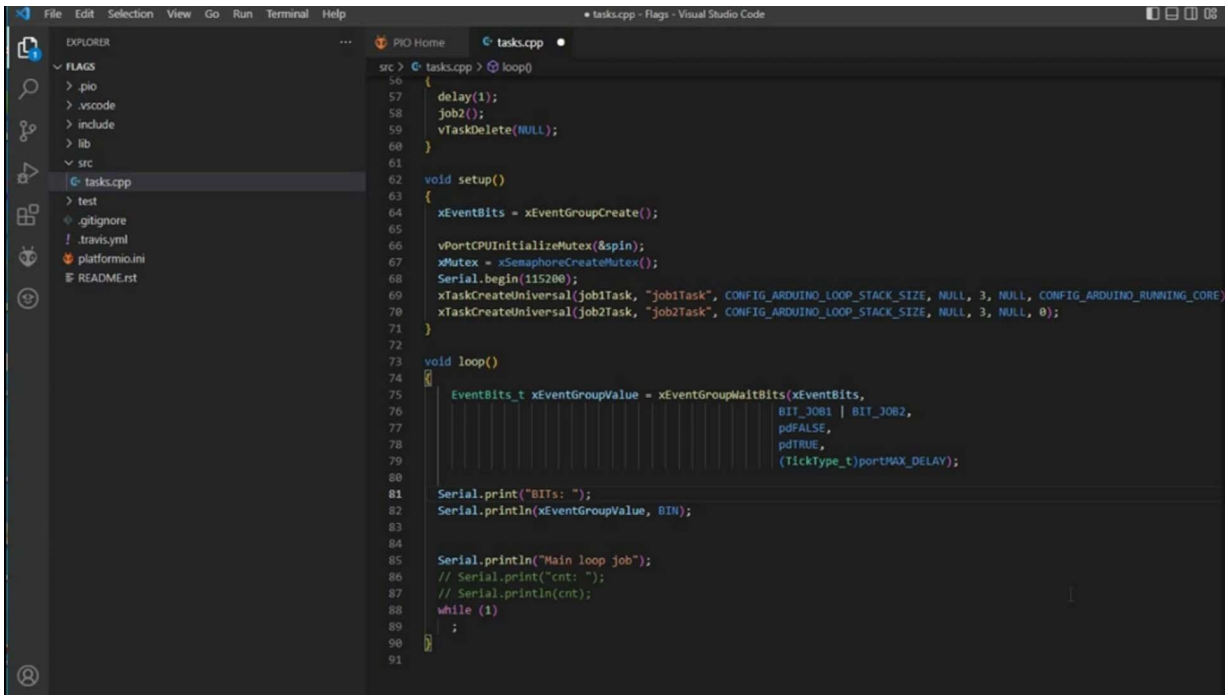


Speaker 1 (00:00:57) - Then we can create this group in setup. We don't need semaphores. I removed it. And here I remove semaphores and queues. We also need to define the bits themselves. Each task will have its own bit. This bit for job 1 and for job 2. We can assign them any numbers. Let it be 5. It can be any. Within 24. Now we can set these bits instead of semaphores or queues. Here. It's a bit for job 1 and the same for job 2. Now we This

is done by the function xEventGroupWaitBits. We don't need this all. I replaced it with the function. bit weighting function.



If this parameter is passed equal to pdTrue, the received bit will be reset upon exiting the function. And if this is true, then the function waits for all specified bits to be set. The bits themselves are set here. These bits. Also they handle groups of bits and a timeout is passed to the function. And timeout. Thus, in our case, the function will not reset the received bits and will wait for all bits to be set.



Let's compile. First I'll uncomment this. Now compile and monitor. This comfort is busy. Compile again. It's loaded with ResetESP.

And waiting for our bits. As we can see in the terminal, the program worked correctly and both specified bits were set. These bits.

EXAMPLE DUMMY CODE

Event flags are synchronization mechanisms used to signal between different threads or processes when a particular event or condition occurs. Here's an example in Python that demonstrates how to use event flags using the threading module:

Python

```
import threading

# Create an event flag
event = threading.Event()

# Function that waits for the event
def wait_for_event():
    print("Thread A is waiting for the event to be set.")
    event.wait()  # Blocks until the event is set
    print("Thread A: Event is set, continuing.")

# Function that sets the event
def set_event():
    print("Thread B is setting the event.")
    event.set()

# Create two threads
thread_a = threading.Thread(target=wait_for_event)
thread_b = threading.Thread(target=set_event)

# Start both threads
thread_a.start()
thread_b.start()
```

```
# Wait for both threads to finish
thread_a.join()
thread_b.join()

print("Main thread: All threads have finished.")
```

In this example, we create an event flag using threading.Event(). Thread A is waiting for the event to be set using event.wait(), which will block until the event is set by Thread B using event.set().

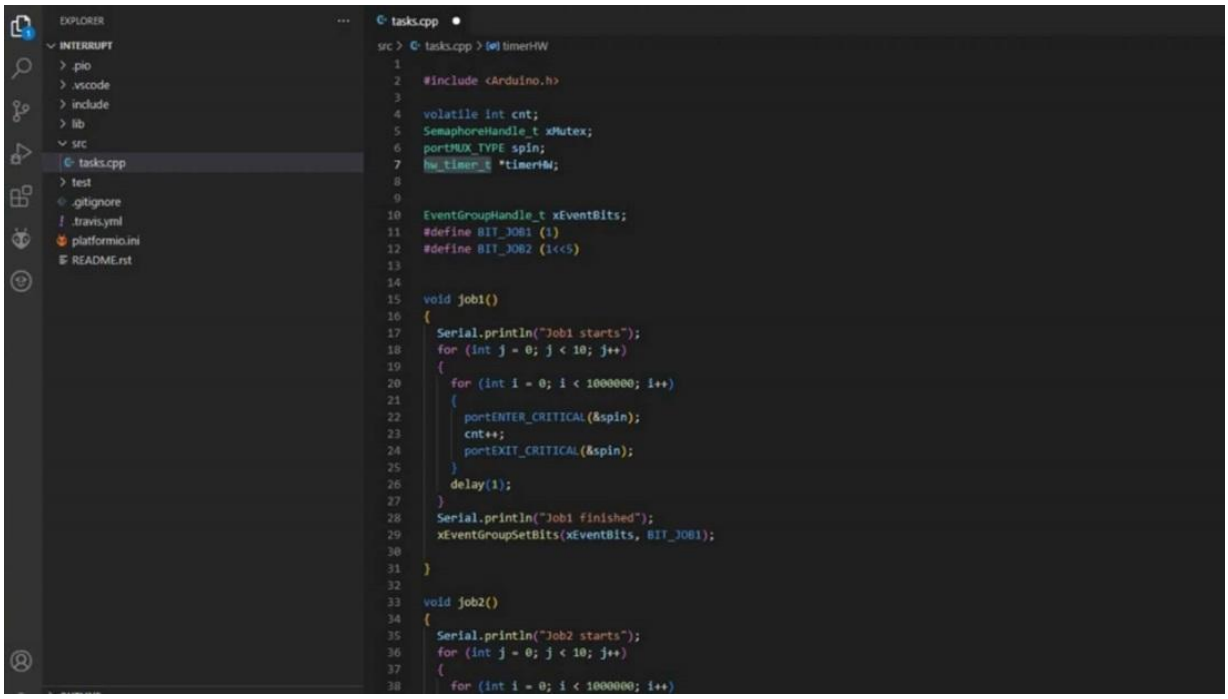
Here's the expected execution flow:

Thread A starts and prints that it is waiting for the event to be set.
Thread B starts and sets the event flag.
Thread A, once the event is set, continues executing and prints that the event is set.
Both threads finish their work.
The main thread prints that all threads have finished.
Event flags are often used to coordinate activities between multiple threads or processes, allowing them to signal each other when specific conditions are met.

HARDWARE INTERRUPTS

Let's now look at hardware interrupts. As you know, interrupts allow you to suspend the main program in order to perform a short function of the interrupt source handler. Also, interrupts can interrupt each other, depending on their priority. Interrupts with high priority can interrupt them with low priority. The FreeRTOS task priority and interrupt priority are two different systems. Any hardware interrupt with low priority will always interrupt the task with high priority.

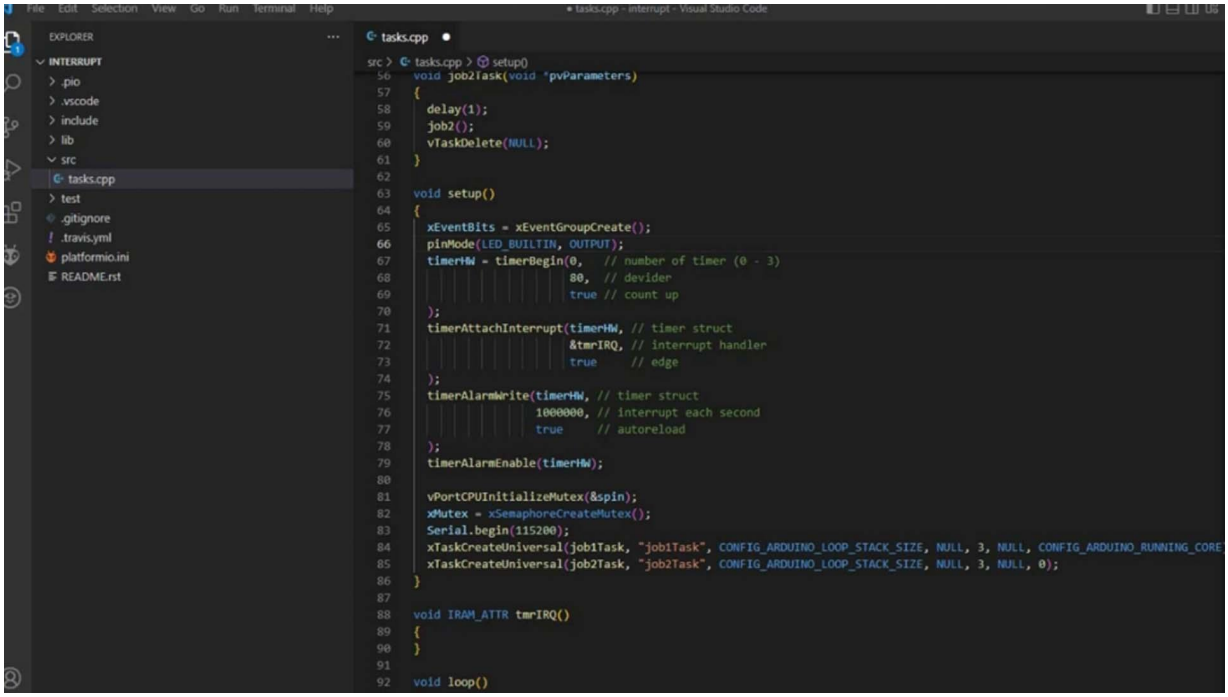
Unfortunately, the Arduino functions do not allow setting the priority of interrupts. Therefore, we will not change it. We will use the default one.



Each ESP32 core has its own interrupt system. Now we will look at the operation of interrupts using the example of a timer. Let's add an interrupt from the timer to our program. Let the interrupt happen every second and the LED will toggle in the interrupt. To define a

timer, use the HardwareTimer type. Let's define an interrupt function for the timer. Now it's doing nothing.

And I'll add a prototype for it. And then we can create and start the timer. Here, with these functions. The first function initializes the timer. It sets the timer number, 0, divider and the direction of the count, up or down. Then the interrupt function that we created earlier is attached to the timer.

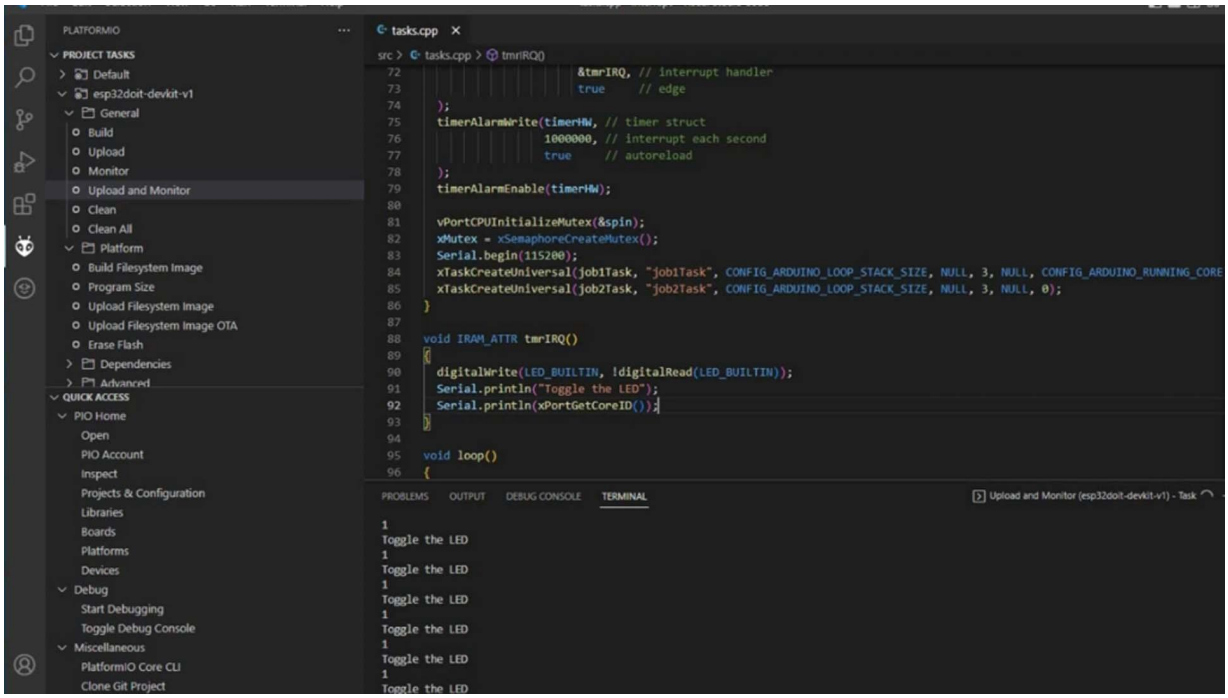


Next, the timer is set to the value to which it will count. This one. It's selected so that an interruption occurs every second. And then the timer starts with the function TimerAlarmEnable. Now let's add an LED to the interrupt. Initialize it first.

This initialization of the LED. Now we insert the toggling of the LED in the interrupt function. I'll also add an output to the monitor, a message about switching the LED.

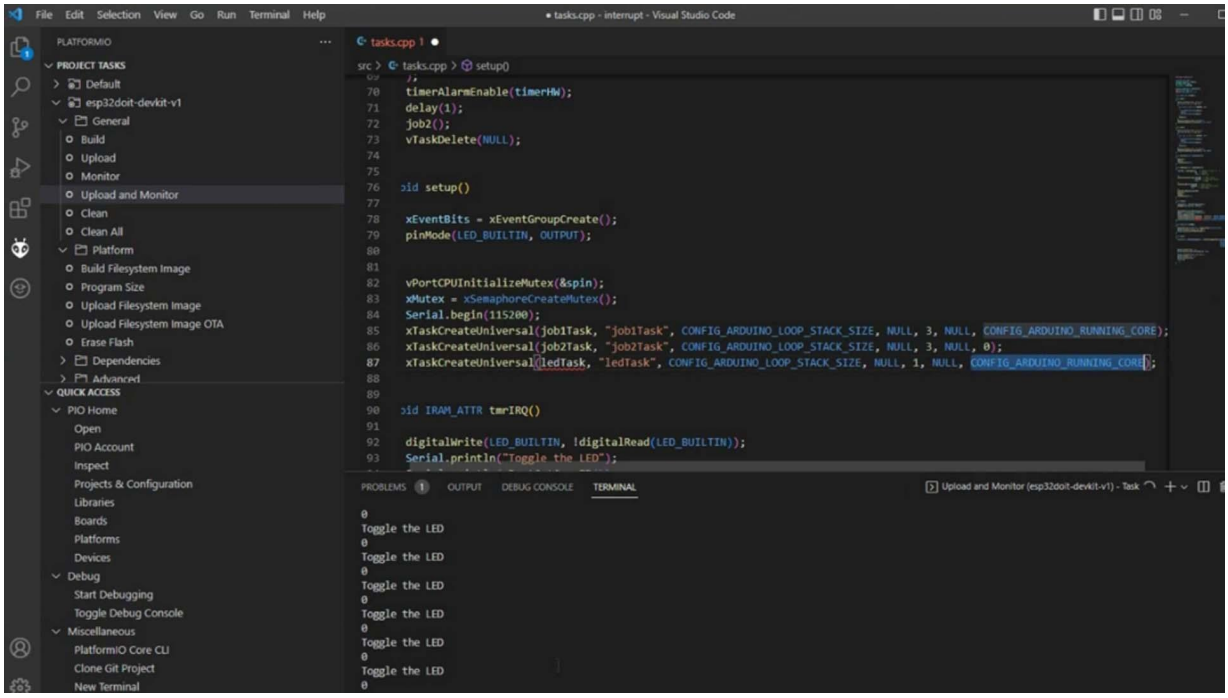
Now we can compile and test how it works. Upload and monitor. toggling. On which core does the interrupt processing function work? We can find out by using the function exportGetCoreId. It returns the number of the core it's running on. I'll output this number to the monitor. This exportGetCoreId. Upload and monitor. Restart ESP.

We see that the number of the core is 1. At the same time, the setup task in which the timer was initialized runs on the same core. Thus, the interrupt function will be run on the core on which the interrupt was initialized.

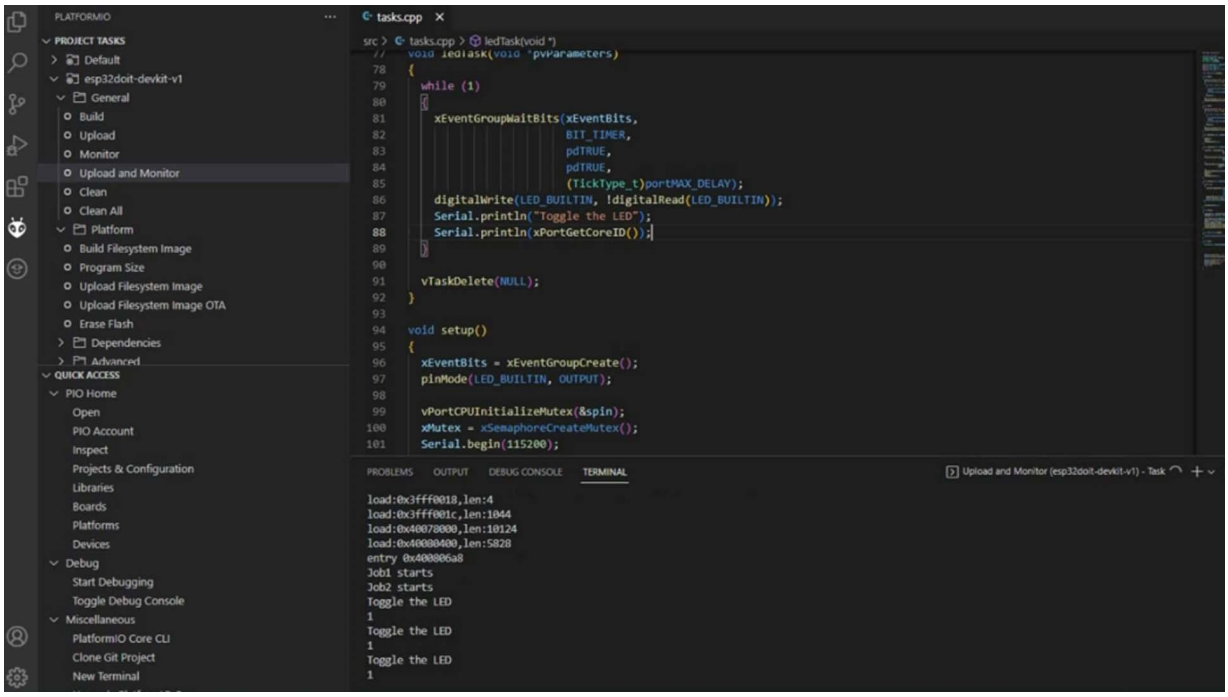


Let's check it out and start the timer on another core. This can be done at the beginning of task 2, which runs on core 0. remove these functions to the beginning of task 2. Here. Upload and monitor. Restart ESP. As we can see, our interrupt works now on core 0. Our interrupt handler has a print output function to the monitor. This function runs for a relatively long time. So, it's better not to use it in interruptions. So, let's create another task and transfer the functionality from the interrupt handler to this task.

And in the handler itself, we will only set a flag that will be received in the task. I'll call it letTask and run it on core 1, which Arduino runs. And adding the task. It's an empty letTask. Define a flag for the timer. Let it be number 2.



Then we can set this flag in our interrupt function here. Now this flag can be accepted in the late task here. Note that this parameter is set to pdTrue. So the flag will be reset when exiting this function. Now we can transfer functions from the interrupt handler to the late task. These are all functions.



And now we can upload and monitor. As we can see the late task is running on core 1. Also all our tasks continue to be performed without interfering with each other. Let's restart ESP32 and take a look at their work. Restarted.

Thus we have created a program running simultaneously on two processor cores and performing several tasks. Now let's look at the flag setting function that we call it from the interrupt. This one. We can see that it's different from the one we call in the task. I will copy and place these two functions side by side so that the difference is visible. Firstly, the function called from the interrupt has the same name. Only from ISR this has been added to it.

It should be borne in mind that all similar FreeRTOS functions have the phrase fromISR in their name. Therefore, only fromISR functions should be called in the interrupt. Secondly, fromISR function has another parameter. This. It's designed to quickly switch to the task with the highest priority at the end of the interrupt. This parameter is optional. I recommend not using it and passing null to functions instead. This one, null. I'll comment on this. Don't use it. Mark here.

INTRODUCTION

Hello and welcome to this new series, where we will learn how to send SMS without the use of a GSM module. We will do this using the ESP32 board and the online simulation platform known as Wokebeam. With this, it is easier to send SMS. We do not have to go through the process of getting a GSM module. We don't have to buy a SIM card and then go to the extent of recharging the SIM card. With just our ESP32, we will be able to do this with a software known as Twilo.

ESP32 is a low-powered, low-cost microcontroller which has both WiFi and Bluetooth built-in. It houses the ESP32 chip. It has a similar board... ESP8266 board. They are all suitable for IoT-based projects. As shown in this picture, we will be using the ESP for this project. This is the Twilo software which we will be using to send the messages. We will be using Twilio to achieve this. With just Twilo and the ESP board, we will be able to send messages by using just one board.

Speaker 1 (00:01:39) - With this, we will achieve a lot of stuff like sending and receiving SMS to control teams or just for communication. Twilo also saves costs. I said this earlier in the introduction. When using Twilio, we do not have to buy a GSM module. The GSM module is ruled out in this case. We don't have to buy a SIM card and we don't have to spend money recharging the SIM card to enable us to... for making and receiving phone calls. They also provide the tool for sending and receiving text messages and performing other communication functions.

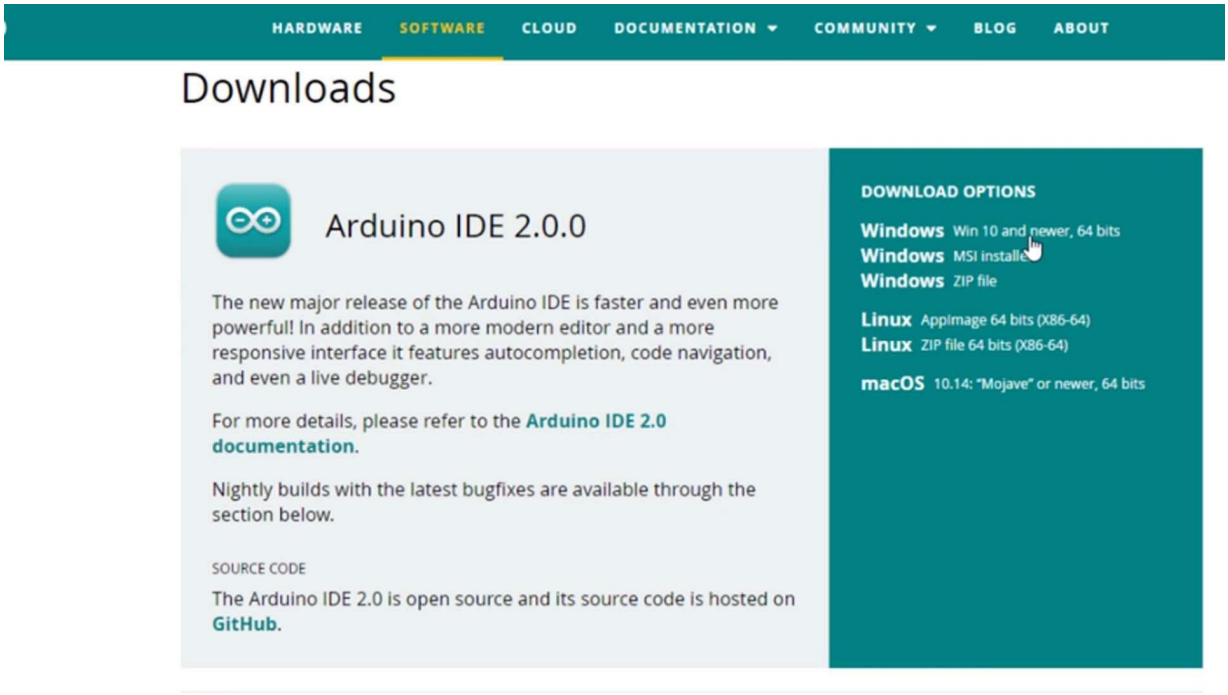
Using its web service APIs. Throughout this tutorial, we will use their programmable messaging services. Before we go further, let's go over to the Twilo website and see what it looks like. In my browser, I'm going to search for twilo.com. That is their website. This is Twilio's website. We will be setting up later. I will show you how to do that. For now, let's just go to the website and see what it looks like. You can go to the website. You can see the services they offer. Book, SMS, You can use it to improve your contact center.

WhatsApp messages and all that. This is the Twilo website. If you haven't seen the Twilo website, which

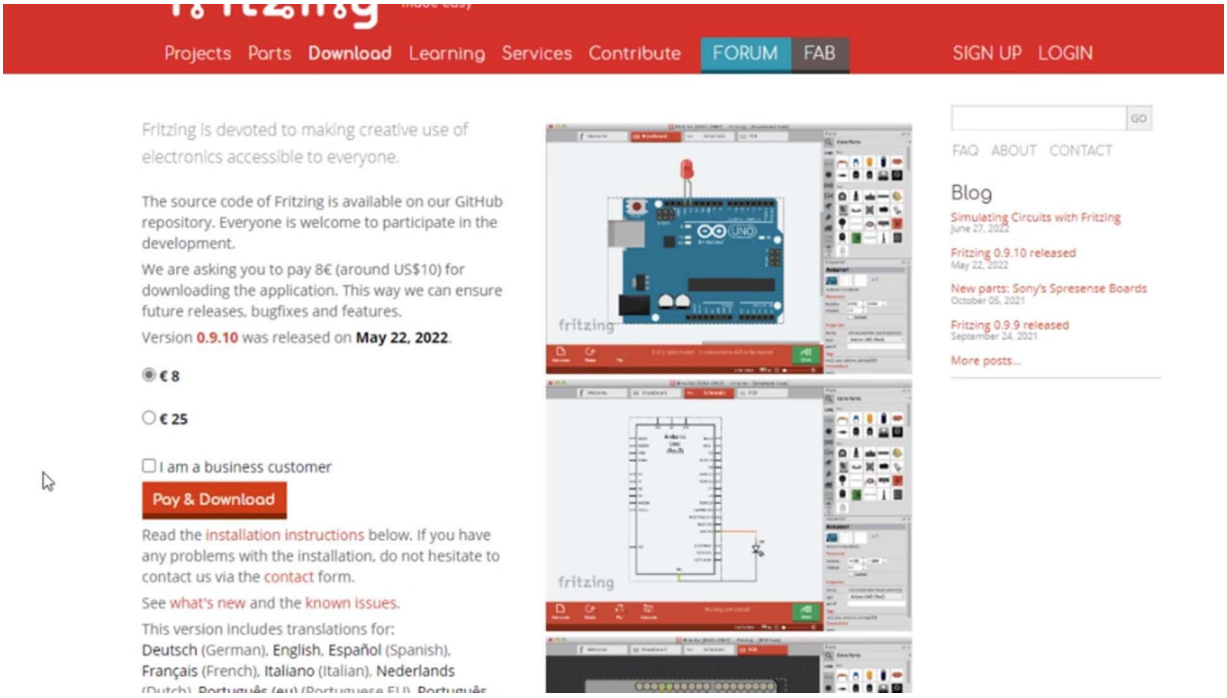
GETTING STARTED

So for the software requirements for this project, we'll be using the Arduino IDE with the ESP32 board manager. We'll need internet connectivity and we'll be using Wokui. For those of us that will be using ESP32 board, the physical board there with

our Arduino IDE. For those that will be using the ESP32 install the Arduino IDE, for those of us that will be using the ESP32 hardware, we'll go to our browser.

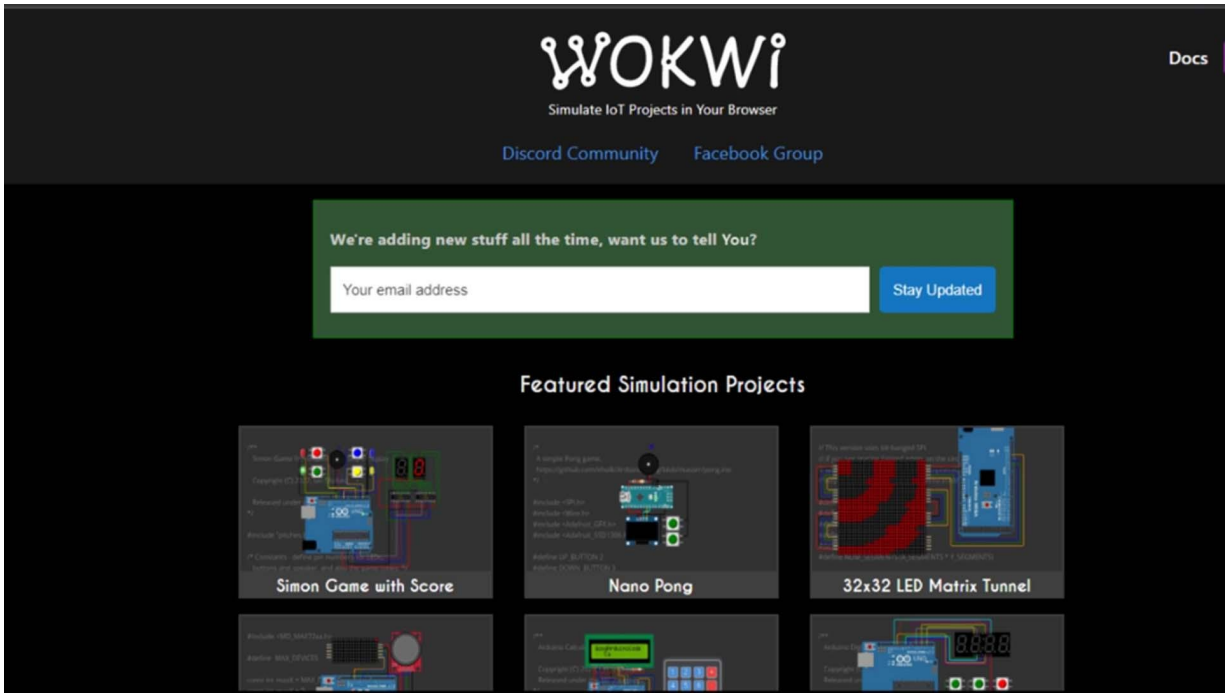


We'll search for an Arduino IDE download. So we'll click on the first link. So look for Windows installation. Windows 10 and newer 64 bits. So you can look for the one that is suitable for your PC. I'll just click on download. Next, Or you can, yeah, just close that. The download will begin in a moment. So while it is downloading. Another software which we won't use in this series, but it's helpful.



For those that will be using the physical ESP setup. You will need the Freezin. Freezin is used for drawing circuit diagrams. So Freezin download. So we open the first link. So Freezin, some websites will require you to pay before you're able to download it. So this website actually has the paid version for it. So we'll go back. We'll go back and look for another website that has the free version. Or the trial version. Okay, good. So we can download from this website. So just download. is quite simple. Open the... and get started. So now in the Arduino IDE We'll have to install the ESP board manager. So go to tools first of all go to preferences So once you connect your ESP board Everything will be set for you as once you plug in your ESP board into your USB port This link is gonna change to ESP and some details So that we'll go to tools Tools, we are looking for board info Keyboard manager I think that should be board manager Okay So search for ESP ESP 32 So this is it We'll install it

How to install the ESP 32 board manager including the Arduino IDE? Let's go ahead and test our... with our computer PC because Wokui is a software that we'll be using to simulate... Let's head on to Wokui.com But this is what the website looks like We'll click on sign in or sign up if you're a first-timer you sign up So I'm gonna sign up with So I'm in



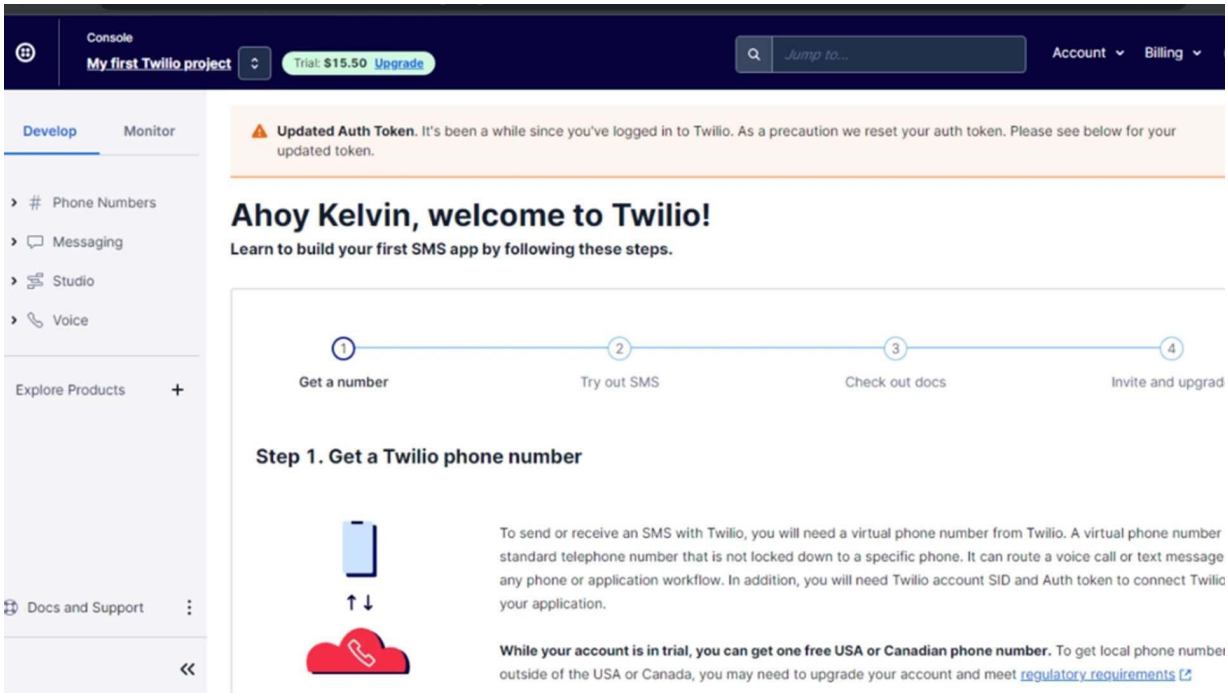
I'm in so you can see and project templates over here Project templates, you can just pick any one you want and edit to your taste So I'll just pick ESP 32 Arduino the one with an LED so we can see on the left hand side we have the Code side while on the right hand side we have the simulation part so Let's simulate this and see how it works. This is just to blink an LED. So it's blinking So now we can change the color of the LED from diagram.json We can change it from

0.5 seconds to 1000 1000 microseconds that is a second Sorry 1000 milliseconds Or one second rather So let's simulate again So this is what we that is what we'll... this series.

TWILIO SET UP

So we have tested our ESP online simulation with the WOPI software So we will go... Twilio... to We will go to the official website twilio.com So this is the website We'll click on sign up and start building. So when you click on sign up, you are going to fill out all these informations here I Start a free trial So I already have an account which I'm gonna log in with So I'll put my password

So verify your identity So after signing up on your new account and when you're logging in for the first time you have to register a phone number Which you will be using. So I'll put in a phone number and verify So a code will be sent to my phone. This is the code that has been sent to my phone So now I'm going to put in their code here 8 9 We will answer a basic question here. We are using the SMS products Alert and notification. We don't need code So get started with Twilio So skip this, we don't need all this Done.



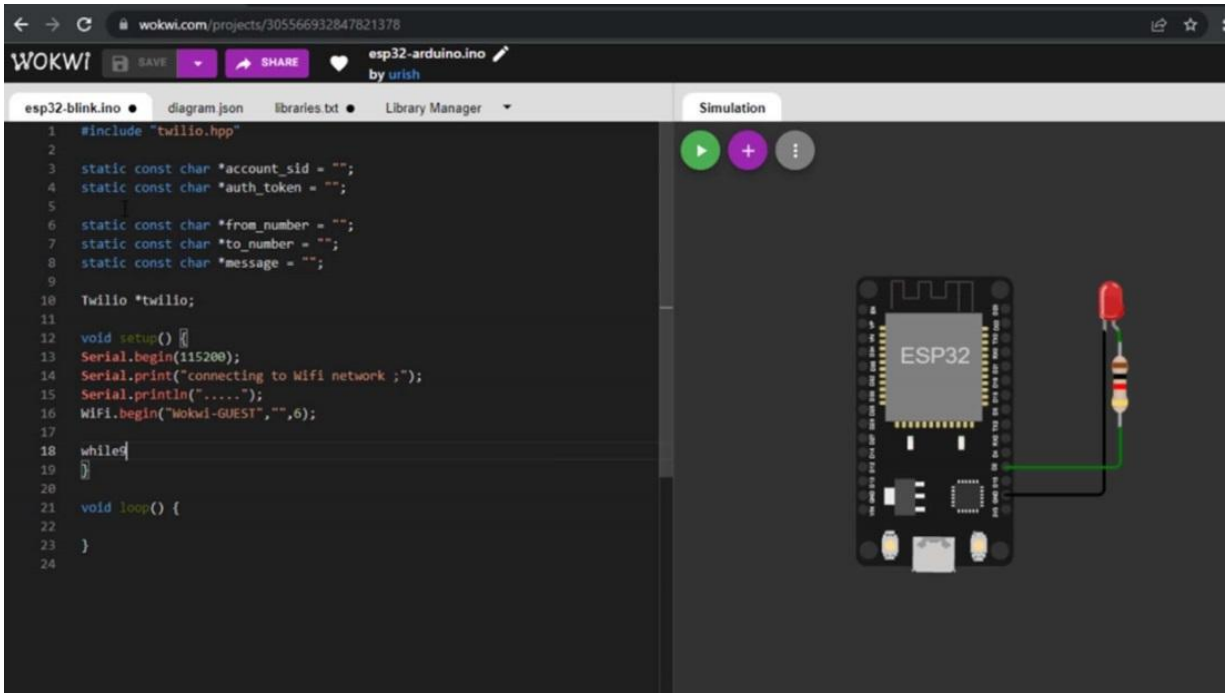
So when we enter our Twilio account First we'll go to phone numbers so that we can purchase a number and manage Buy a number. So we'll search for the number that has SMS Compactivity. So this one has SMS. We can use it to send SMS. So click on buy Click on buy Scroll down to where we have the buy button. We'll buy this So we've bought this now we can send SMS using this number that we've bought So let's go back Let's go to messaging Get set up Start set up message service name we'll just give it We'll just look for a simple name and give it.

We'll just give it ESP alerts ESP... this is a number. We'll add this number, the number we purchased so then this is our... my notepad Paste it in SID. Next I'll view my authentication token. I'll copy, open notepad And paste So try SMS After that you get to this point where you're trying to send SMS. So body of text can be anything can be hello from Hello from my ESP Testing Hello from my ESP test. So let's send the SMS. So the SMS has been sent to our phone So let me get a screenshot from my phone So that's the SMS and See how it comes in in the phone See how

CODE SETUP FOR ESP32 USING TWILIO

So now having set up our Twilio account Let's head over to the wokwi... code for ESP using Twilio. We will be writing a sample code For ESP32 to send SMS using Twilio on the wokwi software So let's get rid of the code in the setup and the loop... Twilo ESP client library Then we come back and include that library Twilo.hpp So next we declare our account SID and the authentication token that we have in our Dashboard, so from number, that is the number we purchased on the

Twilo software Then static constant char, the number we are sending to which is our number just as you registered on the Twilo software Then... the... for now.



Come back to it later than Twilio, Twilo So in the setup we initialize our Twilio monitor first Then we set up a connection for the ESP32 to connect to the internet So on wokwi simulation, it has in-built internet

connection for the ESP32 so Wi-Fi does begin It needs no password SSID is wokwi, guest is in capital letter Then 6, with this the ESP32 will connect to the in-built internet connection of the wokwi software So why Wi-Fi, why the ESP is not connected to Wi-Fi? It will keep printing connecting delay 0.

5 seconds Then when... in line 24, we initialize the connection between the wokwi or the ESP32 and the Twilo. So the delay is a second. Then next is the message function that will send the message to our phone So if the message was sent successfully, it will print Send message successfully else serial.print will respond So that is it for the sample code.

Speaker 1 (00:04:03) - So we get rid of the... LED and the resistor, we don't need that so our account's SID will copy it from where we saved it We'll paste We'll copy the authentication token And paste it there as well Then the number we purchased on our Twilio account We'll put it and the number we are sending to Then the body of the message We'll just use message using Twilio then in brackets testing.

EXAMPLE DUMMY CODE

To set up an ESP32 using Twilio, you would typically use the ESP32's capabilities to connect to the internet (Wi-Fi) and send HTTP requests to Twilio's API. Below is a simplified example using the Arduino framework for ESP32.

Before starting, make sure you have the following information:

Twilio Account: Create an account on Twilio.
Twilio Phone Number: Obtain a Twilio phone number.
Account SID and Auth Token: Obtain your Twilio account SID and Auth Token from the Twilio console.
Here's an example Arduino code using the Arduino IDE for ESP32:

Cpp

```
#include <WiFi.h>
#include <HTTPClient.h>

const char *ssid = "your_wifi_ssid";
const char *password = "your_wifi_password";

// Twilio Account SID and Auth Token
const char *accountSid = "your_twilio_account_sid";
const char *authToken = "your_twilio_auth_token";

// Twilio phone number and recipient phone number
const char *twilioPhoneNumber =
"your_twilio_phone_number";
```

```
const char *recipientPhoneNumber =
"recipient_phone_number";

void setup() {
  Serial.begin(115200);
  delay(1000);

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
}

void loop() {
  // Replace this message with the content you want to
send
  String message = "Hello from your ESP32!";

  // Build the Twilio API URL
  String url = "https://api.twilio.com/2010-04-
01/Accounts/" + String(accountSid) + "/Messages.json";

  // Set up the HTTP client
  HTTPClient http;
  http.begin(url);

  // Set up Twilio authentication
  String credentials = String(accountSid) + ":" +
String(authToken);
  String encodedCredentials =
base64::encode(credentials);
  http.addHeader("Authorization", "Basic " +
encodedCredentials);
```

```
  // Set up the message parameters
  http.addHeader("Content-Type", "application/x-www-
form-urlencoded");
  String postData = "To=" +
String(recipientPhoneNumber) + "&From=" +
String(twilioPhoneNumber) + "&Body=" + message;

  // Send the POST request to Twilio
  int httpResponseCode = http.POST(postData);

  // Check for success
  if (httpResponseCode > 0) {
    Serial.print("Twilio API response code: ");
    Serial.println(httpResponseCode);
    String payload = http.getString();
    Serial.println("Twilio API response: " + payload);
  } else {
    Serial.print("Twilio API request failed. HTTP response
code: ");
    Serial.println(httpResponseCode);
  }

  // Clean up
  http.end();

  // Wait for a while before sending the next message
  delay(5000);
}
```

Please replace the placeholder values with your actual Wi-Fi credentials, Twilio account SID, auth token, Twilio phone number, and recipient phone number.

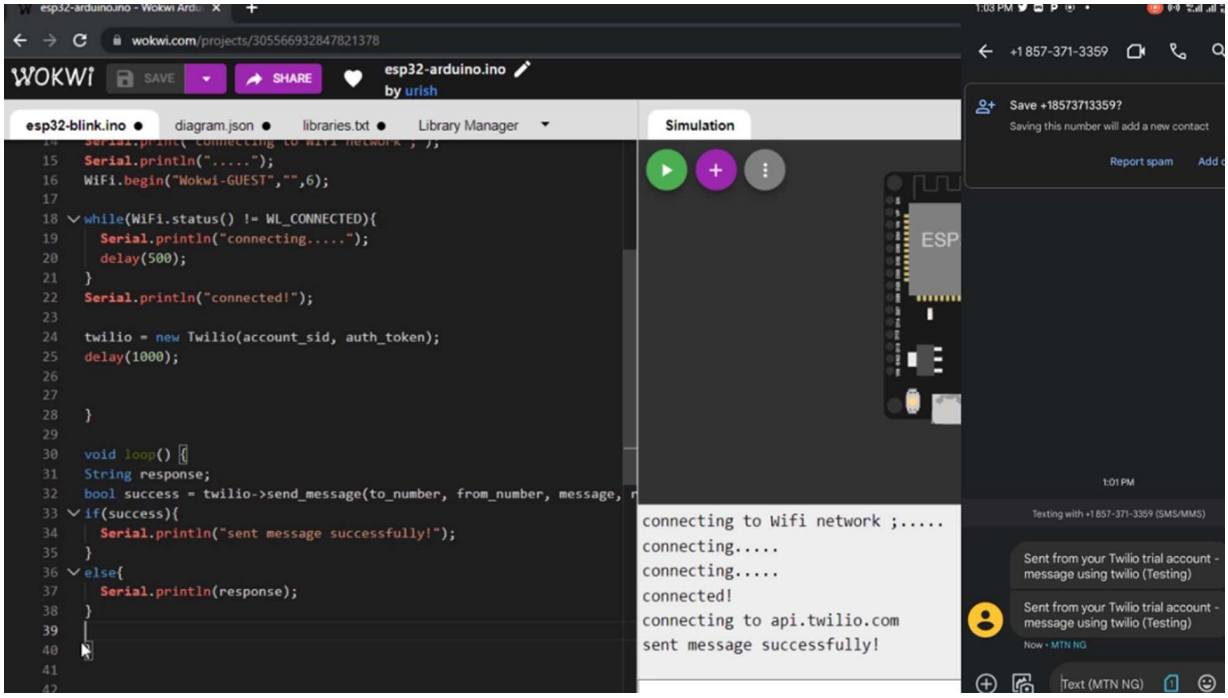
This example sends a simple SMS message to the specified recipient phone number using Twilio's API. Note

that Twilio's API requires HTTPS, so make sure the ESP32 supports HTTPS.

Also, keep in mind that handling sensitive information like Twilio credentials on an embedded device requires careful consideration of security. In a production environment, it's essential to implement secure practices for handling credentials and ensure the overall security of your IoT application.

TESTING CODE SETUP FOR ESP32 USING TWILIO

In this episode, we'll be testing the sample code we wrote in episode 4 So that's my mobile phone by the side. Let's simulate it So that it's connected. API of Twilio to send us... you can see Twilio again and we've received SMS and you can see the feedback on the serial monitor to send a message successfully so now let's Move this message.



Let's move it out from setup and move it to loop so that We don't have to restart the simulation to keep receiving the messages So we just copy it We just copy it from the setup Come down to the loop and paste it in the loop so same thing for the success condition if it is a success it should give us feedback and The serial monitor Then I'm gonna add a general delay so that it doesn't keep sending immediately. So a delay of 3 seconds Let's simulate So connecting to Twilio So we've received the message.

So now we are not going to refresh the simulation and the messages will keep coming in Because now we put it in the loop so that is it for... ESP32 with Twilio software.

EXAMPLE DUMMY CODE

Testing an ESP32 code setup that uses Twilio involves verifying that the ESP32 can connect to Wi-Fi, send HTTP requests to Twilio's API, and receive the expected responses. Below is an example code to test the setup for sending a message using Twilio. This code can be run on your ESP32 to test the Twilio integration.

For testing, you may want to use a development board with serial communication capabilities, such as the ESP32. You can view the serial monitor output to check the status of your tests.

Please ensure you have already set up the Twilio and Wi-Fi configurations in your main code, as mentioned in the previous response.

Cpp

```
#include <Arduino.h>
#include <WiFi.h>
#include <HTTPClient.h>

const char* ssid = "your_wifi_ssid";
const char* password = "your_wifi_password";
const char* accountSid = "your_twilio_account_sid";
const char* authToken = "your_twilio_auth_token";
const char* twilioPhoneNumber =
"your_twilio_phone_number";
const char* recipientPhoneNumber =
"recipient_phone_number";
```

```
void setup() {
  Serial.begin(115200);
  delay(1000);

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
   delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");

  testTwilioIntegration();
}

void loop() {
  // Not used in this test
}

void testTwilioIntegration() {
  String message = "Testing Twilio integration from
ESP32";

  // Build the Twilio API URL
  String url = "https://api.twilio.com/2010-04-
01/Accounts/" + String(accountSid) + "/Messages.json";

  // Set up the HTTP client
  HTTPClient http;
  http.begin(url);

  // Set up Twilio authentication
  String credentials = String(accountSid) + ":" +
String(authToken);
  String encodedCredentials =
base64::encode(credentials);
```

```
  http.addHeader("Authorization", "Basic " +
encodedCredentials);

  // Set up the message parameters
  http.addHeader("Content-Type", "application/x-www-
form-urlencoded");
  String postData = "To=" +
String(recipientPhoneNumber) + "&From=" +
String(twilioPhoneNumber) + "&Body=" + message;

  // Send the POST request to Twilio
  int httpResponseCode = http.POST(postData);

  // Check for success
  if (httpResponseCode > 0) {
    Serial.print("Twilio API response code: ");
    Serial.println(httpResponseCode);
    String payload = http.getString();
    Serial.println("Twilio API response: " + payload);
  } else {
    Serial.print("Twilio API request failed. HTTP response
code: ");
    Serial.println(httpResponseCode);
  }

  // Clean up
  http.end();
}
```

Upload this code to your ESP32 and open the serial monitor. The ESP32 will attempt to connect to Wi-Fi and send a text message using Twilio's API. You can observe the serial monitor output to check if the ESP32 successfully connects and sends the message. Be sure

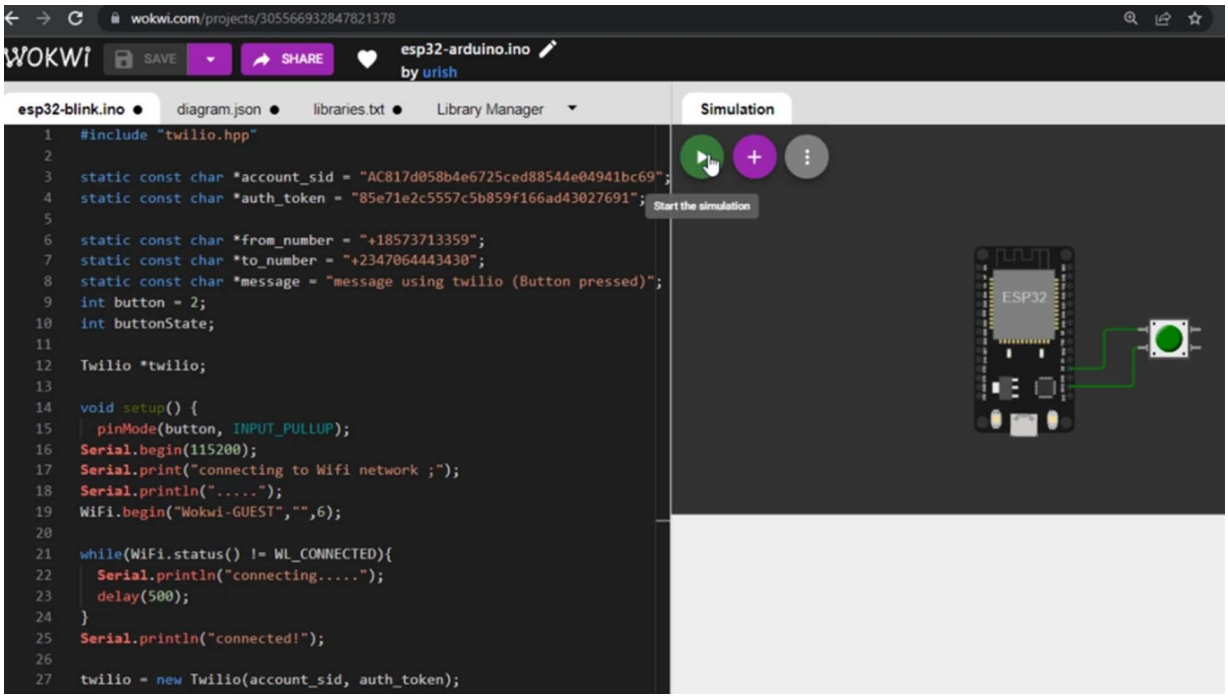
to replace the placeholder values with your actual Twilio and Wi-Fi credentials.

Keep in mind that testing Twilio integration often requires an active internet connection and valid Twilio credentials, and it may incur costs, depending on your Twilio usage.

SEND SMS ON PUSH BUTTON

In this episode we are going to send the SMS by pressing a push button. When a... SMS will go through. there. So connect the pin to digital pin 2... to the ground. There you go. You can see the inputs. So this is the code we used in episode 5. We'll be making some slight changes here. We'll change the message details to be button pressed. Then we'll come down and... resistor. So int... then button state which is a variable. We did not assign any value to it.

So in the loop we'll comment this out and then the int button state is going to be a digital read button. Then if button state is low. So when it is low we'll call the SMS function. Let's send the SMS. When the button state is high it does nothing. It's just blank. So it only sends the SMS when the button is pressed. That's when it's going to send the SMS. So let's simulate it and see what happens. And we have our mobile phone set up. It's connected to the internet. So now we are going to press the button.



So you press the... Twilio to be able to send the message. So we have received the message. That is the message but there is no indicator over this side that the message was sent successfully. So let's simulate again. So we'll press the button now. We'll press the button again. So in the code we did not put a condition that if SMS was successful, that is if the message was sent successfully, it should print SMS sent successfully. So we did not include that in our code. So that is why when we receive another message it won't... the serial monitor.

So we'll pause the simulation and we'll go back to our code. We'll just copy the one we commented on there. Copy and we'll paste it just below the SMS function. So we'll paste it and we'll remove the comments. Okay so let's simulate again. This time around we should get feedback from the serial monitor when the SMS has been sent successfully. So it's connected. Let's press our push button. Alright we press it and it's connecting to Twilio to send the

SMS. So we've received the message and you can see the feedback on the serial monitor sent message successfully. So if we press... will see the feedback on the serial monitor as well. So that Send SMS on the push button.

EXAMPLE DUMMY CODE

To send an SMS when a push button is pressed, you'll need an Arduino board (e.g., Arduino Uno), a GSM module (e.g., SIM800L), a push button, and a working knowledge of the Arduino IDE. Here's a basic example using Arduino and the SIM800L GSM module:

Cpp

```
#include <SoftwareSerial.h>

// Define the GSM module's TX and RX pins
SoftwareSerial mySerial(7, 8); // RX, TX

const int buttonPin = 2; // The push button is connected to digital pin 2
int buttonState = 0; // Store the current button state
int lastButtonState = 0; // Store the previous button state
unsigned long lastDebounceTime = 0; // The last time the button state changed
unsigned long debounceDelay = 50; // Minimum time between button state changes

void setup() {
  mySerial.begin(9600); // Initialize GSM module communication
  pinMode(buttonPin, INPUT);
  // Make sure the GSM module is ready (wait for a few seconds)
  delay(2000);
}
```

```
void loop() {
  int reading = digitalRead(buttonPin);

  if (reading != lastButtonState) {
    lastDebounceTime = millis();
  }

  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != buttonState) {
      buttonState = reading;

      if (buttonState == HIGH) {
        sendSMS(); // When the button is pressed, send an
SMS
      }
    }
  }

  lastButtonState = reading;
}

void sendSMS() {
  mySerial.println("AT"); // Test the GSM module
  delay(1000);

  mySerial.println("AT+CMGF=1"); // Set SMS mode to text
  delay(1000);

  mySerial.print("AT+CMGS=\"+1234567890\"\r"); //
Replace with the recipient's phone number
  delay(1000);

  mySerial.print("Hello, this is an SMS sent from your
Arduino!"); // SMS content
  delay(1000);

  mySerial.write(0x1A); // Ctrl+Z to end the SMS
```

```
  delay(1000);
}
```

In this code, we are using the SoftwareSerial library to communicate with the SIM800L GSM module. Connect the GSM module's TX to pin 7 and RX to pin 8 on the Arduino.

The push button is connected to pin 2 and configured with a debounce mechanism to avoid false readings.

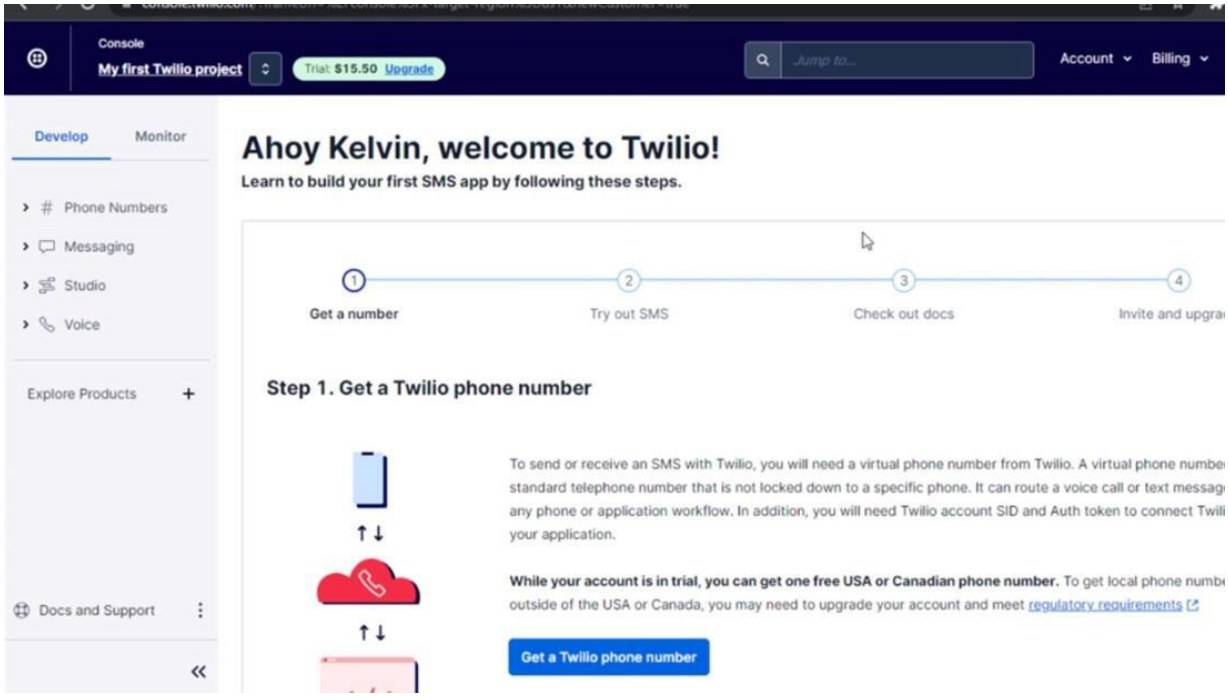
When you press the button, it triggers the sendSMS function. Modify the recipient's phone number and the SMS content as needed.

Note that you should have a working SIM card with a data plan and activated GSM module for this to work. Also, the power requirements of the GSM module may require an external power source, depending on your setup. Make sure to consult your GSM module's datasheet and documentation for proper setup and power considerations.

TWILIO SET UP

So we have tested our ESP online simulation with the WOPI software So we will go... Twilio... to We will go to the official website twilio.com So this is the website We'll click on sign up and start building. So when you click on sign up, you are going to fill out all these informations here I Start a free trial So I already have an account which I'm gonna log in with So I'll put my password

So after signing up on your new account and when you're logging in for the first time you have to register a phone number Which you will be using. So I'll put in a phone number and verify So a code will be sent to my phone. This is the code that has been sent to my phone So now I'm going to put in their code here 8 9 We will answer a basic question here. We are using the SMS products Alert and notification. We don't need code So get started with Twilio So skip this, we don't need all this Done.



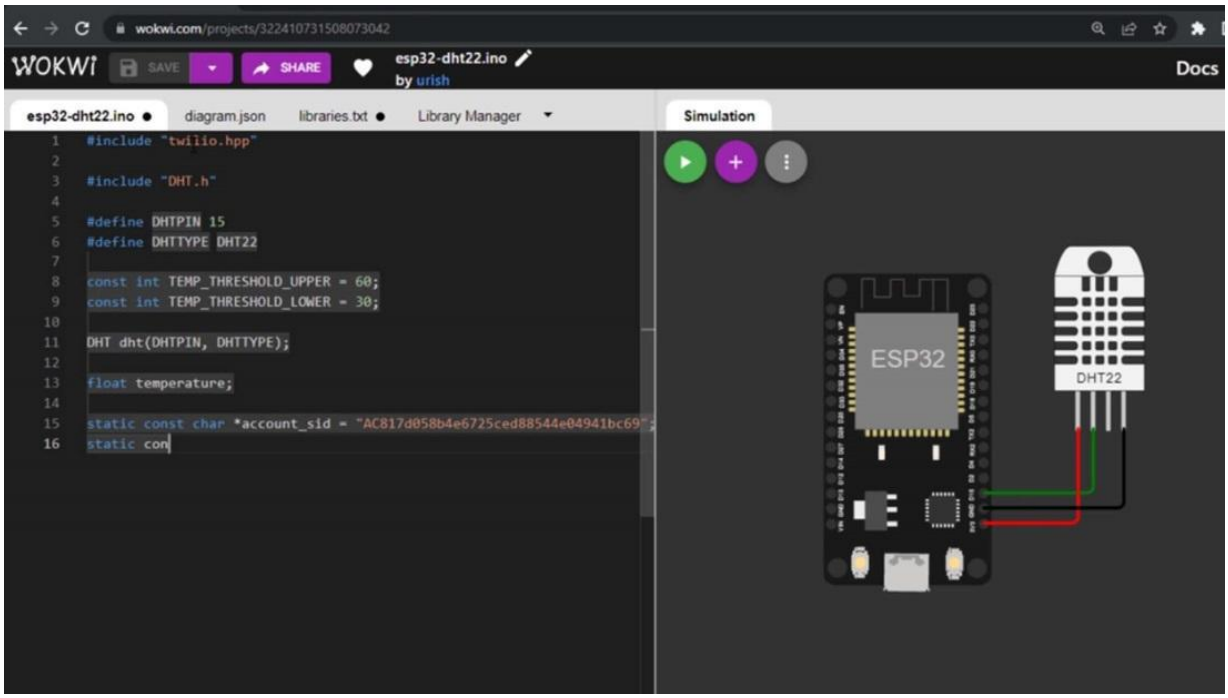
So when we enter our Twilio account First we'll go to phone numbers so that we can purchase a number and manage Buy a number. So we'll search for the number that has SMS Compactivity. So this one has SMS. We can use it to send SMS. So click on buy Click on buy Scroll down to where we have the buy button. We'll buy this So we've bought this now we can send SMS using this number that we've bought So let's go back Let's go to messaging Get set up Start set up message service name we'll just give it We'll just look for a simple name and give it.

We'll just give it ESP alerts ESP... this is a number. We'll add this number, the number we purchased so then this is our... my notepad Paste it in SID. Next I'll view my authentication token. I'll copy, open notepad And paste So try SMS After that you get to this point where you're trying to send SMS. So body of text can be anything can be hello from Hello from my ESP Testing Hello from my ESP test. So let's send the SMS. So the SMS has been sent to our phone So let me get a screenshot from my phone So that's the SMS and See how it comes in in the phone See how

SEND SMS CONTROLLED BY DHT22

In this episode The DHC 11 temperature and humidity sensor will be used to control the SMS that will be sent to our mobile phones. So when the temperature rises above a certain threshold, which we are going to set It will trigger the SMS. So let's begin. So that we are able to input our own code So let's install the library, the Flare library. Install the Twilio ESP clients library first Then we'll install the DHC sensor library. So let's begin the code Include the Twilio library which we already installed.

Next we'll include the DHC... DHC pin which is connected to pin 15 The DHC type is DHC 22. That's what we'll be using

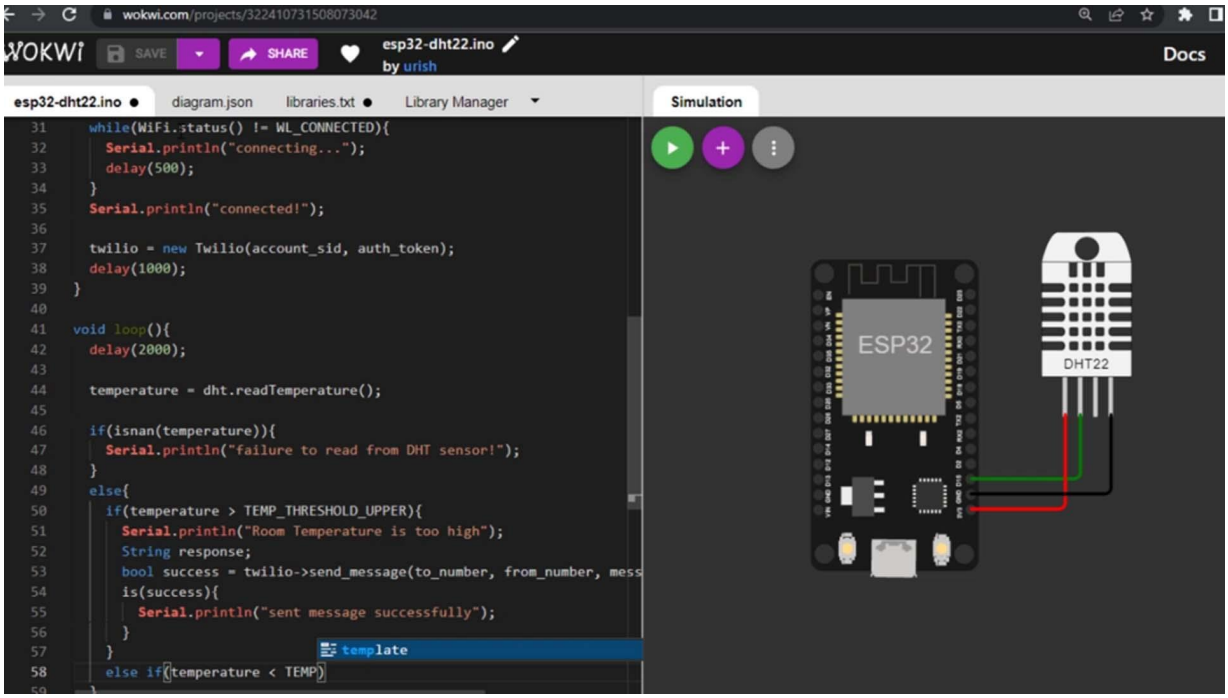


Then we are going to declare the temperature threshold for the upper value Which is 60. Then lower the value we'll use 30 Okay, let's leave it at 30. So float temperature that is. We've declared the Variable but we've not assigned any value to it So now the account's

SID which we... So be careful not to make mistakes. It's best you Copy and paste it Just type in mine out then my Authentication token. So

Next I'll declare another variable, the number which I'm sending from as the number which I purchased on the Twilio platform. I'll put it there and then the number which will be sent to which is my registered phone number Same as on the Twilio platform So the message content, it can be anything you want. So ours will be room temperature is too high so for the void setup Initialize the serial monitor Initialize the temperature and humidity sensor. Then initialize the connection.

The loop will set a delay in between the readings which is 2 seconds then temperature is equal to DHT read temperature Temperature was the float variable we... read from the DHT sensor so else our first condition if the temperature is above the temperature... 60 Serial.



print room temperature is too high and Then the send message function will follow else If temperature is lower than the temperature threshold lower Serial.print room temperature is normal. It won't send any SMS. the simulation

is coming up, okay So let's simulate connecting to the network So you can see the temperature is below 60 and the Serial.print room temperature is normal. So let's increase it above 60 So once you increase it above 60, it initializes the SMS system. It's connecting to Twilio... see the message has come into our mobile phone. So open it Room temperature is too high. As you see, let's take the temperature, let's take it lower. So after sending the SMS, it has come back.

It's gonna go back to room temperature. So the SMS is automated In regards to the level or how hot or cold the room is. So that is it for this project.

EXAMPLE DUMMY CODE

To send an SMS controlled by a DHT22 temperature and humidity sensor, you'll need an Arduino board (e.g., Arduino Uno), a GSM module (e.g., SIM800L), a DHT22 sensor, and a working knowledge of the Arduino IDE. Here's a basic example using Arduino and the SIM800L GSM module:

Cpp

```
#include <SoftwareSerial.h>
#include <DHT.h>

// Define the GSM module's TX and RX pins
SoftwareSerial mySerial(7, 8); // RX, TX

// Define the DHT22 sensor
DHT dht(2, DHT22);

// Set your phone number here
const char* phoneNumber = "+1234567890"; // Replace with your recipient's phone number

void setup() {
  Serial.begin(9600);
  mySerial.begin(9600); // Initialize GSM module communication
  dht.begin();
  // Make sure the GSM module is ready (wait for a few seconds)
  delay(2000);
}
```

```
void loop() {
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();

  if (!isnan(temperature) && !isnan(humidity)) {
    // Check if temperature exceeds a certain threshold
    if (temperature > 30.0) {
      sendSMS("Temperature is too high. Current
Temperature: " + String(temperature) + "°C");
    }

    // Check if humidity is too low
    if (humidity < 30.0) {
      sendSMS("Humidity is too low. Current Humidity: " +
String(humidity) + "%");
    }

    // Delay between readings
    delay(5000); // Adjust as needed
  }
}

void sendSMS(String message) {
  mySerial.println("AT"); // Test the GSM module
  delay(1000);

  mySerial.println("AT+CMGF=1"); // Set SMS mode to text
  delay(1000);

  mySerial.print("AT+CMGS=\"");
  mySerial.print(phoneNumber);
  mySerial.println("\"");
  delay(1000);

  mySerial.print(message); // SMS content
  mySerial.write(0x1A); // Ctrl+Z to end the SMS
```

```
  delay(1000);
}
```

In this code, we are using the SoftwareSerial library to communicate with the SIM800L GSM module. Connect the GSM module's TX to pin 7 and RX to pin 8 on the Arduino.

The DHT22 sensor is connected to pin 2 and configured to read temperature and humidity.

The loop function reads temperature and humidity from the DHT22 sensor and checks whether they exceed certain thresholds. If the conditions are met, it sends an SMS with the sensor data to the specified phone number.

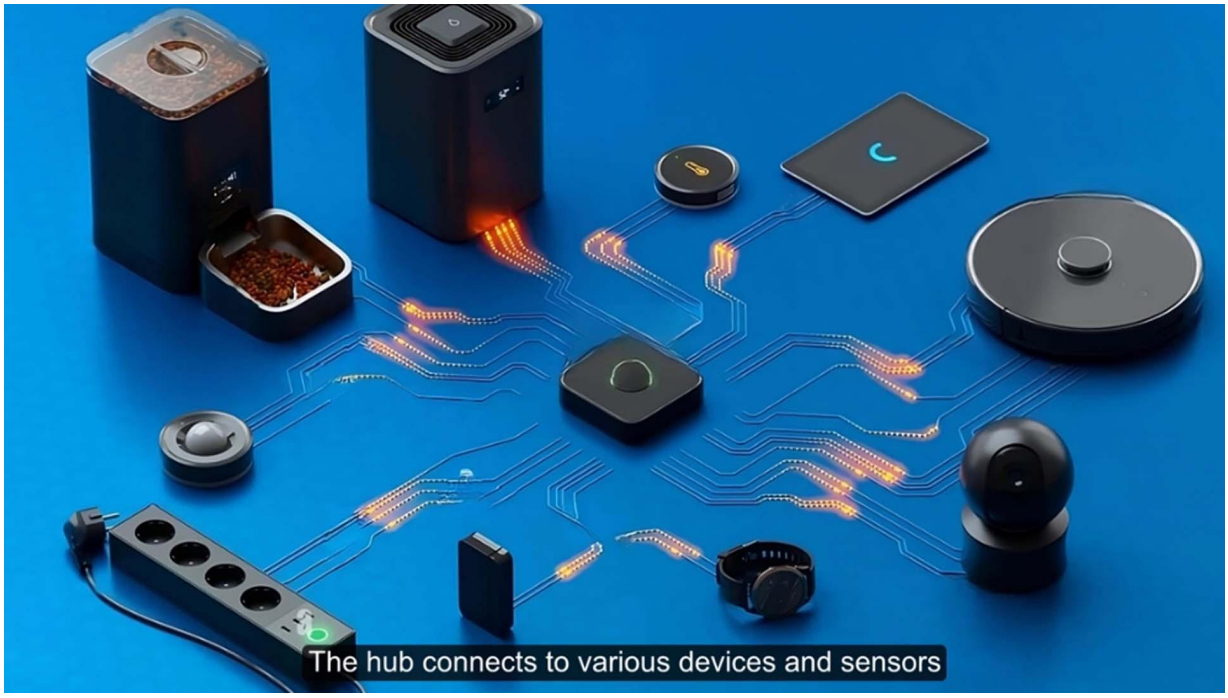
Replace phoneNumber with the recipient's phone number. Adjust the temperature and humidity thresholds as needed. The delay functions control how often the sensor readings and SMS sending are performed. Make sure you have a working SIM card with a data plan and an activated GSM module for this to work.

Keep in mind that the power requirements of the GSM module may require an external power source, depending on your setup. Make sure to consult your GSM module's datasheet and documentation for proper setup and power considerations.

INTRODUCTION TO HOME AUTOMATION

As we are going to develop a home automation project, it is crucial to have a clear understanding of the concept behind this technology. Home automation is a system that enables the control and automation of various electrical devices and appliances in a home. This technology uses different types of sensors, controllers, and actuators to provide greater control and convenience to homeowners.

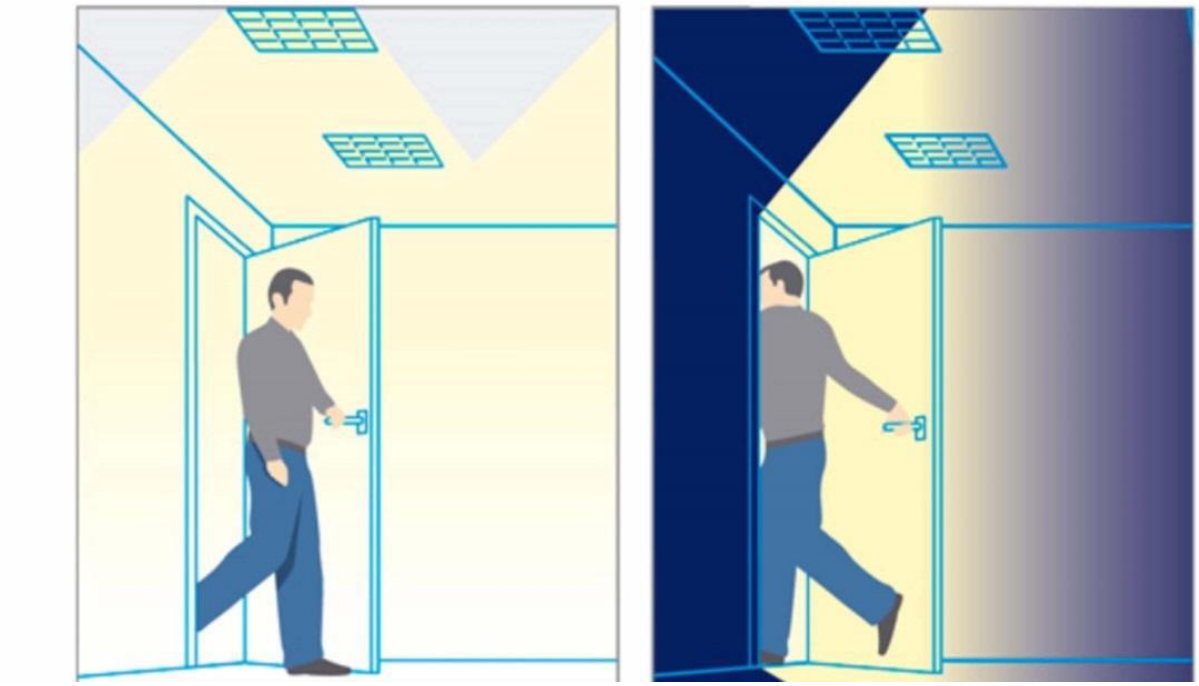
With home automation, one can control lights, heating and cooling systems, security cameras, smart locks, and other appliances using a smartphone, tablet, or a computer. Home automation systems typically consist of a central hub or controller that serves as the brain of the system.



The hub connects to various devices and sensors throughout the home and can be programmed to perform a wide range of actions, such as turning on lights when someone enters a room, adjusting the thermostat based on the temperature outside, or even starting the coffee maker in the morning. Home automation systems can also be integrated with voice assistants like Amazon Alexa or Google Assistant, allowing users to control their smart home devices using voice commands.

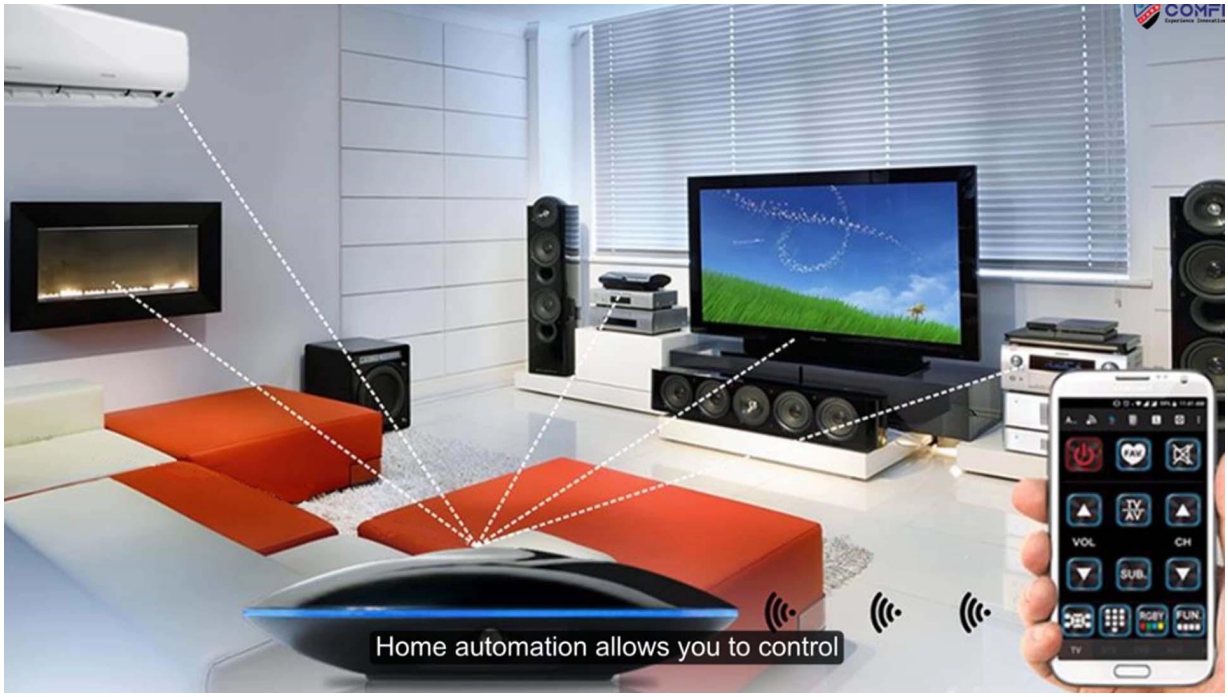
Overall, home automation can provide greater convenience, comfort, and security to homeowners while also helping to save energy and reduce utility bills. Applications of Home Automation Lighting Control Home automation allows you to control your lights remotely or automatically according to preset schedules or conditions. For example, you can set your lights to turn on when you arrive home or turn off when you leave. Climate Control Home automation lets you control your heating, ventilation, and air conditioning system remotely or automatically.

You can adjust the temperature and humidity levels in your home according to your preferences or occupancy patterns. Security and Surveillance Home automation enables you to monitor your home security and surveillance systems from anywhere using your mobile device or computer.



You can receive alerts when there is motion or noise detected, or when doors and windows are opened or closed.

Entertainment Systems Home automation allows you to control your audio and video systems, such as TVs, speakers, and gaming consoles from a central hub or remote control.

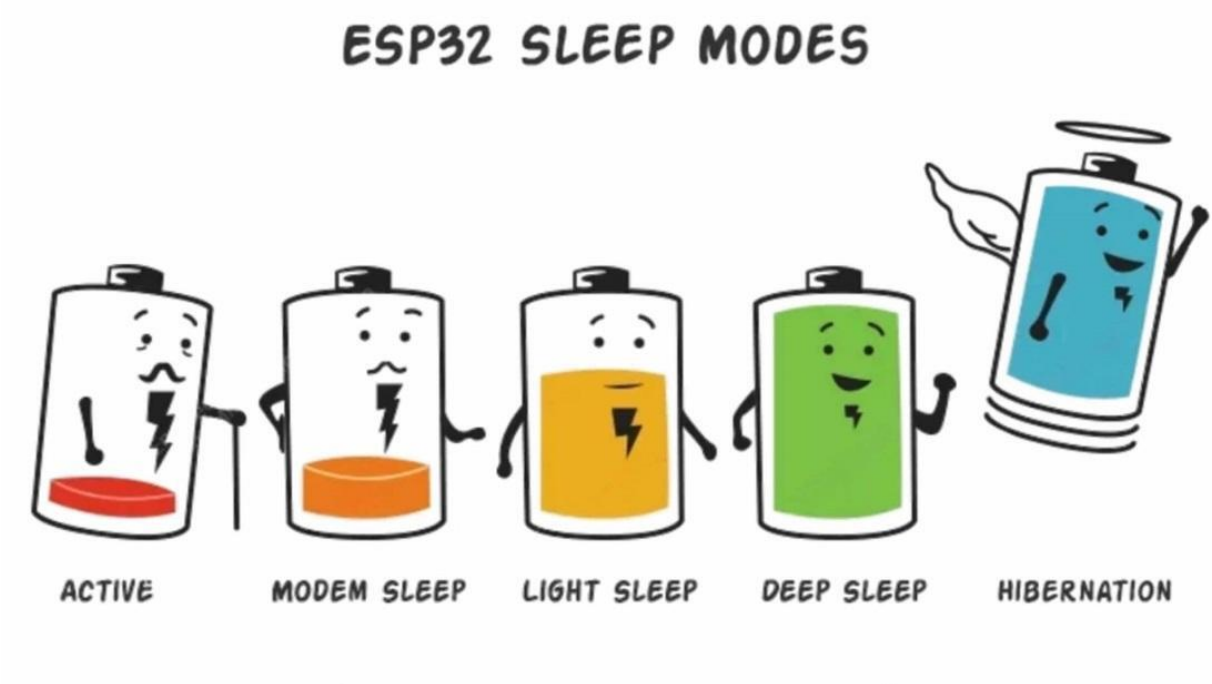


You can stream music and videos from your smartphone or tablet to your home theater system or play your favorite games on your big screen TV. Energy Management Home automation helps you to conserve energy and reduce your utility bills by automatically turning off lights, appliances, and other devices when not in use, or by optimizing the use of renewable energy sources such as solar panels and wind turbines.

Health and Wellness Home automation can support your health and wellness by tracking your fitness and health data, such as heart rate, blood pressure, and sleep patterns, and providing you with personalized recommendations and feedback. You can also connect with health and wellness professionals remotely for consultations and support.

GETTING STARTED WITH ESP32

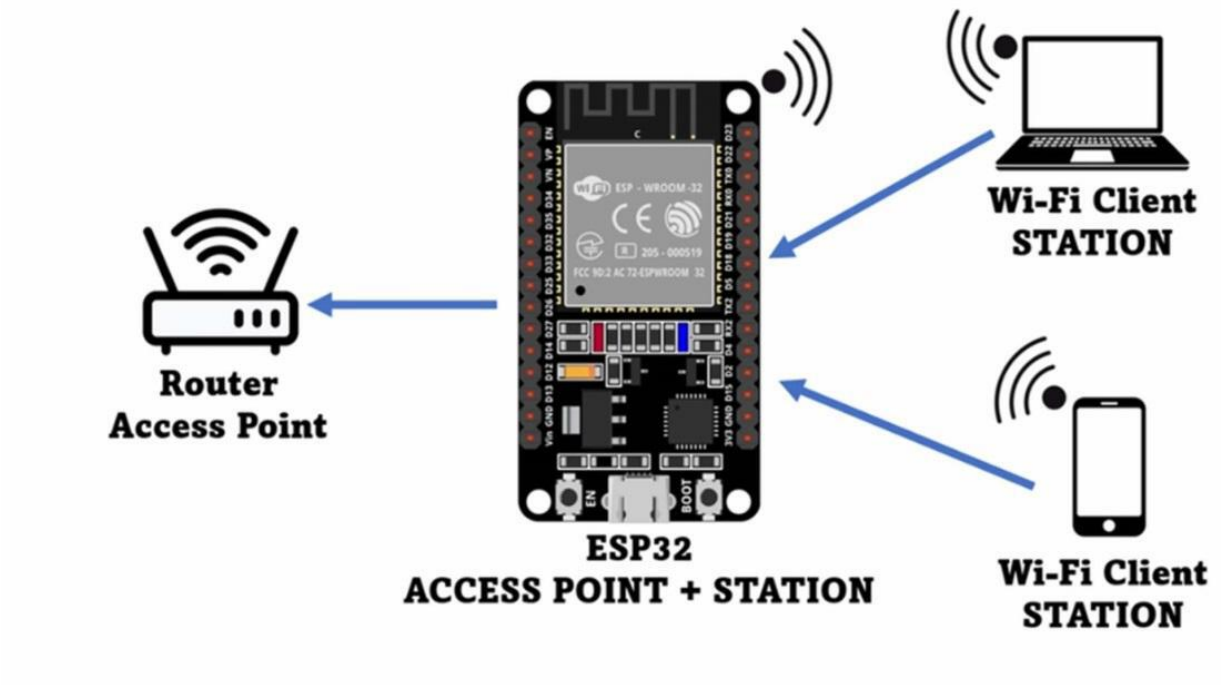
Before proceeding with the setup and testing of the ESP32, it is essential to gain a comprehensive understanding of this development board. The ESP32 is a powerful microcontroller that is designed to be low power and energy efficient.



It features two 32-bit processor cores and runs at clock speed of up to 240 MHz. This makes the ESP32 much more powerful than many other microcontrollers in the market and allows it to handle complex tasks and applications. One of the key features of the ESP32 is its built-in Wi-Fi and Bluetooth connectivity.

This allows the ESP32 to connect to the internet and communicate wirelessly with other devices, such as smartphones, tablets, and other IoT devices. The Wi-Fi connectivity also allows the ESP32 to act as an access point or a station, making it easy to integrate into

existing networks. The ESP32 also has a number of built-in peripherals, including GPIO, SPI, I2C, UART, ADC, and more.



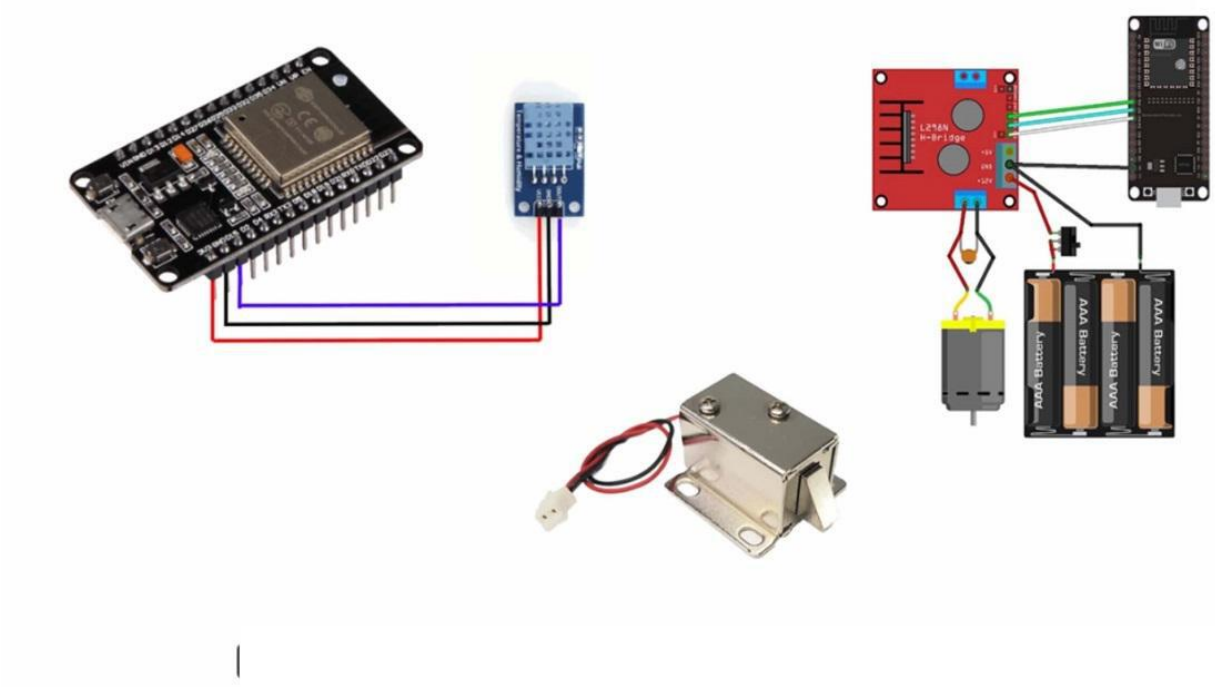
These peripherals make it easy to connect the ESP32 to a wide range of sensors, actuators, and other devices, and enable it to perform a variety of tasks, such as reading sensor data, controlling motors, and communicating with other devices. The ESP32 also has a real-time operating system that allows it to manage multiple tasks and run them simultaneously. This is important for IoT applications, which often involve multiple sensors and devices that need to be monitored and controlled.

Overall, the ESP32 is a powerful and flexible microcontroller that offers a wide range of features and capabilities for IoT applications. Its built-in Wi-Fi and Bluetooth connectivity support for multiple peripherals and real-time operating systems make it a popular choice for developers and hobbyists alike.

MASTERING GPIO PINS

When it comes to developing a hardware project, the general-purpose input-output pins play a crucial role. These pins serve as the interface between the board and the external components, making them a critical component of any project. Thus, gaining an in-depth understanding of the various functionalities of these pins is essential to the success of the project.

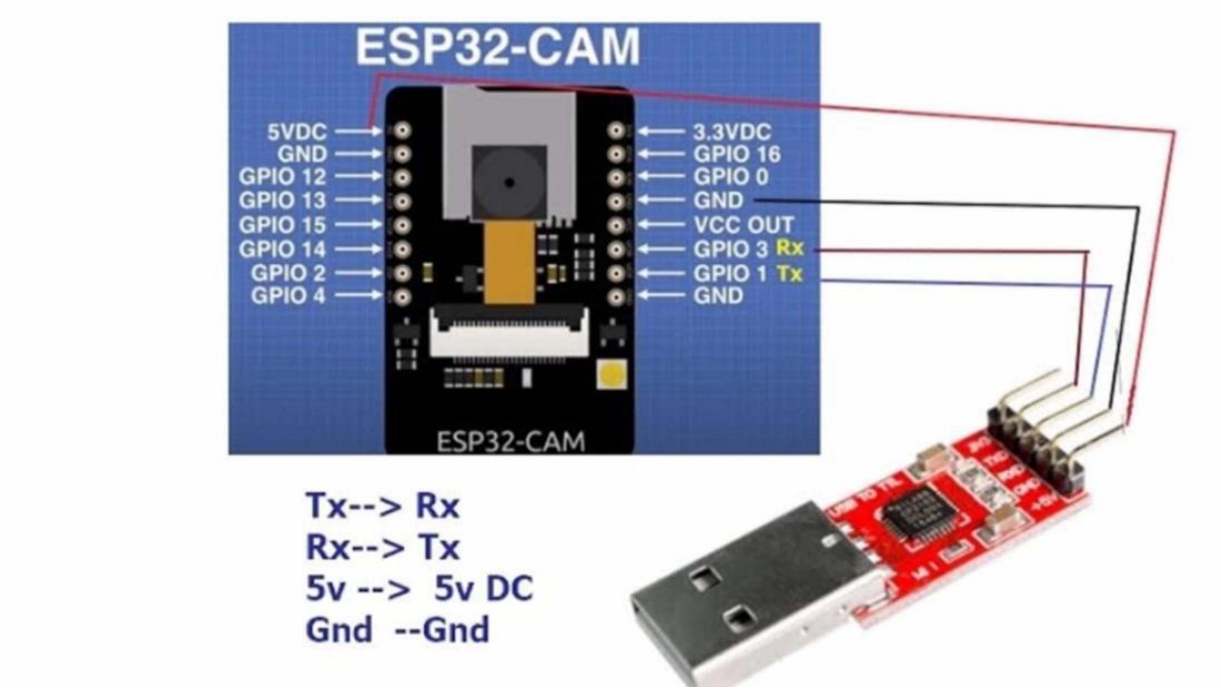
The ESP32 microcontroller has a wide range of GPIO pins that can be used for a variety of purposes, including interfacing with sensors, controlling motors and actuators, and communicating with other devices.



The ESP32 development boards come with so many versions, pin-outs, shapes, etc. In this project, we are going to use the ESP32 30-pin DevKit version 1 development board. This board is embedded with the microcontroller ESPWROOM32. This ESPWROOM32 IC

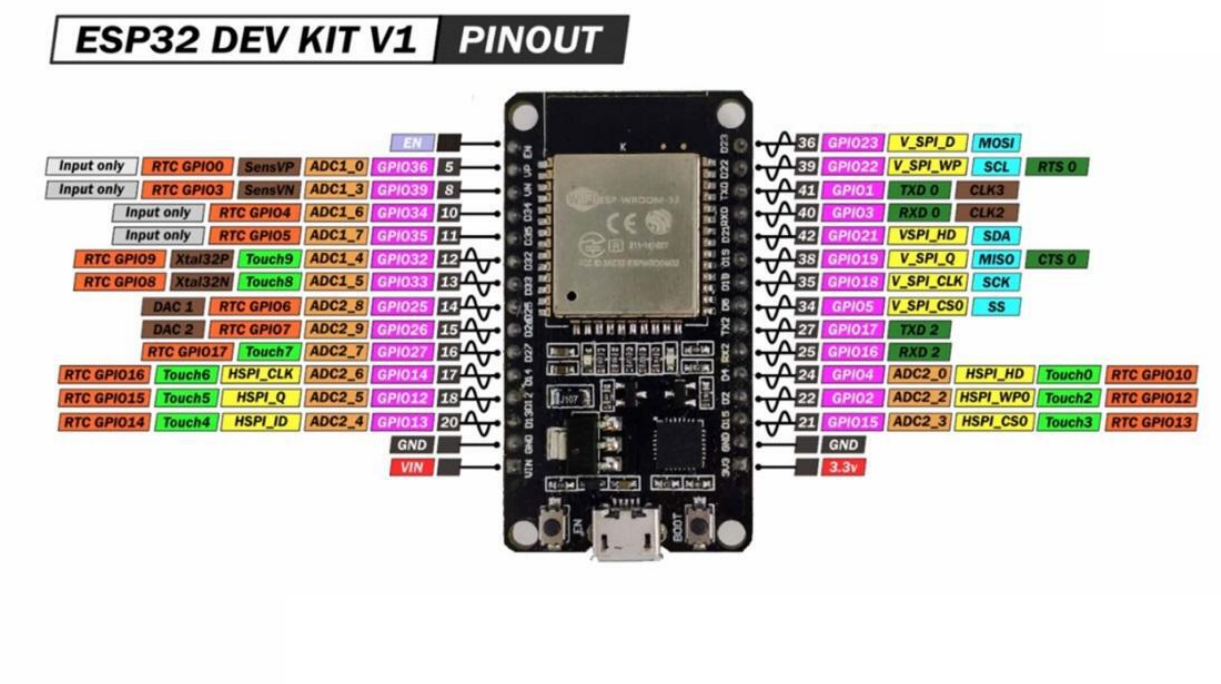
contains a 4MB SPI flash memory IC and a 40MHz crystal oscillator, PCB antenna, etc.

As the ESP32 board consists of a 4MB flash memory to store its primary program, some GPIO pins are directly connected to this flash memory. So, these pins can be used only for the programming of the board.



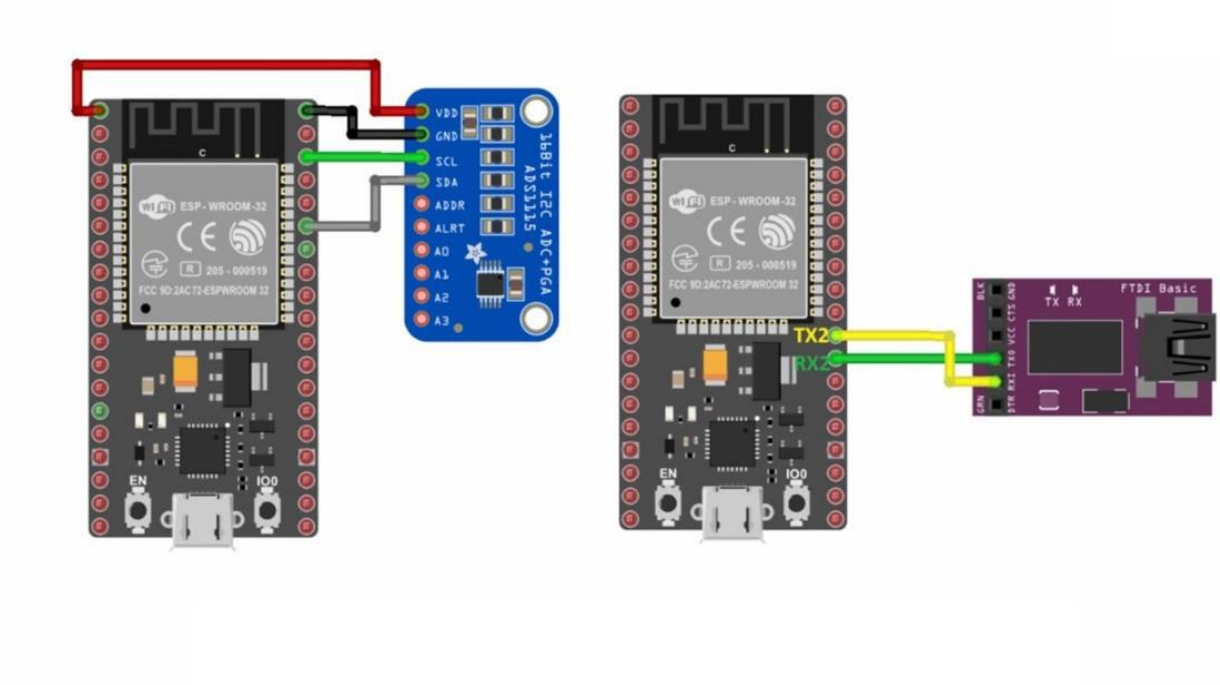
These pins cannot be used for other functions. Here you can see the pin-out diagram of the ESP32 DevKit development board. You can see each of the pins of this board as multiple functions. Each function is clearly noted with different colors.

Now, let us discuss these functions with respective pins to understand how to and which pins should be used. The term GPIO is abbreviated as General Purpose Input Output. board, 25 GPIO pins are available to connect with external circuits.

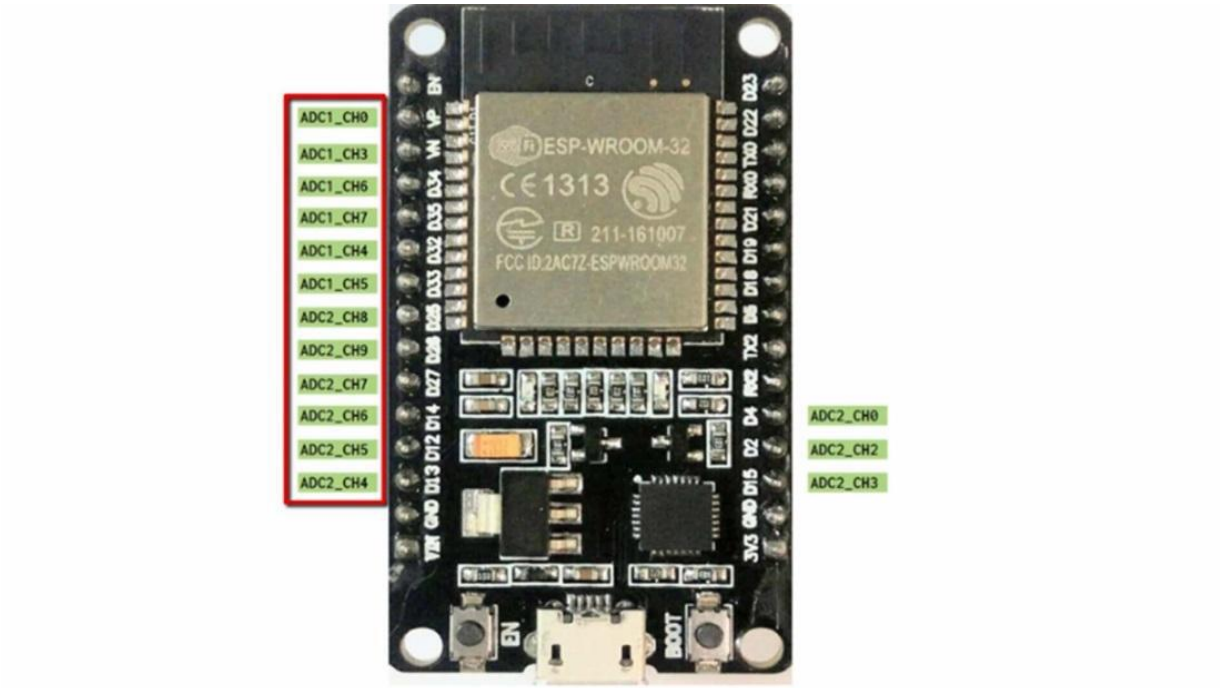


It also has some other GPIO pins that are connected internally with some ports and ICs. The GPIO pins are also used for other functions such as analog-to-digital converter, digital-to-analog converter, real-time clock, etc. But only one function will work at a time.

So, we can configure the GPIO pin as an ADC or an UART in the program. You can see in the diagram that pin number 13 to 36 on the left and 15 to 23 on the right have GPIO functionality. ADC means analog-to-digital converter.



The ADC pins help to connect external analog devices and components with this board. So, it can measure analog values such as voltage, current, etc. These ADC pins are also used in the sleep mode for low power consumption. The GPIO pins from 13 to 36 PWM control, etc. PWM means Pulse Width Modulation.

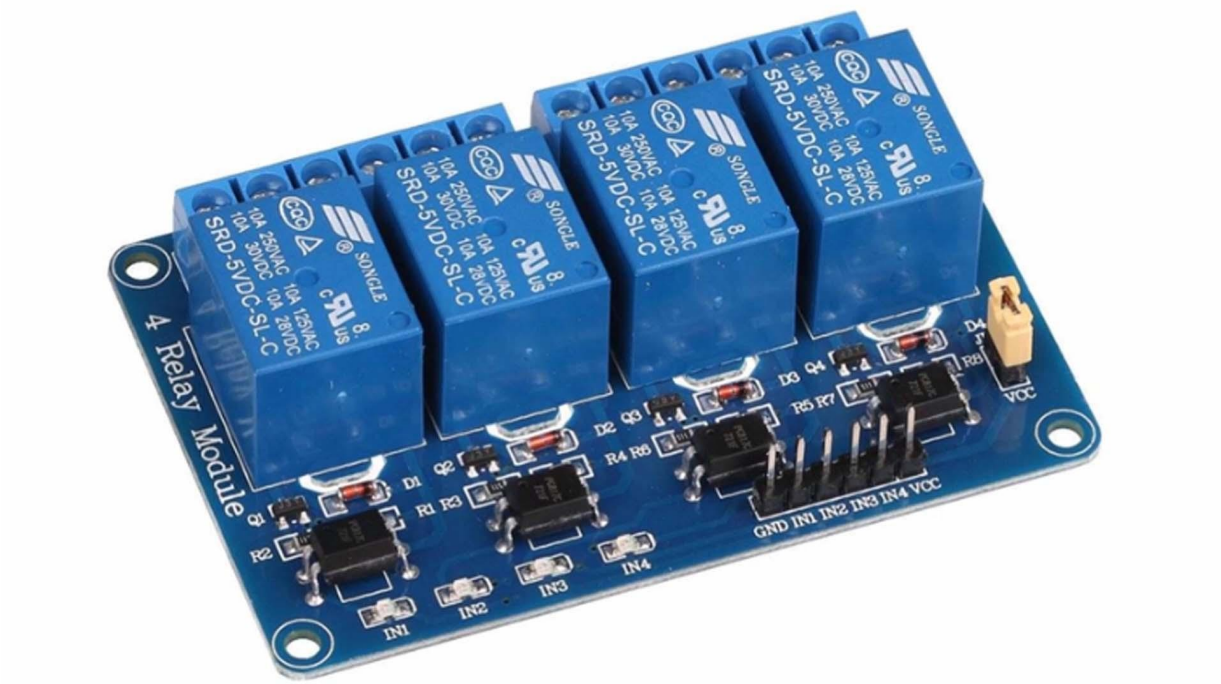


There is a difference between a normal digital signal and a pulse width modulated signal, although they look the same. The digital signal has a constant or fixed time period and frequency, whereas the PWM signal has a variable time period and frequency. The PWM function is very useful in applications such as motor speed control, brightness control, variable load controls, etc. In the ESP32 board, almost all the pins are PWM enabled except the EN, GND and VN on the left and GND and VDD on the right hand side.

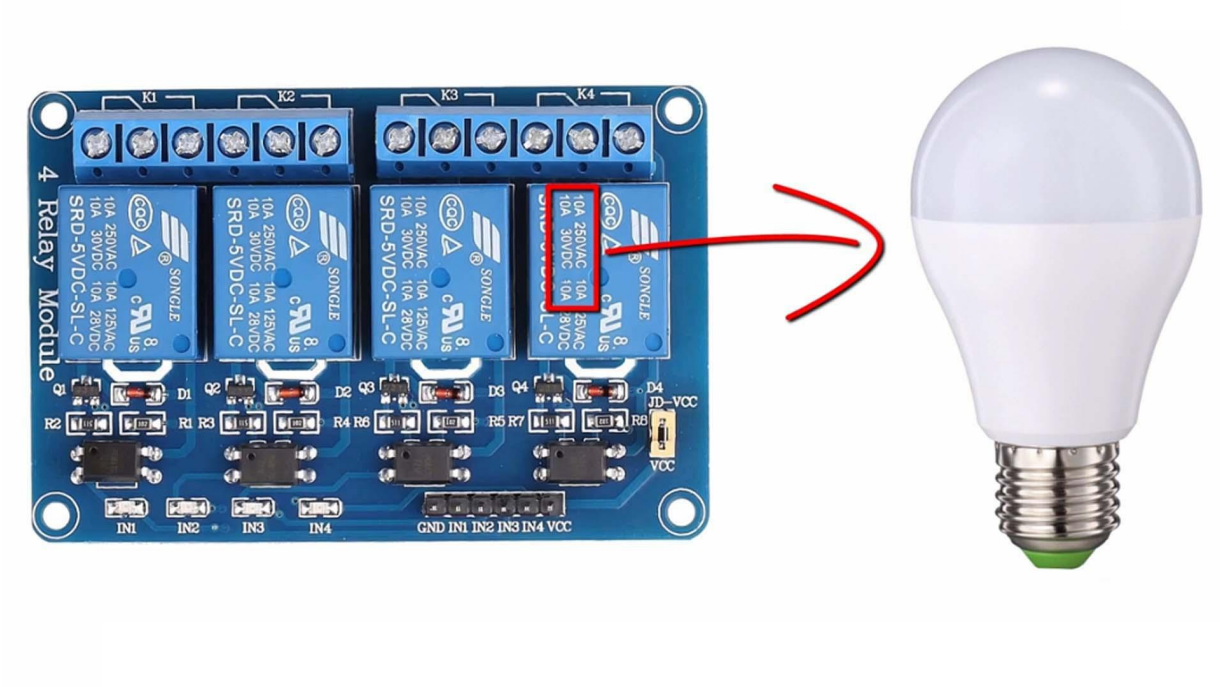
The 30-pin ESP32 board is equipped with 9 touch sensor pins, which are distributed across the board. The touch sensor pins are located on both the left and right sides of the board, with GPIO pin numbers 13, 12, 14, 27, 33 and 32 on the left side and pin numbers 15, 2 and 4 on the right side. These touch sensor pins can be used to detect touch inputs and enable touch-based user interfaces for various applications.

HARDWARE REQUIREMENTS FOR THE COMPLETE PROJECT

Let us see the ESP32 Development Board. This is the actual microcontroller board that will be used for programming and deploying the main project. Micro USB Cable. This will be used to power the ESP32 board along with that it will also help us to deploy the program on the board. Jumper wires.



These are used to create the connection between the ESP32 board and the relay. Approx 15 jumper wires are required for the project. 4-channel relay. This will act as a virtual switch to control AC appliances with ESP32. Bulbs. You can connect any appliances to the relay whose capacity is up to 10 amps.



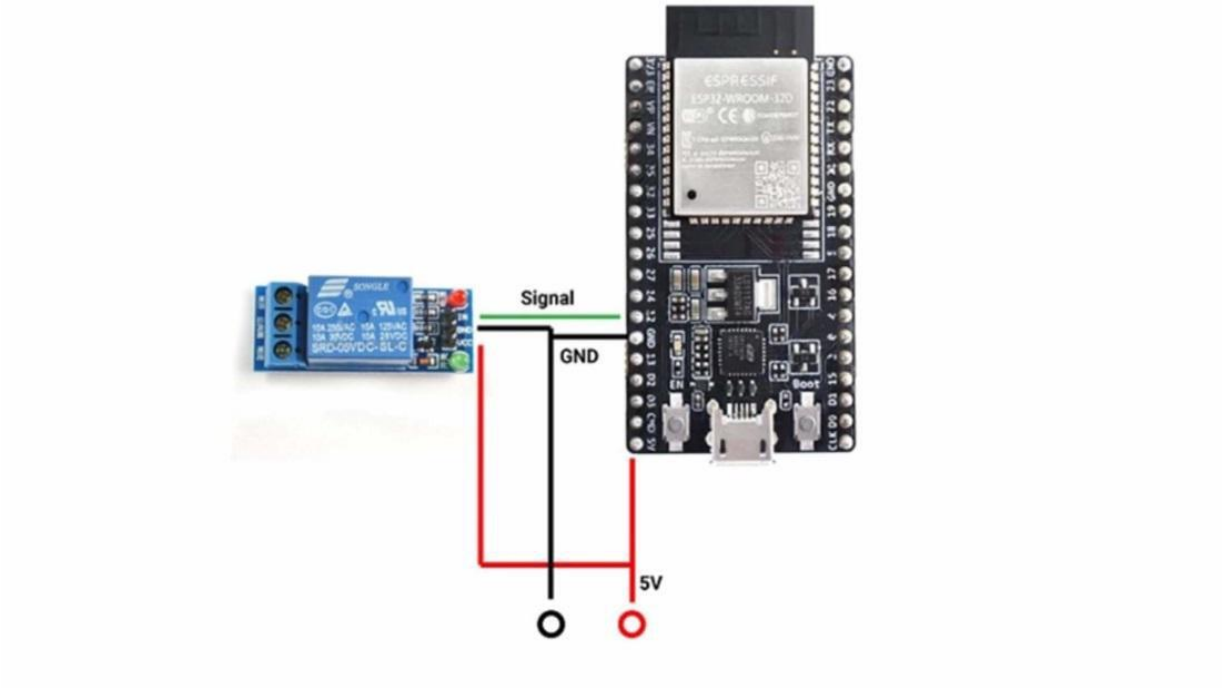
For the demonstration of the project, we are using four different colors of night bulbs whose voltage ratings are 240 volt AC. Bulb holders. These will be used for connecting the 240 volt bulbs. You should have four different holders to connect the four bulbs. Kindly note, if you are using a B22 type bulb, it is recommended to use holders that are specifically designed for B22 bulbs. Similarly, if you are using an E27 type bulb, it is best to use holders that are compatible with E27 bulbs.

This ensures a proper fit and secure connection between the bulb and the holder, allowing for safe and efficient operation. Electrical switch. In situations when internet connectivity is unavailable, we can use an electrical switch to manually control AC appliances. For this purpose, we require five switches. Four switches will be installed together in one board and the fifth switch on a separate board.

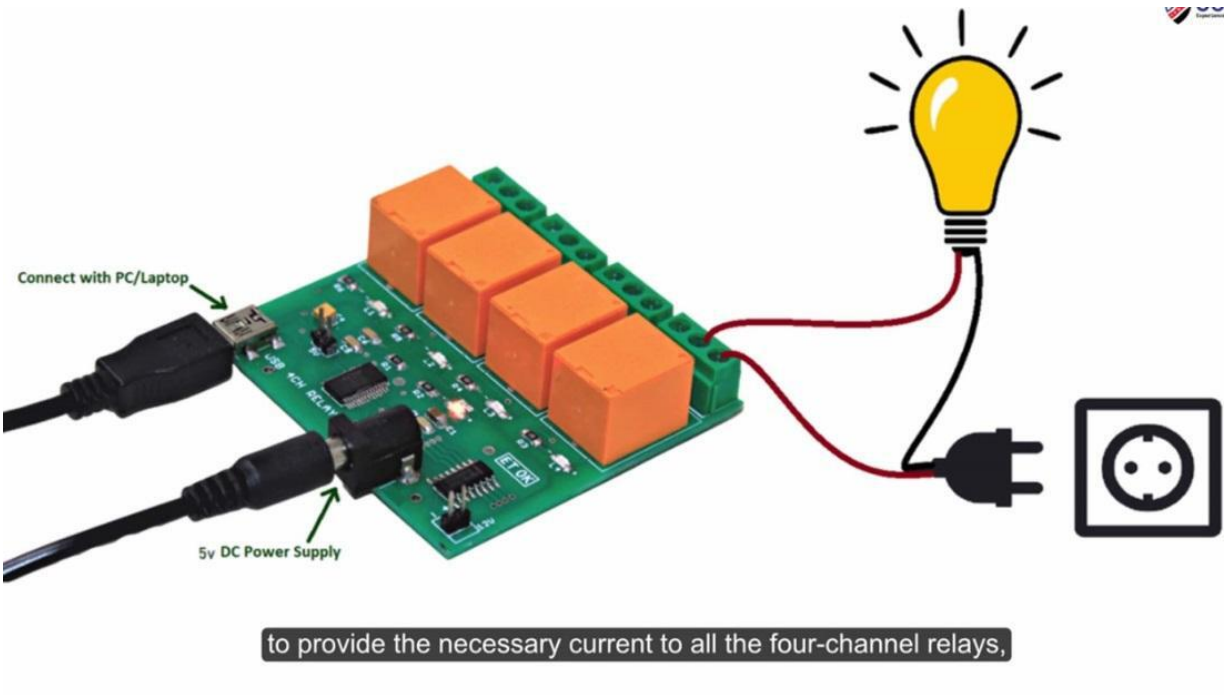
By keeping two ways to control AC appliances, we can ensure that our system remains reliable and functional even when the failure of internet connectivity occurs. Switch mounting box. To accommodate our switching needs, we will require two different types of switch mounting boxes. The first type will be used to hold an individual switch, while the second type will be capable of holding up to four switches. Copper wire 0.5 mm2. This will help us to connect the mains power supply with the AC appliances and the relays.

It is recommended that you obtain a high-quality twisted wire that is 3 meters in length. Using a reliable and durable wire ensures that you can effectively and safely connect your devices. Two-pin plug. This will help us to connect the wire with the 240V mains power supply socket. External power supply. The USB port on a laptop or desktop computer typically supplies a maximum of 5V and 0.5A of current.

This is sufficient to control a single-channel relay, but when we connect four relays to the ESP32, the board may not be capable of providing enough current to drive them all. In such a scenario, it is necessary to use an external power supply to provide the necessary current to all the four-channel relays, ensuring that they can function reliably and effectively.

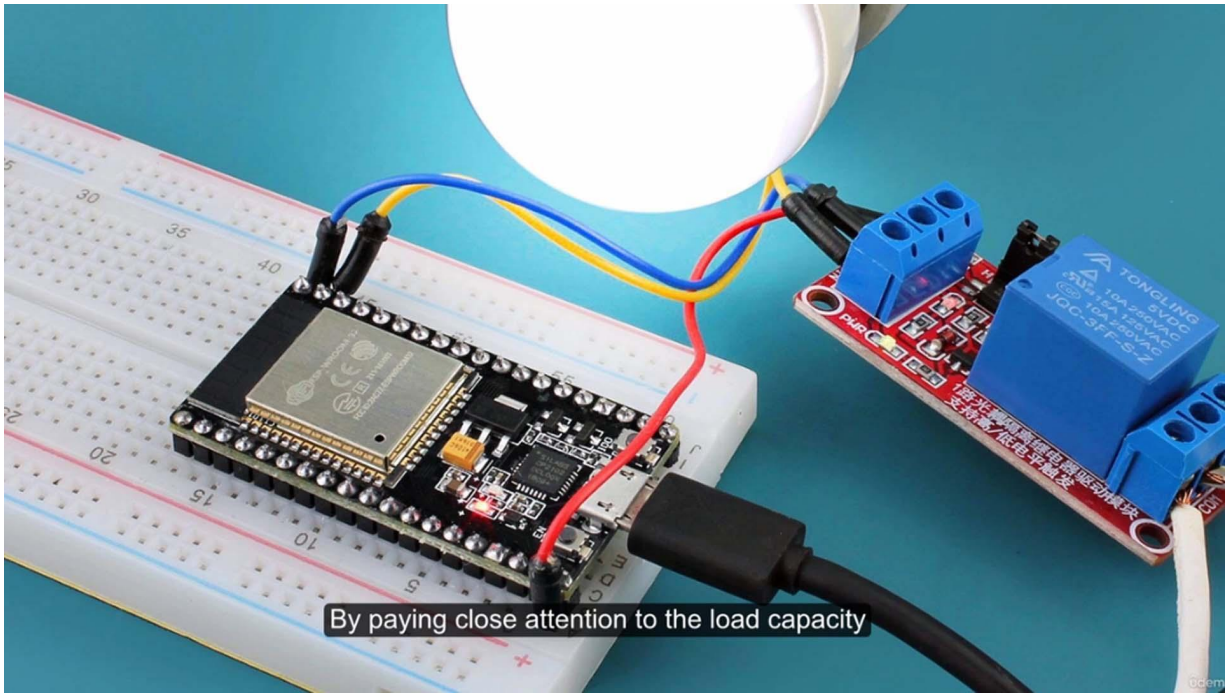


We need a 5V 2A power adapter. 220V table fan. During the final project demonstration, we will be replacing the fourth bulb that is currently connected to the relays with the fan.



This change will enable us to showcase the system's ability to connect and control a broad range of AC appliances, as long as their

load capacity is under 10A. It is important to note that we have deliberately chosen to limit the load capacity to 10A, as this is the maximum load capacity for the 5V relays that are commonly available for project purposes. Additionally, these relays are rated to handle up to 250V AC, making them suitable for a wide range of applications.



By paying close attention to the load capacity and voltage ratings of the components we use, we can ensure the safe and reliable operation of our system. Wire Stripper. A wire stripper is an essential tool that allows us to remove the outer plastic layer from a wire, enabling us to connect the wire to various components such as plugs, holders and relays. We will be also using this tool when we need to add jumper wires to the output of an external power adapter. Electrical Tape.

We will be using this tape to provide insulation and protection for electrical connections and wires. Screwdriver. We just need a general-purpose star-head screwdriver for tightly fixing the wire on the relays and a flat-head screwdriver to fix the switches on the board. Electrical Safety Hand Gloves. Working with a 240V mains power supply can be hazardous, which is why it is essential to take

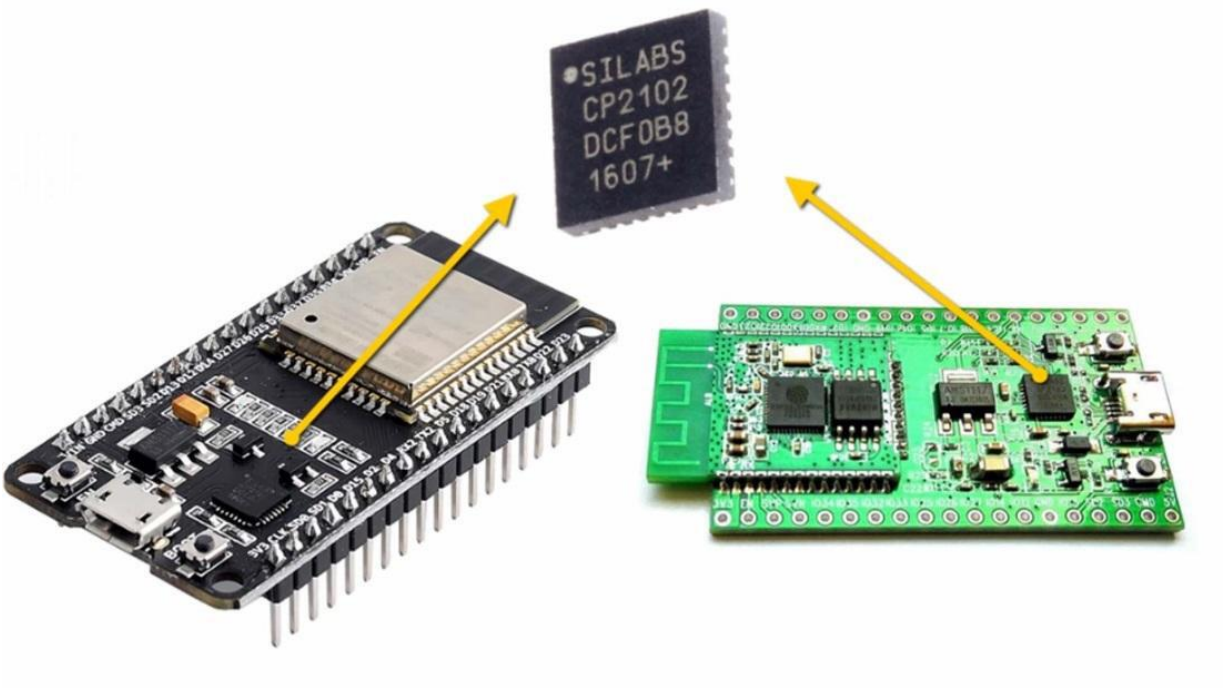
appropriate precautions. In such situations, an electrical hand glove comes handy.

These gloves are designed to protect you from electrical shocks and other electrical hazards that may arise during the electrical connection. By wearing electrical safety gloves, you can significantly reduce the risk of electrical accidents and ensure that you can work safely and confidently on your project.

CONNECTING AND VERIFYING THE USB TO UART CHIP IN ESP32

As you can see, we have an ESP32 microcontroller board and a microUSB cable. Upon closer inspection of the ESP32 board, we can see that it has 30 pins and the label ESP32 DEV KIT V1 is clearly visible when the board is inverted. To connect the ESP32 board to the USB port of your PC or laptop, we need to use a microUSB cable. At the edge of the board, there is a microUSB female port where we need to insert a male microUSB connector. Now connect the cable to the ESP32 board making sure that it fits properly.

Next we need to connect the other end of the USB cable to a USB port on our computer. Once connected, a red LED on the ESP32 board should light up indicating that the board is powered on. Before we proceed ahead, can you please tell me how you will make sure which USB to UART chip is embedded in your ESP32? Many ESP32 boards come equipped with a CP2102 USB to UART chip that allows them to communicate with a computer through a serial interface using a COM port. However, some boards may have a CH340 chip instead.

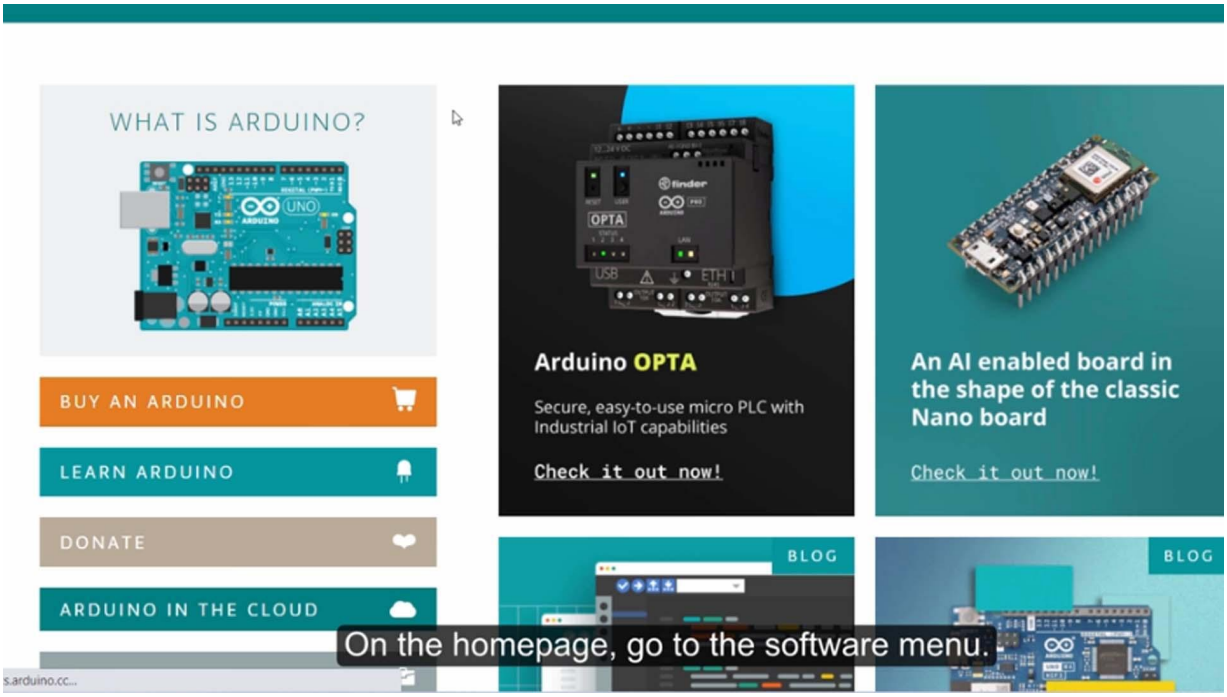


Before installing the appropriate driver, it is important to verify which chip is embedded in your ESP32. To do this, closely inspect the chip to determine its identity. Secondly, the device manager will also help you to find the same. Now let me check mine. Click on the Start menu and search for the device manager. Upon opening the application, you can find that in the Other Devices section, CP2102 USB to UART Bridge Controller is listed. So it confirms that I have CP2102 based ESP32. Upon clicking on it, you can see its properties.

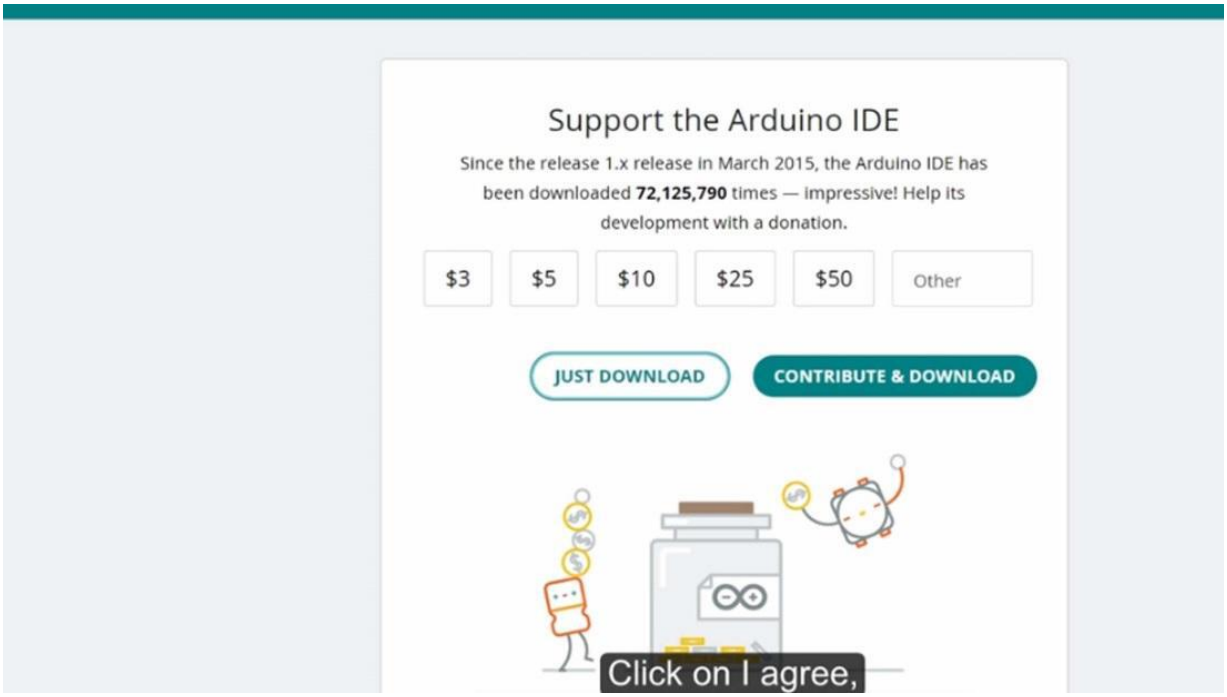
Error device status. It is mentioned that the device driver is not installed along with the error code 28. Please note that whenever you see the error code 28, it simply means that the driver for the particular device is not installed on your computer or the installed driver is corrupted or outdated.

ARDUINO INSTALLATION

Before installing the driver for the device, let us just check if the driver comes along with the Arduino IDE. So go to your web browser and type Arduino. Now we can see that the first web link is the official URL for Arduino. So click on it. On the home page, go to the software menu. Here we can see that we have a download section and the latest version of Arduino IDE is mentioned. On the right hand side, we can download the IDE based on different operating systems. In our case, we are going ahead with the first option.



It will take you to the download page. You may donate some amount to the Arduino team to support the ongoing development. It is optional and we can skip this by clicking on just download. The system will ask for the path where to download the file and after choosing the desired path click on save and it will start downloading. After successful download, open the installer. Click on I agree. Then on the next screen, we can see two options available, all users and current user.



If the PC is being accessed by multiple users, then you can go ahead with the first option. Here I am the only user, so I am going ahead with the second option. After making the choice, click on the next button. Thereafter, we can see the destination folder of the installation. We keep it as it is and click on install. It will start the installation, so wait for its completion. After successful installation, you have the option to tick whether you want to run Arduino IDE or not.

I am unticking this so it will not open this time and click on the finish button. Now let's again open the device manager and click on scan for hardware changes. As you can see, the driver for this device is still not installed even after the installation of the Arduino. This confirms that Arduino doesn't come with a CP2102 driver. Now let us open our Arduino IDE. We will see a blank sketch. Since the font size is small, we can increase the same.

Go to the edit menu and you can see here the increased font size option with a shortcut of ctrl plus equal to. We can use both ways by clicking the increase font size option or by using the keyboard shortcut.

EXAMPLE DUMMY CODE

To set up an ESP32 board in the Arduino IDE, follow these steps:

1. **Install the Arduino IDE**:
   If you haven't already installed the Arduino IDE, download and install it from the [Arduino website](https://www.arduino.cc/en/software).

2. **Install ESP32 Board Support**:
   To add support for the ESP32 boards in the Arduino IDE, you'll need to install the ESP32 board package. Follow these steps:

   - Open the Arduino IDE.
   - Go to "File" > "Preferences."
   - In the "Additional Boards Manager URLs" field, add the following URL:
     ```
     https://dl.espressif.com/dl/package_esp32_index.json
     ```
   - Click "OK" to close the Preferences window.

3. **Install ESP32 Board Package**:
   Now, you need to install the ESP32 board package. Here's how:

   - Go to "Tools" > "Board" > "Boards Manager..."
   - In the "Boards Manager" window, type "esp32" into the search bar.
   - You should see "esp32" by Espressif Systems. Click the "Install" button to install the package.

4. **Select Your ESP32 Board**:
   After the installation, you can now select your specific ESP32 board from the list. Here's how:

   - Go to "Tools" > "Board."
   - Select your ESP32 board model from the list. For example, "ESP32 Dev Module" or "NodeMCU-32S."

5. **Select the COM Port**:
   - Go to "Tools" > "Port" and select the COM port to which your ESP32 is connected. The COM port number will vary depending on your system.

6. **Write and Upload Code**:
   You can now write your code for the ESP32 in the Arduino IDE. Here's a simple "Hello, World!" example for the ESP32:

```cpp
void setup() {
  Serial.begin(115200);
}

void loop() {
  Serial.println("Hello, World!");
  delay(1000);
}
```

7. **Upload Code**:
   - Click the right-facing arrow icon (or "Upload" in the "Sketch" menu) to upload your code to the ESP32.

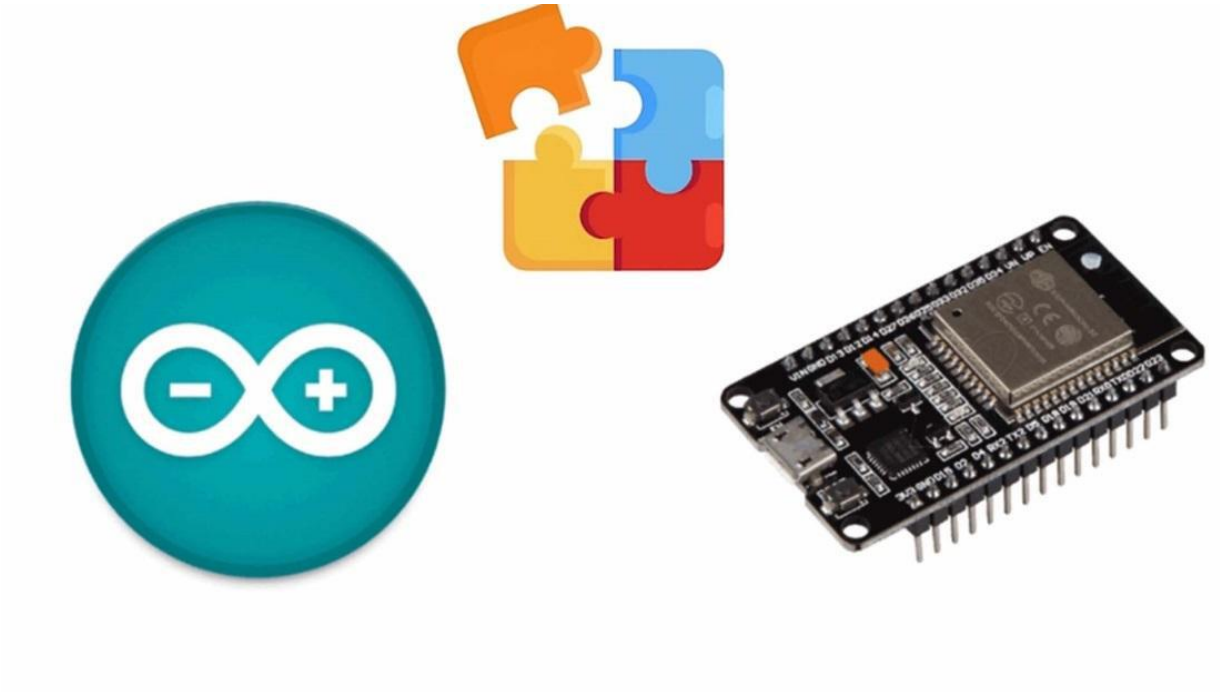
8. **Monitor Serial Output**:
   - After successfully uploading your code, open the Serial Monitor (Tools > Serial Monitor) to view the serial

output. Set the baud rate to 115200 to match your code's configuration.

Your ESP32 is now set up in the Arduino IDE, and you can start developing and uploading code to it. Make sure you have the necessary drivers installed for your ESP32 board if you encounter any connectivity issues.

SETTING UP ESP32 IN ARDUINO IDE

Now, we are ready to set up ESP32 in the Arduino IDE. So, let's start. Go to the File menu and select the Preferences option. It will open a small window. Scroll down and you can see there are additional board manager URLs that are currently empty. Please note that the Arduino IDE doesn't come with an ESP32 board library. Hence, to enable the ESP32 board compatibility with the Arduino IDE, it is necessary to install the ESP32 board library. We will do this by adding the ESP32 board's URL here.



I have the URLs written in the notepad, which I will share with you in the resource section of this lecture. Just copy the two URLs one by one and paste them into the preferences window. As it is separated by a comma, the IDE will automatically consider it as two different URL links. Now click on OK and it will start downloading some packages required for ESP32 boards. After the download gets

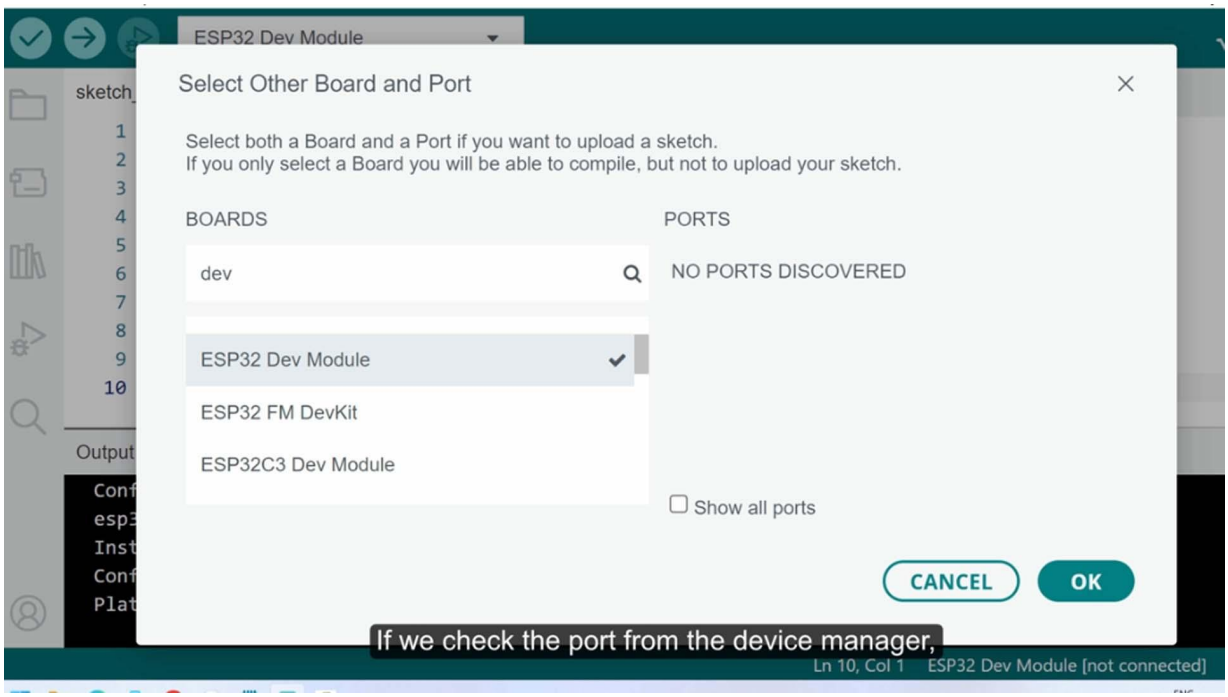
completed, click on the board manager icon located on the left side. Type ESP32 in the search box.

It will show a board having the name ESP32 by Espressif Systems. If we click on the More Info link, it will open the GitHub repository of ESP32 in the browser. Go back to the IDE. Please note that the latest version of the ESP32 board library is not compatible with our ESP Rainmaker program and installing the latest version will result in unexpected errors during development. So select version 2.0.3 from the drop-down menu and click on Install. It will start downloading packages for the board.

It will take some amount of time, so kindly wait for its completion. When the installation is complete, you can notice that it is showing version 2.0.3 installed. In place of the install button, it is now showing the update button. Kindly note that since we have installed an older version of the ESP32 board library, you may get an update notification in your Arduino IDE. Please make sure you always avoid the same. Now let's go to the board and port option.

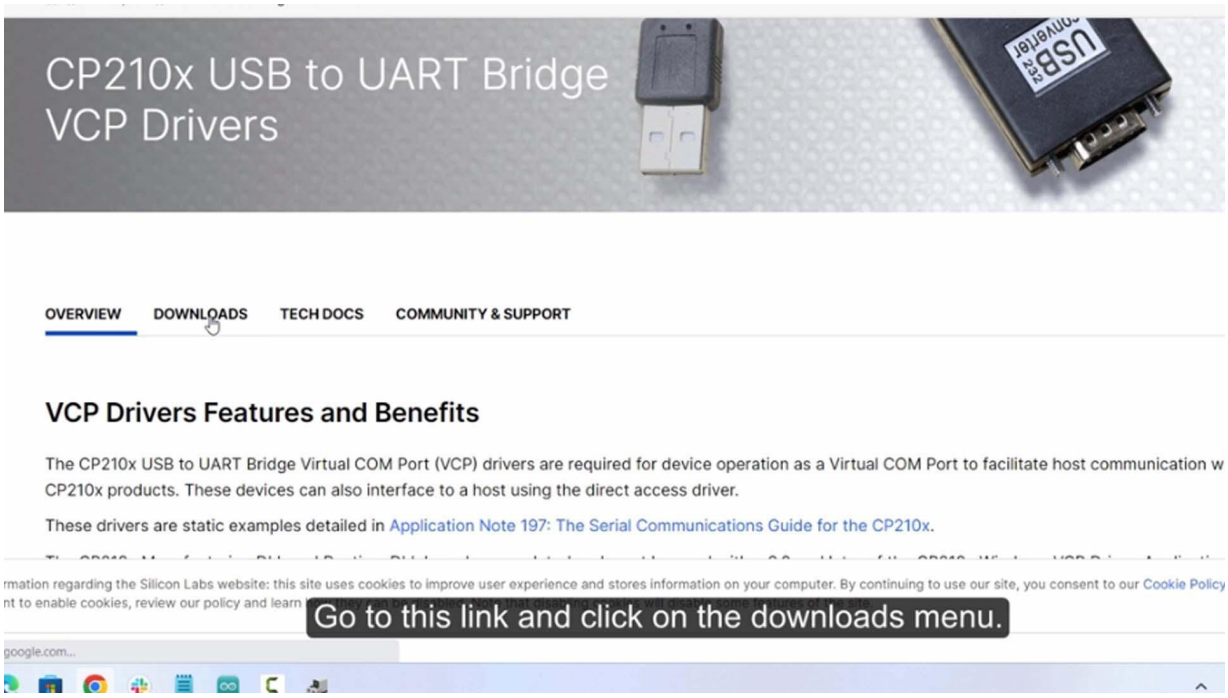
If we type dev, we can see the ESP32 dev module is listed in the boards, but there is no port available in the list. If we check the port from the device manager, we can still see that the CP2102 is listed in

other devices and it is showing the same message that the drivers for this device are not installed. So we need to manually install the driver for this chip.



Go to the browser and search for CP2102 driver and you can see the official web link from Silicon Labs. Go to this link and click on the Downloads menu.

Here we can see the list of software download links. We will go ahead with the universal driver. Upon clicking on it, it will ask you to choose your destination folder. Click on the save button and the download will start.



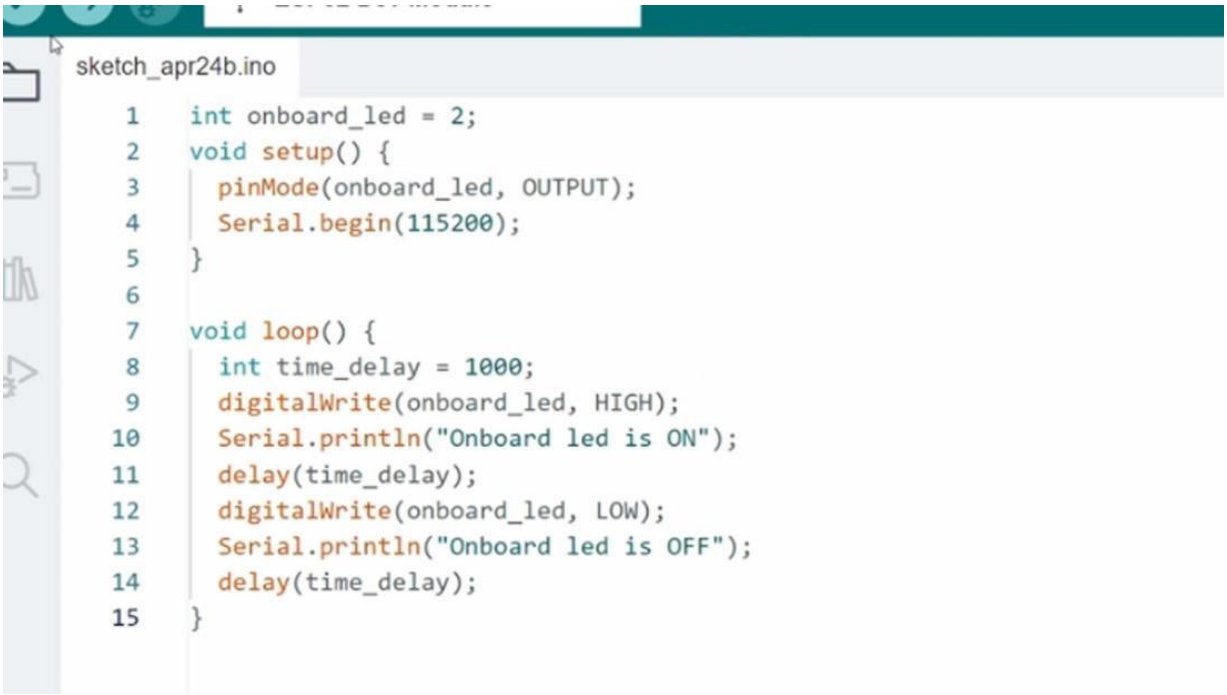
After the download gets completed, open the folder where the file has been downloaded. This is a zip file, so first extract it and we can see all the extracted files here. Now go to the device, right click and choose the update driver. Click the second option and browse the location of the CP2102 driver which you had extracted earlier.

Choose the parent folder, click ok and then click on the next button. The driver will be installed and a message will be displayed confirming that the driver has been updated successfully. You will notice that in the device driver hardware list, the CP2102 device is now listed inside the ports section and upon clicking on it, it shows that the device is working properly and the virtual com port number 3 is allocated to this device. Now in the Arduino ID, inside the board and port window, the same USB COM port is visible.

This means that our ID also detected the port, so select the same and click on ok. You can notice that on the bottom side, there is written ESP32 dev module on COM3. It indicates that the computer has recognized the connected ESP32 board and is ready to communicate with it.

TESTING THE ESP32 BOARD (PART 1)

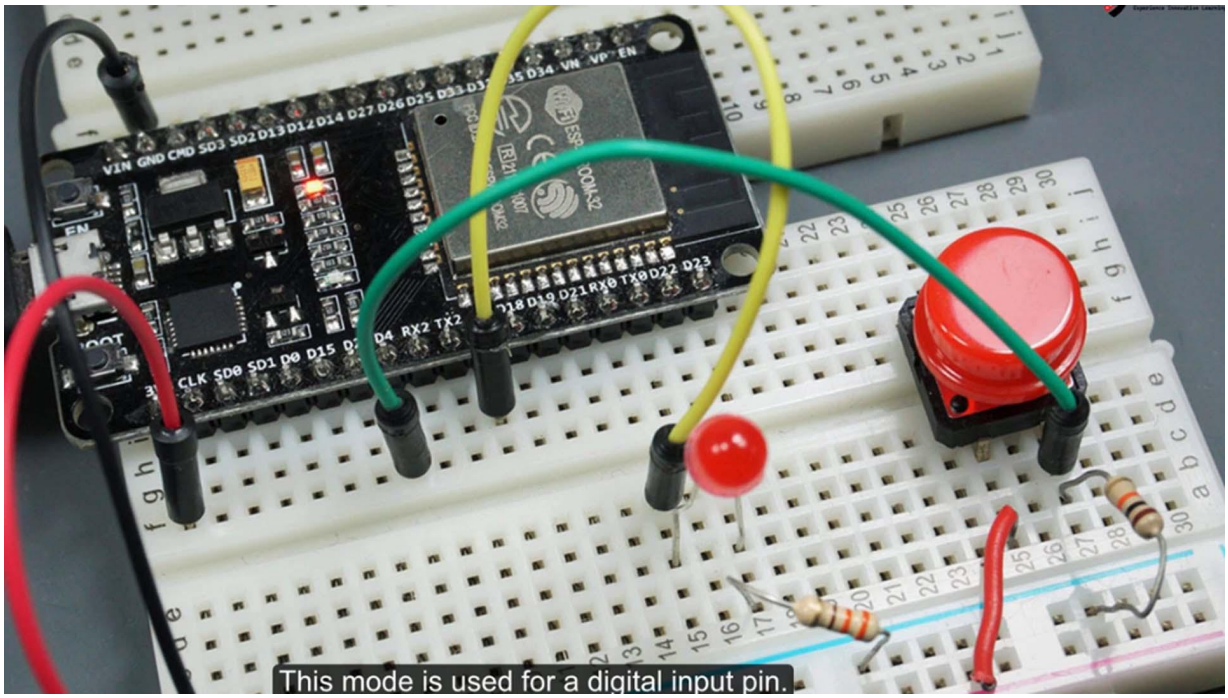
Now that our ESP32 board is properly set up, we can proceed to test it with a basic program. This program is responsible for blinking the inbuilt LED on the ESP32 board. Carrying out this test will ensure that the board which we have purchased online or from a local market is functioning properly or not. In case it is not working, you can immediately ask the seller for a replacement. Let us now go through the code. In the first line of code, we are declaring a variable.



This is the data type of a variable followed by the name of the variable and then the assignment operator and finally the value of the variable. So the onboard LED variable will store the inbuilt LED pin number 2. We have mentioned this because the LED is internally connected to pin number 2 and we don't have to make any external connection. This variable is declared above void setup so that we

can access this global value from anywhere in our program. This is the default setup function which does not return any value.

That is why you can notice that it is mentioned as void before the function name. The pin mode function is used to configure the mode of a digital input or output pin. The function sets the pin mode as either input, output or input pull up. The first argument takes the digital pin number that we want to configure and the second argument specifies the mode of the pin. The possible modes are. Input This mode is used for a digital input pin. The pin will be configured to read the state of an external device connected to it.



Output This mode is used for a digital output pin. The pin will be configured to send a digital signal to an external device connected to it. Input Pull-up This mode is used for a digital input pin with an internal pull-up resistor. The pin will be configured to read the state of an external device connected to it with an input pull-up resistor connected internally. Since an LED is an output device, we need to send a signal to light it. That's why we have mentioned it as an output.

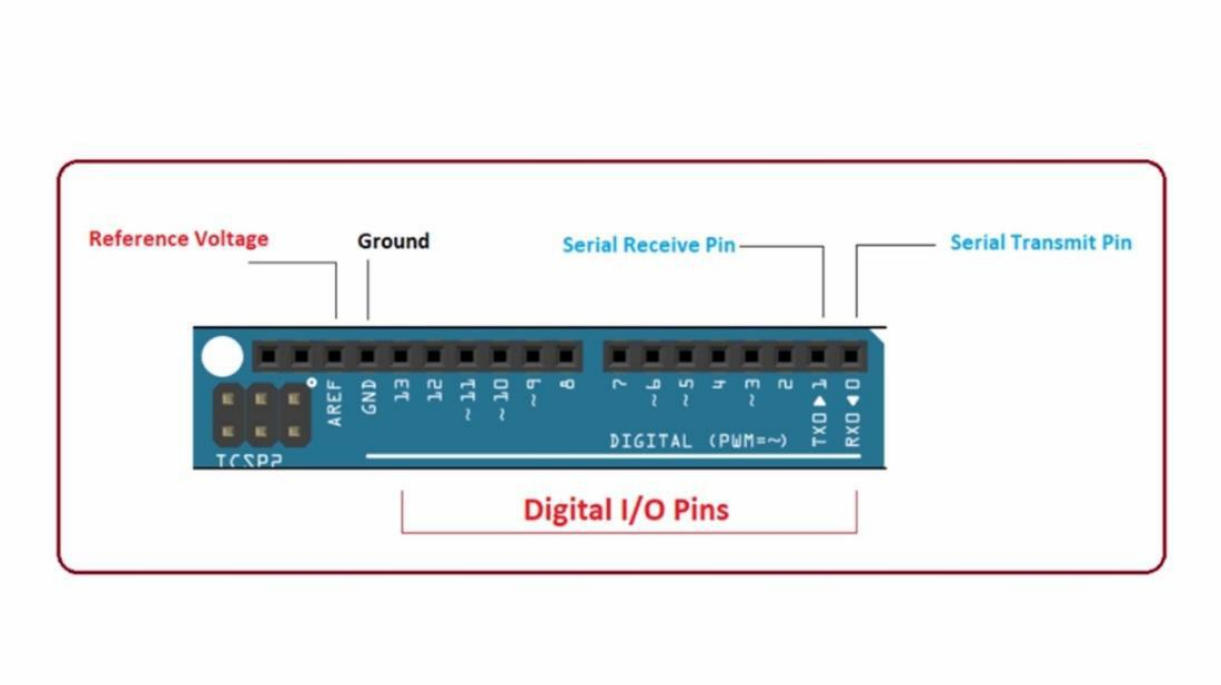
This function is used to initialize the serial communication at a specific baud rate between the ESP32 and the connected device,

such as a computer, another microcontroller, or a sensor module. Inside the parenthesis, we have passed the baud rate. The baud rate is the speed of the serial communication measured in bits per second. Common baud rates include 9600, 115200, 57600, and 38400. The recommended baud rate for ESP32 and PC communication is 115200.

This baud rate ensures reliable and fast communication between the ESP32 and the PC, and it is widely used in many ESP32 projects.

TESTING THE ESP32 BOARD (PART 2)

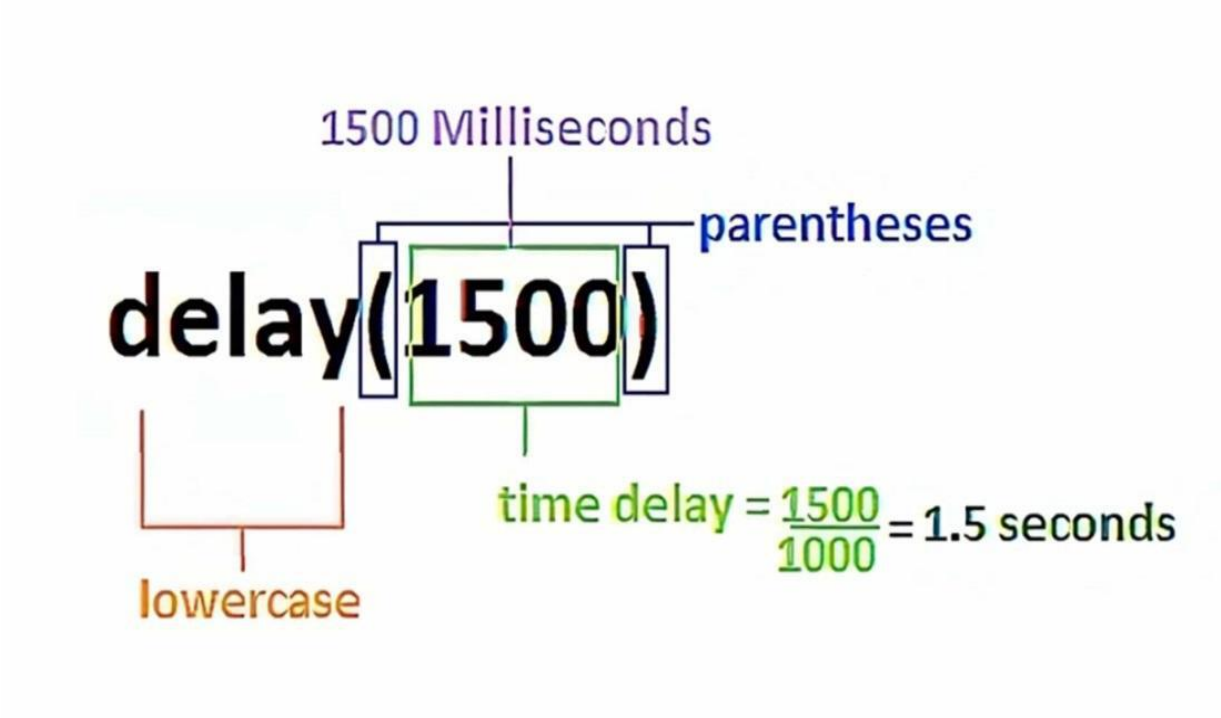
The loop function is a built-in function that runs continuously after the setup function has been completed. The loop function is where we put the main code of our program and it repeats itself until the microcontroller is powered off or reset. Here we have created a variable named time underscore delay and its value is set to 1000. We will use its value in the upcoming code. The digitalWrite function is used to set the voltage level of a digital pin on the microcontroller.



Digital pins can be either input or output and digitalWrite is used only with output pins. Here the pin is the number of the digital pin to be set and the value is either high or low to set the voltage level of the pin. In our case, we have passed onBoard underscore led, the variable contains the value 2, so here pin number is 2 and the value is high, this means it will glow. Then we simply print the message on

the console that the onBoard led is on. Next we have written the delay function.

This function is used to pause the program for a specified amount of time.



The delay function takes one argument which is the amount of time to pause the program in milliseconds. In our case, we have passed the variable time underscore delay as the parameter, which means that the program will pause for 1000 milliseconds which represents 1 second. So this means that the lead will remain on for 1 second. Now we are using all the three functions again but with different parameters.

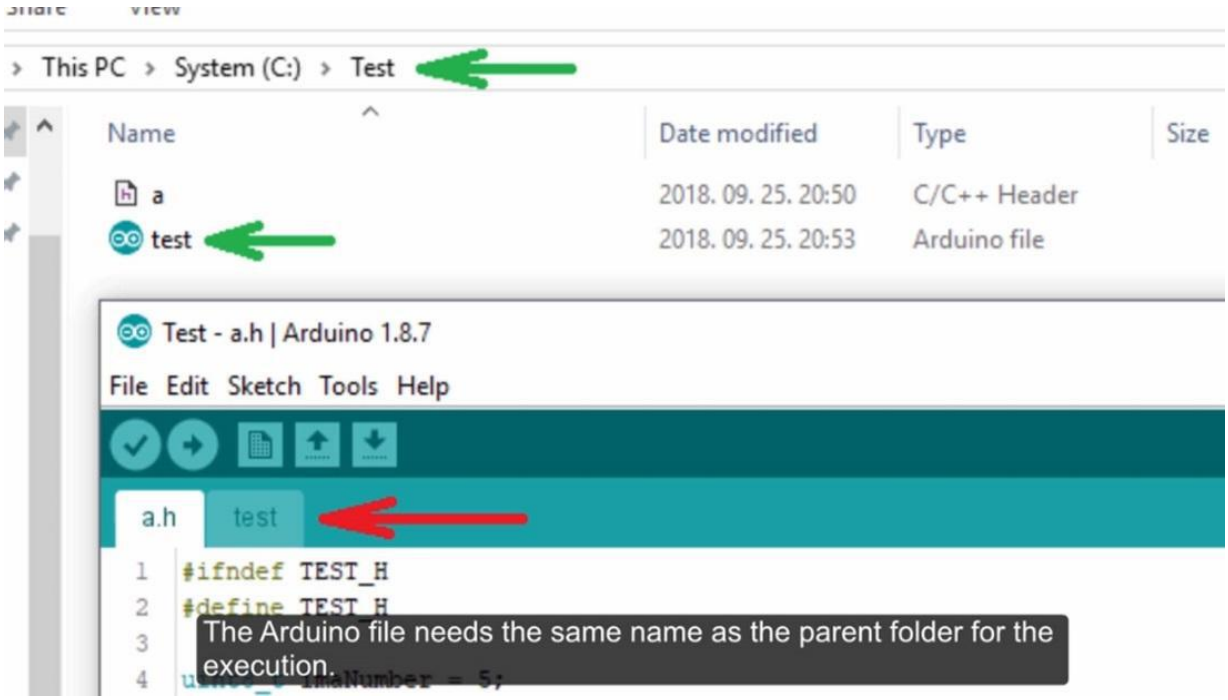
```
1   int onboard_led = 2;
2   void setup() {
3     pinMode(onboard_led, OUTPUT);
4     Serial.begin(115200);
5   }
6
7   void loop() {
8     int time_delay = 1000;
9     digitalWrite(onboard_led, HIGH);
10    Serial.println("Onboard led is ON");
11    delay(time_delay);
12    digitalWrite(onboard_led, LOW);
13    Serial.println("Onboard led is OFF");
14    delay(time_delay);
15  }
```

Inside the digitalWrite function, we have passed onBoard underscore led as the first parameter and the value low as the second parameter. This line of code will turn off the led and it will print the message onBoard led is off. To read the message properly and to keep the led turned off, we will pause execution for 1000 milliseconds. You can change the duration as per your convenience. So when this part of the code is repeated continuously, it will create a blinking pattern. Now before executing this program, first save it.

Go to the file menu and click on the save option. Choose the desired folder where you want to save the program. Here I am saving this file inside the ESP32 code directory and renaming it as testing underscore ESP32. If you navigate to this specific directory, you will notice that the Arduino software has automatically created a new folder with the same name that you have given while saving the program. Apart from that, the program is also moved inside this newly created folder. So this is how Arduino saves the program.

The Arduino file needs the same name as the parent folder for the execution. Now go back to the Arduino ID, select the board and port option and type deb on the board search box. Select the ESP32 dev

module and virtual com port and then click on the ok button. Now we are ready for the code execution.

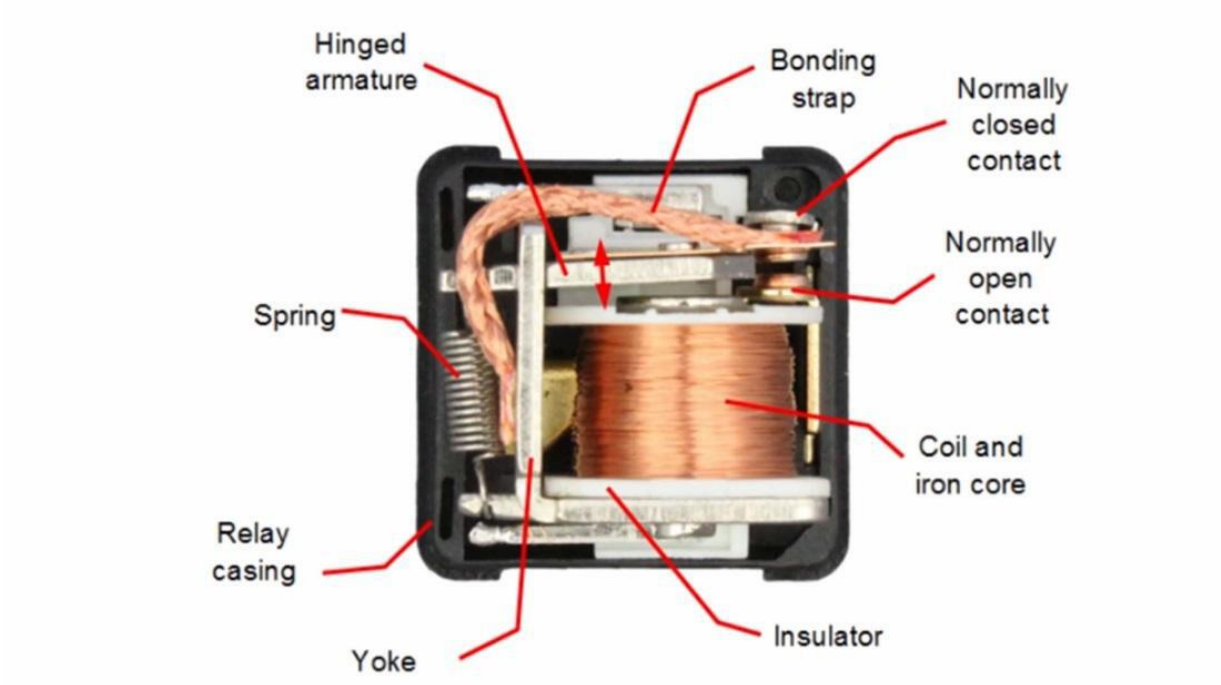


Let us first compile it using the right symbol icon and you can see that in the output terminal window no errors were found. So finally click on the arrow key button to upload the code on the ESP32 board.

After uploading begins, wait for the output message on the output console window. When it shows 100% completion then go to the tools menu and select the serial monitor. This will open one more console window beside the output console. Here you can see the message onboard LEDs on and onboard LEDs off getting printed one by one. On the hardware side, we can notice that the blue LED on the ESP32 is turning on and off creating a blinking pattern. So this confirms that our ESP32 is working fine and we can proceed ahead with the project development.

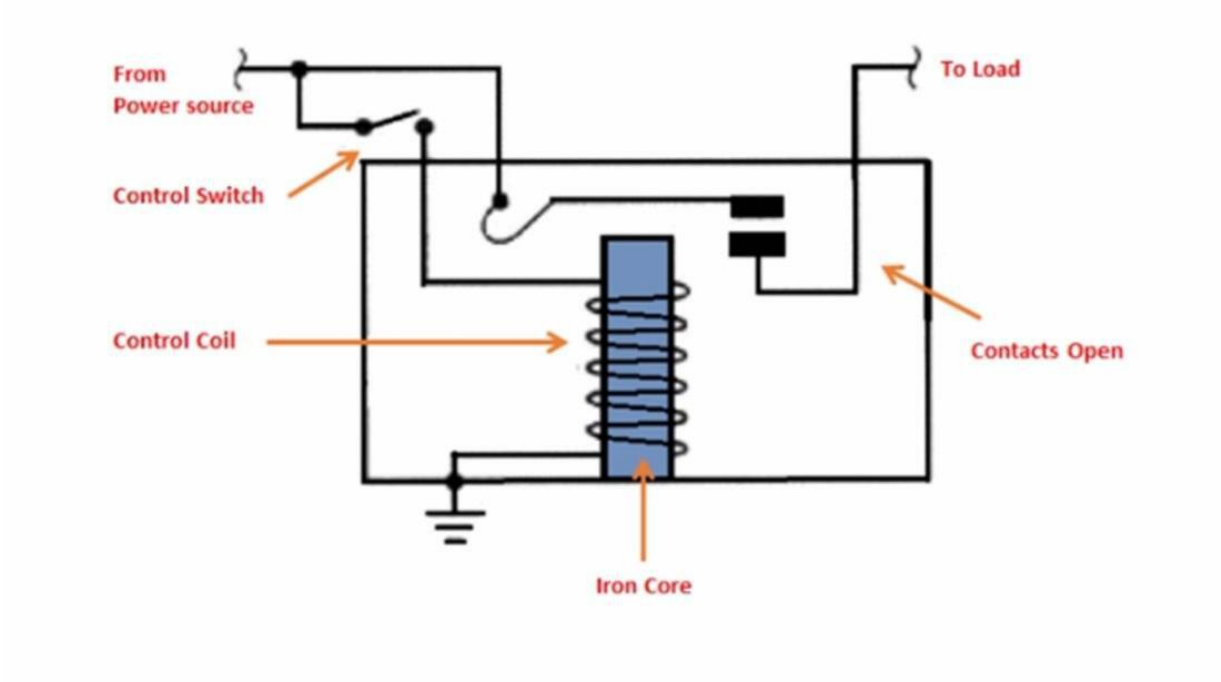
INTRODUCTION TO RELAY

As you already know that we are developing a project for controlling home appliances. So in this project relay plays an important role. So before going ahead with the testing of this device, let us gain some knowledge about it. A relay is an electrical device that is used to control the flow of current between two circuits by opening or closing a switch. It is essentially an electromechanical switch that is controlled by an electrical signal. Relays have two main parts, the coil and the contacts.



The coil is typically a wire wrapped around a core that produces a magnetic field when an electrical current is applied. The contacts are the switch that opens and closes based on the state of the coil. An electromechanical relay transfers signals between its contacts through mechanical movement. It has three sections i.e. input section, control section and output section. A simple relay is a two-way switch making the connection with a different circuit on either side.

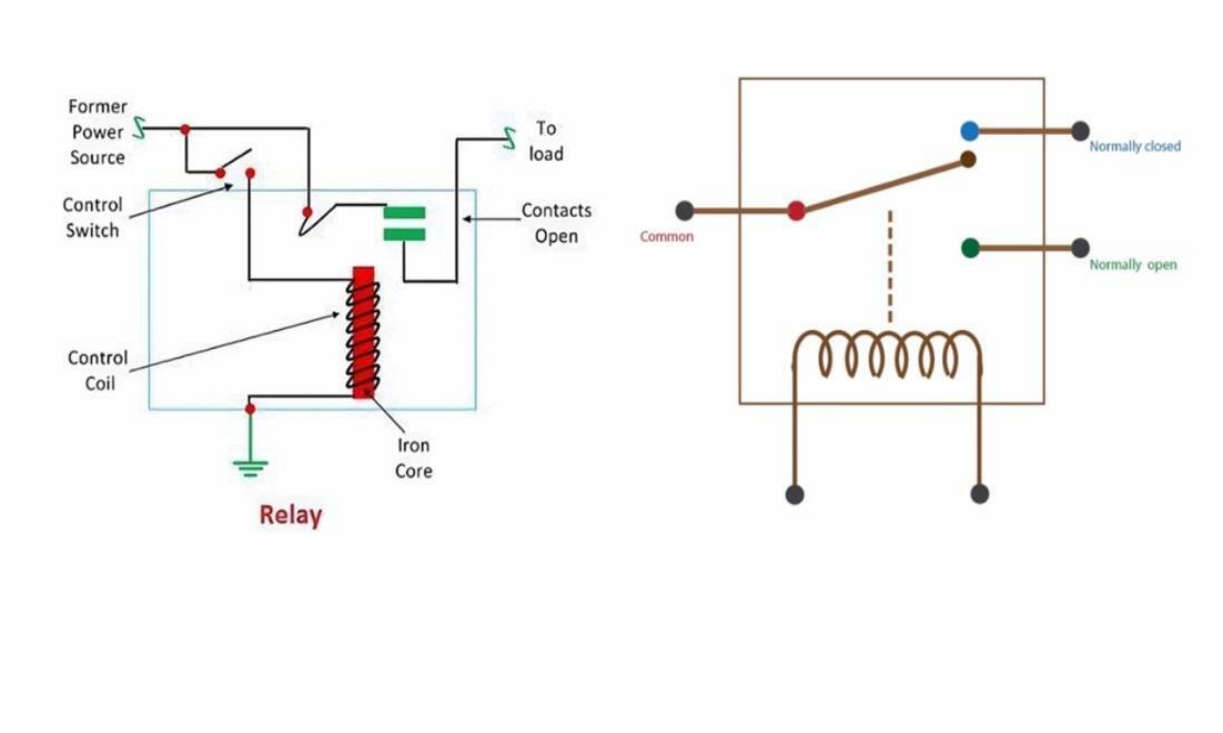
It has three contacts, normally closed, common and normally open, which are abbreviated as NC, COM and NO, respectively. Initially, when no power is supplied to the activated coil, contact is made between NC and COM terminals. If you connect a bulb with COM and NC, at that time it will glow. And also by the name, you can understand that normally closed means it is connected by default. Similarly, when the relay coil receives power, the COM inside the relay leaves the normally closed NC connection and makes contact with the normally open NO terminals.



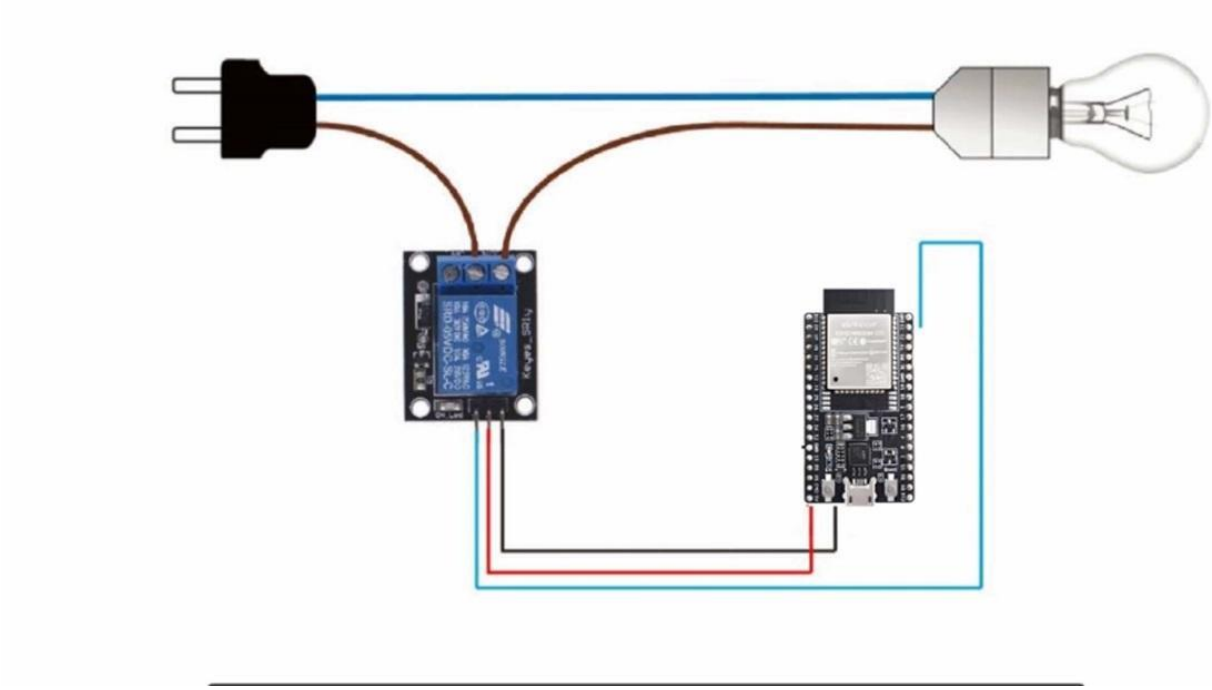
This allows current to flow through the NO terminal and power the bulb, causing it to light up. Relays are commonly used in applications where it is necessary to control a high current or voltage with a low current or voltage signal. Relays are often used in industrial control systems, home automation and automotive applications. For example, a relay can be used to turn on a light or a motor when a switch is pressed or to activate a solenoid valve to control the flow of a liquid or gas.

UNDERSTANDING THE CIRCUIT DIAGRAM TO TEST ONE INPUT OF 4 CHANNEL RELAY

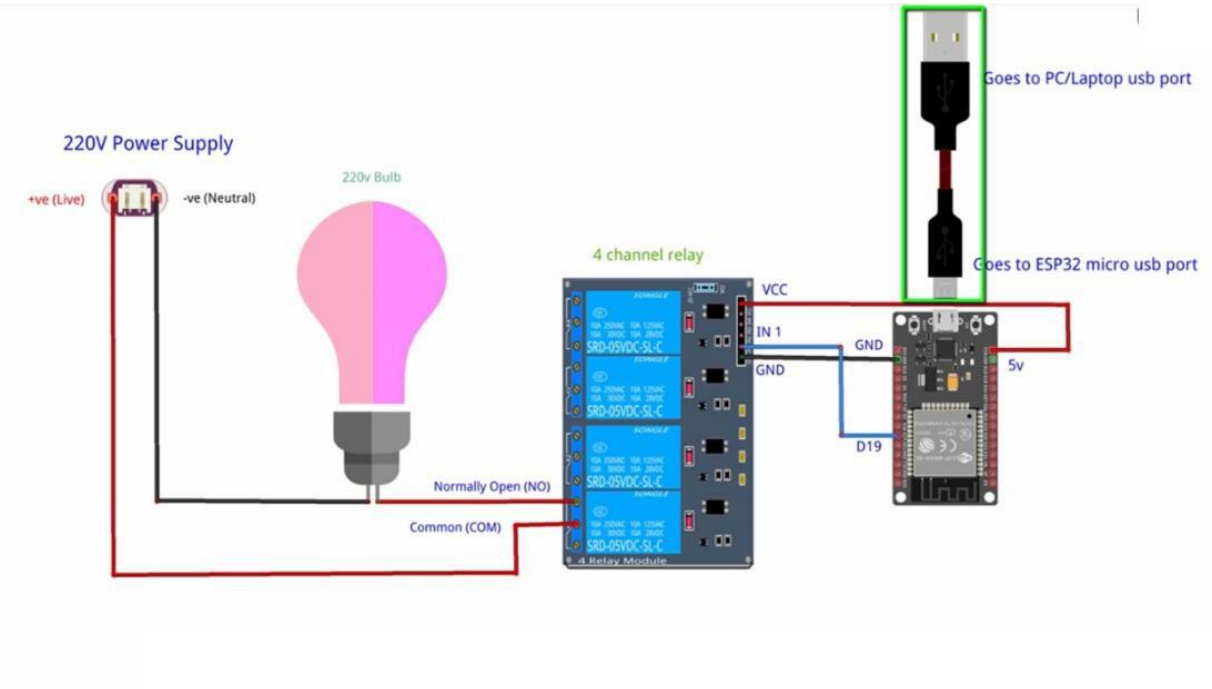
Now that we have covered the theory behind relays, it's time to put it into practice and test its functionality. This testing will serve two purposes. Firstly, it will help us to determine if the relay we purchased is functioning correctly or if we need to request a replacement from the e-commerce site. Secondly, by performing a basic test, we will gain a better understanding of how the relay operates. To test the relay, let us understand a circuit diagram that shows the connection of a single relay from a set of four relays.



Speaker 1 (00:01:02) - The purpose of demonstrating a single connection is to provide a practical understanding of how the relay operates. Once you understand the principle of this, programming the remaining relays will become a simple task. We are using a micro USB cable which will be used to power the ESP32 from the PC or laptop and along with that, it is also used for uploading the code into the ESP32 board.



The ESP32 is connected to the USB cable and 4-channel relay. The 5V pin of ESP32 is connected to the VCC pin of the relay.



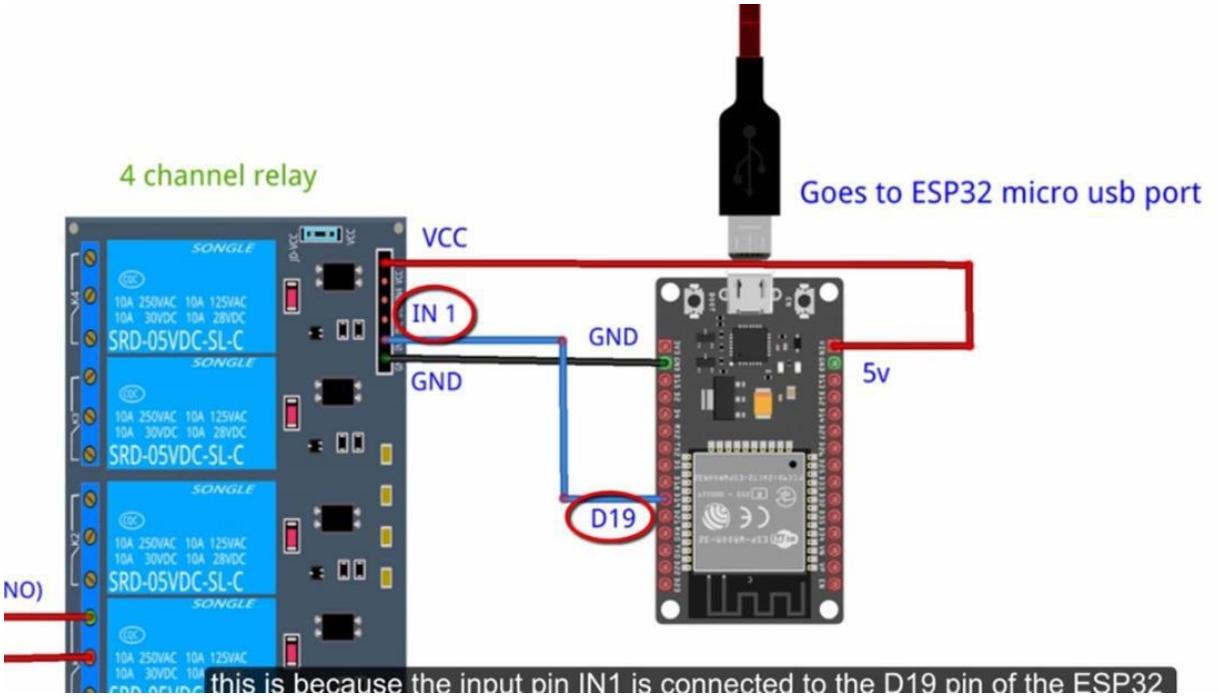
The GND of ESP32 is connected to the GND of the relay. Please note that the GND remains common throughout the board. You can

also share the GND present on the other side of the board. The D19 pin of ESP32 is connected to the data pin IN1 of the relay.

The IN1 represents the first relay that is present on this side and it goes on sequentially as IN2, IN3 and IN4. In the 4-channel relay module, we are using only one relay, where normally open is connected with one of the terminals of the 220V bulb and the common is connected with the positive live terminal of the 220V power supply. The negative neutral of the 220V power supply is connected to the other end of the 220V bulb. So, the relay is acting as a switch between the bulb and the power supply.

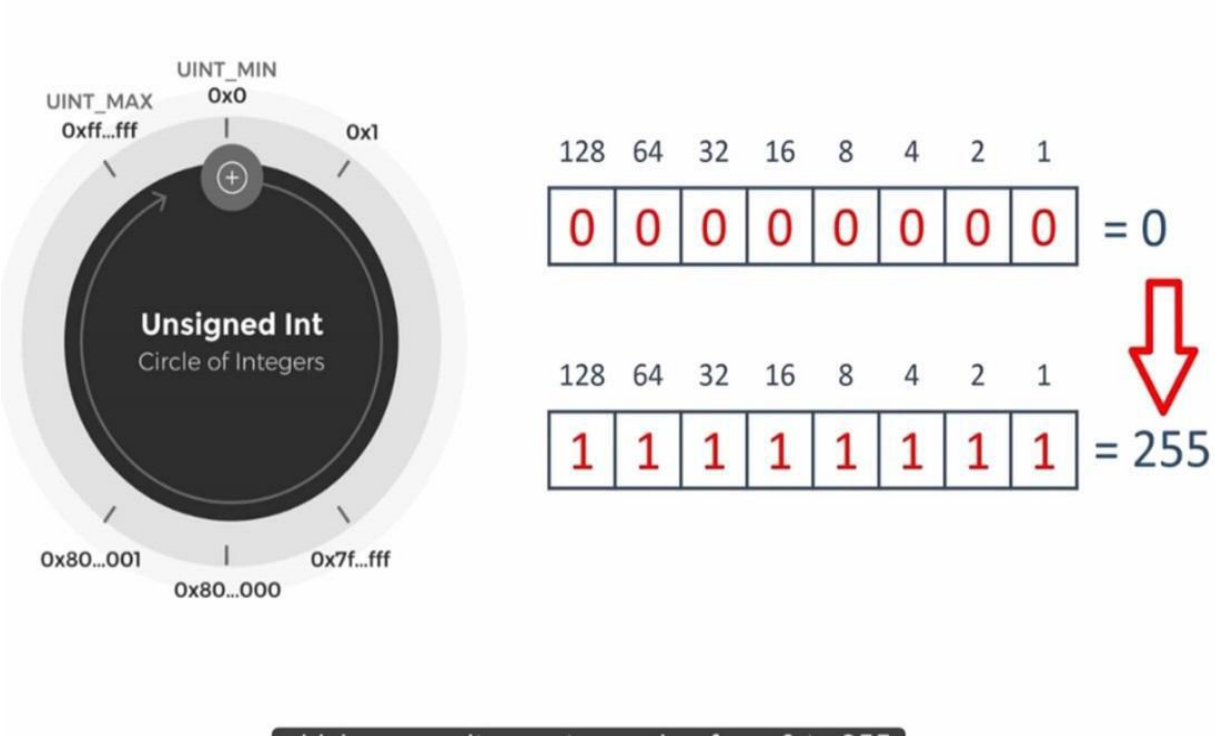
UNDERSTANDING THE CODE TO TEST ONE INPUT OF 4 CHANNEL RELAY

In the previous lecture, we discussed the connections required for the project. Now we will focus on the code which will make the entire circuit work. Once we have a clear understanding of this code, we will proceed with the deployment and observe its functioning. Here we are declaring a static variable named relay of type 8-bit unsigned integer that is initialized with the value of 19. This is because the input pin IN1 is connected to the D19 pin of the ESP32 board. You may be wondering why we are using this complex data type instead of a simple int.



However, there are specific reasons for this choice. Memory usage. It uses less memory than an int. In embedded systems with limited memory resources, this will be an important consideration. Data

range. It is an unsigned 8-bit integer data type which means it can store value from 0 to 255.



This can be useful in situations where you need to work with data that falls within this range. The operation of an 8-bit unsigned integer can be faster than int operations on some processors including the ESP32. Code clarity.

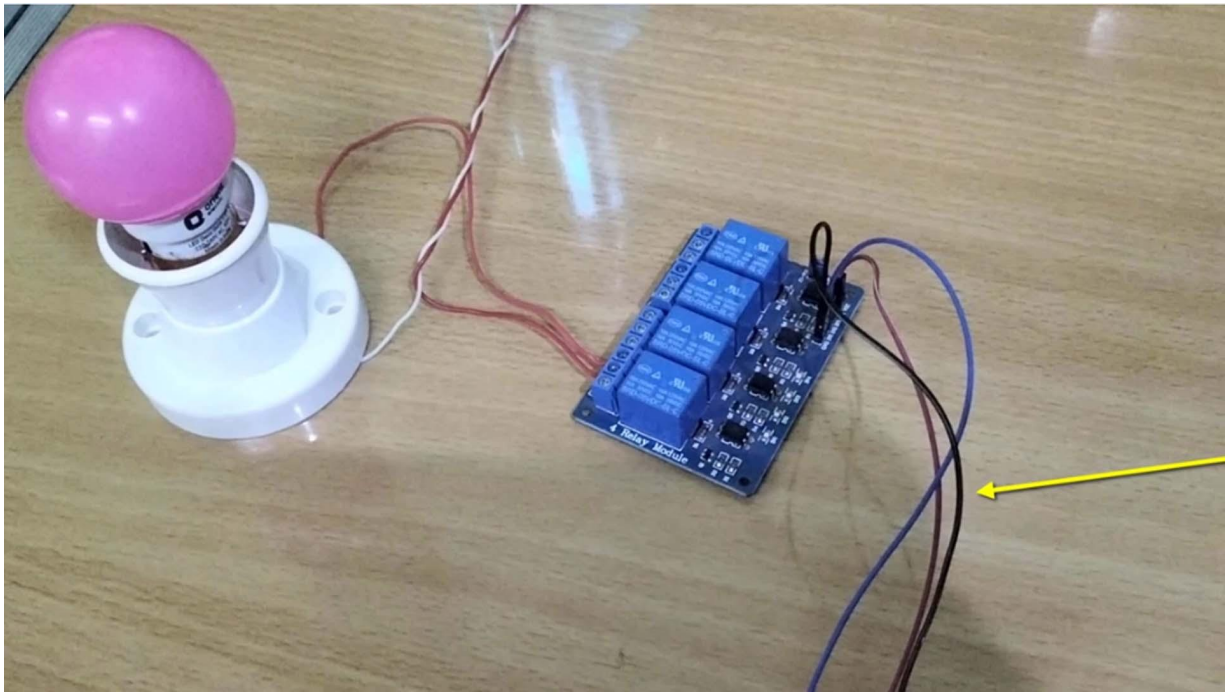
Using this, we can improve code clarity and reduce the risk of unintended consequences. By explicitly specifying the variable type and scope, it can be easier to understand and maintain the code over a period of time. We have already understood these lines of code in the previous lecture.

You can just notice that we have mentioned the relay as output because we are sending the signal to the relay to turn the device on or off. We are not reading any incoming signal from the relay. We have also understood the loop function in the previous lecture.

So, let us go ahead. Here we can observe that these lines of code are similar to the previous program to blink led with the same duration of 1000 milliseconds. Here we are displaying the message bulb is off and bulb is on one by one with one second of delay. So, this is the code for testing one input of the 4-channel relay with normally open connectivity. In the next lecture, we will execute and check the output of our program.

OUTPUT - TESTING ONE INPUT OF 4 CHANNEL RELAY

Before going ahead with the final output, let me first explain to you how I have made the connection. So this is the mains AC power supply of 220V via the board. The plug is connected to the 220V bulb as we have already shown in the circuit diagram. Also we can see that this is the USB cable connected to the USB port of the laptop. The other end of the USB cable is connected to our ESP32 board. The board is connected to the 4-channel relay using the jumper wires.



The black color of the wire is connected between ESP32 and the relay, which is a ground connection. The red color of the wire is connected from 5V of the ESP32 to the VCC of the relay. And the blue color of the wire is connected from input pin IN1 of the relay to the digital pin 19 of the ESP32. So let us see the output after uploading the program into the ESP32 board. Please pay attention

here. Sometimes after opening the Arduino IDE, it will ask to update the packages or boards.

You have the option to choose whether you want to update one library or more or you can skip the update. Here we strongly recommend that you don't click on install or choose any update message because it will update the ESP32 board. And then the program will not execute properly and you may encounter a lot of errors. To prevent any such issues from occurring, avoid clicking on any update prompts that may appear. In the system, we can see this is the source code of testing one input of 4-channel relay using normally open connectivity.

Before uploading the code, first compile it to make sure that there are no errors in our program. When the message done compiling appears, it means that we have no errors in our code. So let's go further and upload our program. Now upload the code into our ESP32 board. But please make sure that all the components are connected properly as mentioned in the entire circuit diagram.

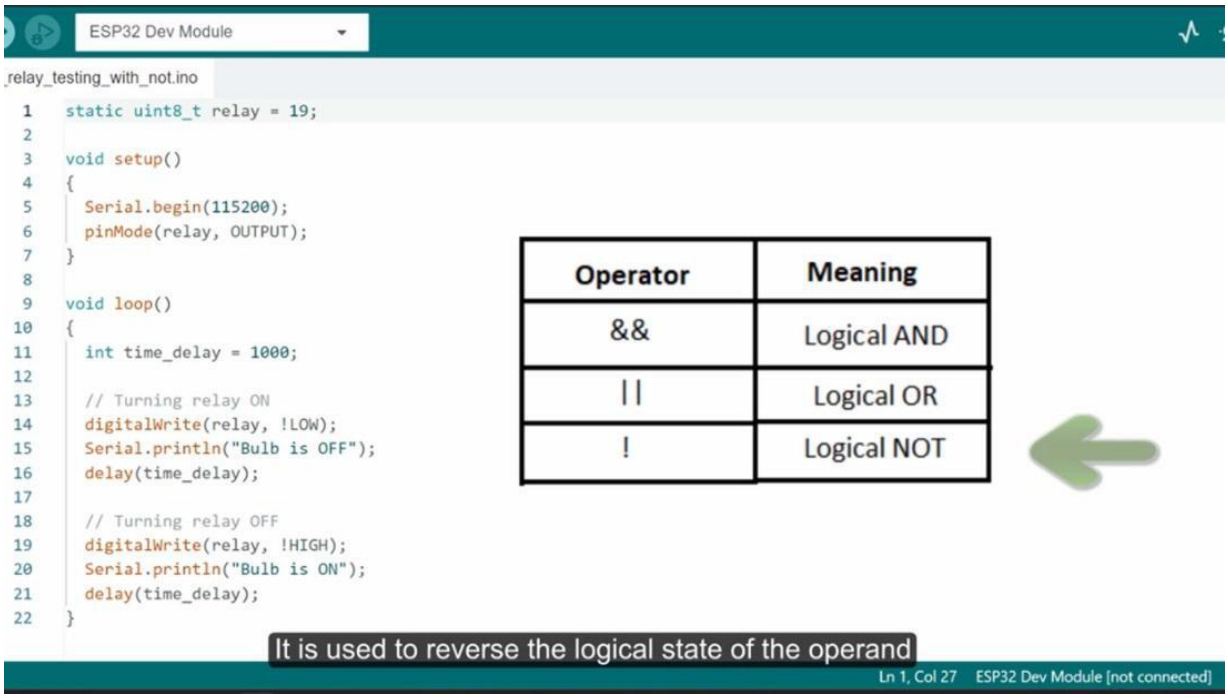
After clicking on the upload button, it will start uploading code inside the board and when it will show 100% completion, then go to the tools menu and select the serial monitor.

This will open a new console window where you can see all the outputs of the program. Here we can see it is showing the bulb is off and the bulb is on one by one. Apart from that, on the hardware side, the bulb is also turning on and off as per the delay specified in the program. But if you observe closely, you will realize that it is not working as per the program.

Speaker 1 (00:03:52) - The message is showing in the opposite way. It means when the bulb is on, the printing bulb is off and when the bulb is off, the printing bulb is on. Let us see how we can fix this problem in the next lecture.

RESOLVING THE INVERSE OPERATION OF THE RELAY

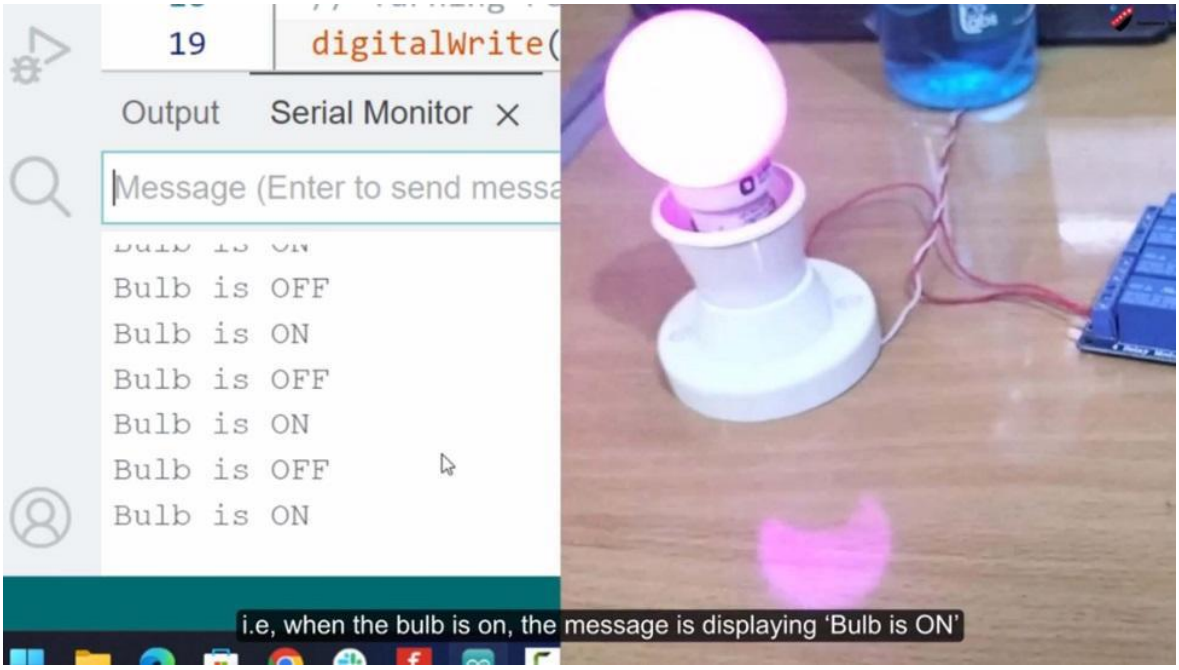
Let us now explore how we can fix the relay that is currently operating in the opposite direction. We can solve this problem in two ways. Method 1 The first method is quite simple. You don't have to make any changes on the hardware side. We will simply modify our code by adding a logical NOT operator in our code. It is used to reverse the logical state of the operand and is denoted by the symbol !. Go back to your program and here we are changing the value inside the digitalWrite function using an !.



So, adding a logical NOT in front of low will give the value high which means it will reverse the output of the bulb. So, the bulb will glow instead of turning off. Similarly, we will change here also. We will prefix the exclamation mark with high. This will turn off the bulb instead of turning it on. We are done. We don't need to make any

more changes to the code. Now we can compile the program and when the compilation is done, click on the upload button.

After 100% completion of the upload, go to the serial monitor and you can see that this time the output is showing correctly i.e. when the bulb is on, the... and when the bulb is off, the message is displaying the bulb is off. Method 2 Now let us explore the second method which involves a hardware change.

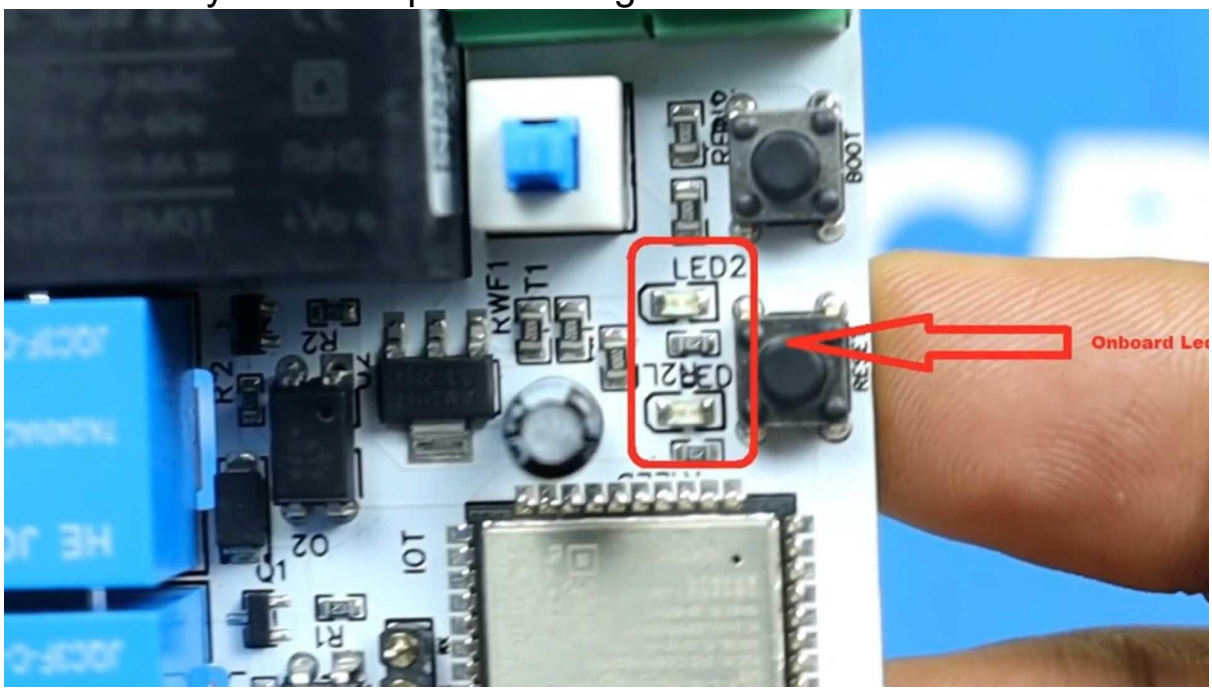


This approach doesn't require any modifications to the code. Instead, we need to revisit the relay connection point. In our previous setup, the second terminal of the bulb was connected to the normally open pin of the relay.

To implement this method, we need to disconnect the connection and shift it to the normally closed pin instead. It is crucial to ensure that the mains power supply is disconnected while making this change for safety purposes. Once the modification is done, you can upload the previous code and you will observe that it is functioning correctly with the updated relay configuration.

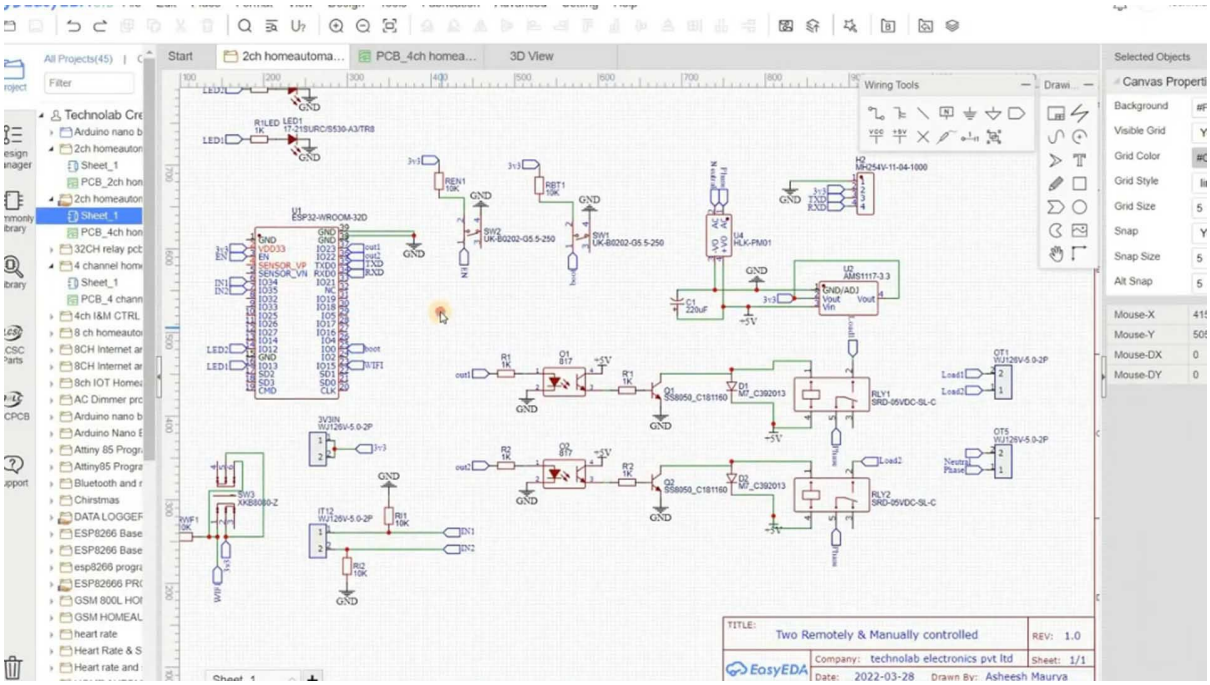
2 NODE SMT SMART HOME-AUTOMATION PCB

Hello, welcome back to another project. Now in this project, I am going to introduce my newly designed fully SMT component using 2-channel remotely and manually controlled home automation PCB. I have also made an 8-channel SMT home automation PCB, a 4-channel SMT Home automation PCB, and also a TMT component used home automation PCB. Now in this project, I will show you a 2-channel fully SMT component using another home automation PCB.

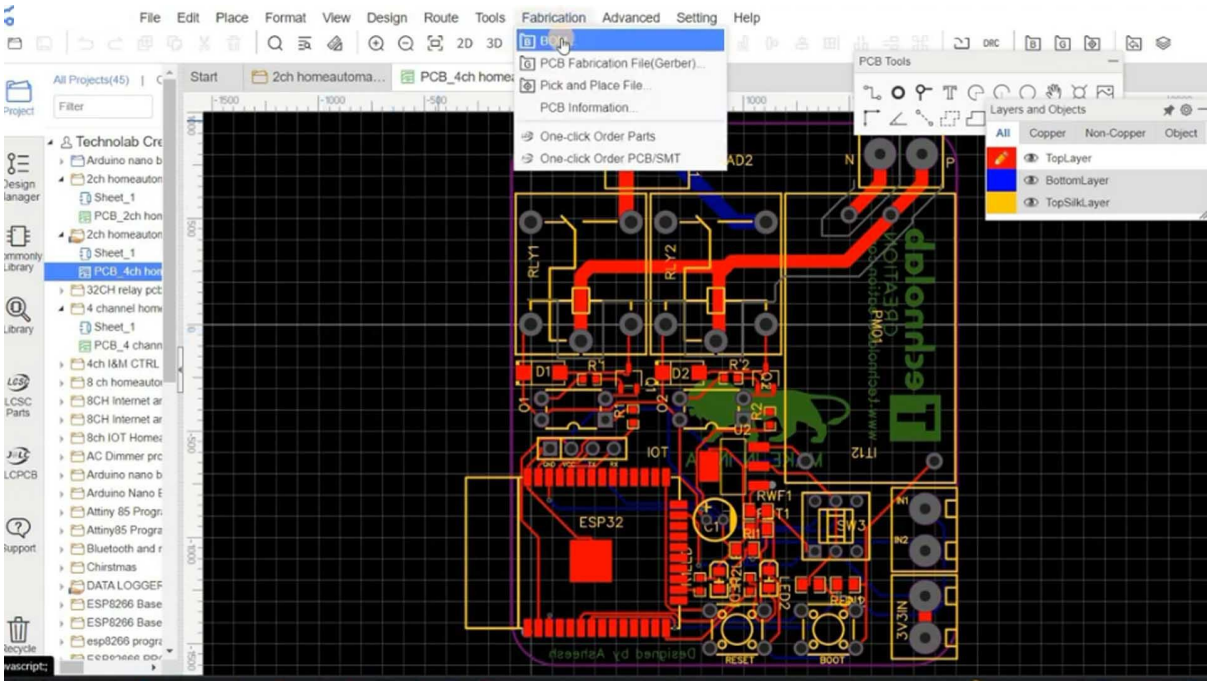


This home automation PCB is very small in size and compact, which can easily fit in your electrical switchboards. This PCB has an inbuilt OTA button, that is, you can update the code wirelessly over the air. There are 2 onboard LEDs which you could use according to your need, like testing your code or something else. In my case, I have used it as a Wi-Fi indicator, that is, if a Wi-Fi connection is available, then both the LEDs will glow as only a single LED will glow. By

using this PCB, I will make a small home automation system that is an Internet and manually controlled home automation system using the BlinkIoT cloud platform. In this home automation system, we can control our home appliances using the Blink smartphone application from anywhere in this world. Apart from this, we can also control our home appliances through manual switch buttons that we generally use in our homes, and we can also monitor the real-time status on the Blink smartphone. Also, this PCB is compatible with all the smart speakers available in the market like Amazon Alexa, Google Home, and Apple series. During the project, I will explain the circuitry of this home automation system, how we can connect our home appliances to this PCB, and also I will let you know how to flash the code. So let's get into this project. JLC PCB is a fast electronic manufacturing company. JLC PCB SMT services fulfill customers' money and time-saving needs. Customers enjoy low-cost, high-quality, and fast assembly services at $8 setup fees. At the same time, they assemble electronic products from PCB design to PCB assembly production on the same online platform. JLC PCB provides a one-stop service from PCB design and PCB prototype to PCB assembly. And you can track their electronic manufacturing process in real-time. JLC PCB takes 24-hour SMT build time and provides the fastest delivery. Get your PCB assembly product in one week from ordering, power sourcing, assembly, and PCB assembly prototyping. Thousands of components supported by JLC PCB and its reliable component partners like Digikey and Mouser 2000. In-stock components available in the JLC PCB SMT parts library. This benefits customers to source components much faster and easier, bringing you a shorter PCB assembly production time. JLC PCB provides free PCB design software, cheap PCB prototypes, fast and low-cost SMT services, saving your electronic product cost. With 16 years of PCB manufacturing experience, JLC PCB also establishes a well-trended engineer and customer support team devoted to ensuring PCB assembly services are faster and cheaper.



To ensure assembly of your PCB in the best service, this is the schematic of today's home automation PCB. If you want, you can download this schematic from the description to design your own custom PCB. Now convert this schematic into a PCB. After completing the design of your PCB, you can directly order the PCB from JLC PCB or just download the Gerber file from here. After that, go to the JLC PCB website, then click on the Quote Now button under the summit assembly. After that, upload the Gerber file of your PCB. After that, select the number of PCBs and color masking of PCB if you want. After that, select the summit assembly service, and here you have to select on which surface you want your components to be soldered, either top surface or bottom. After that, click on the confirm button.



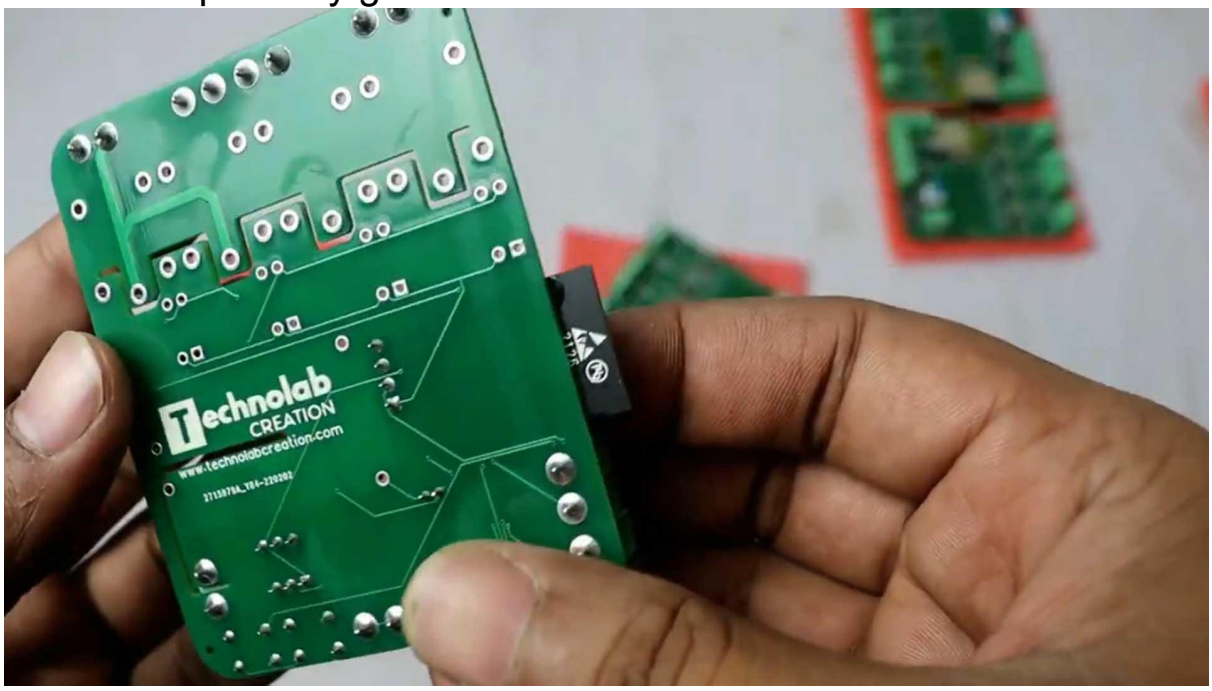
Now here you have to upload two more files. One is the CPL, that is the pick and place file, and another one is BOM, that is the Bill of Materials. You can download these files from your EasyEd account. Just open that PCB project on your EasyEd account and then click on Fabrication, then BOM. Now click on Export. Now click on Export BOM to download the BOM file. Similarly, download the CPL file. After downloading both the files, just upload both the files here onto this page. After that, select next. Here, it will show all the SMT components which are to be soldered, and also we can select which components will be soldered or not. Select the components according to your preference. After that, click on the next button, then click save to cart to complete your order. After a week, my PCB arrived at my place in a new blue box from JLC PCB. Let me open the box. The packaging of the PCB in bubble wrap is very good. Here it is, our home automation PCB. The quality of the PCB is good, and the SMT components are soldered well after soldering. The rest of the components PCB looks like this. Neat, clean, and well-arranged. To flash the code into the ESP32 chip, I will use an ESP32 Development Board. Now make the connections according to this schematic.

MANUAL CONTROL HOME-AUTOMATION SYSTEM USING ESP32

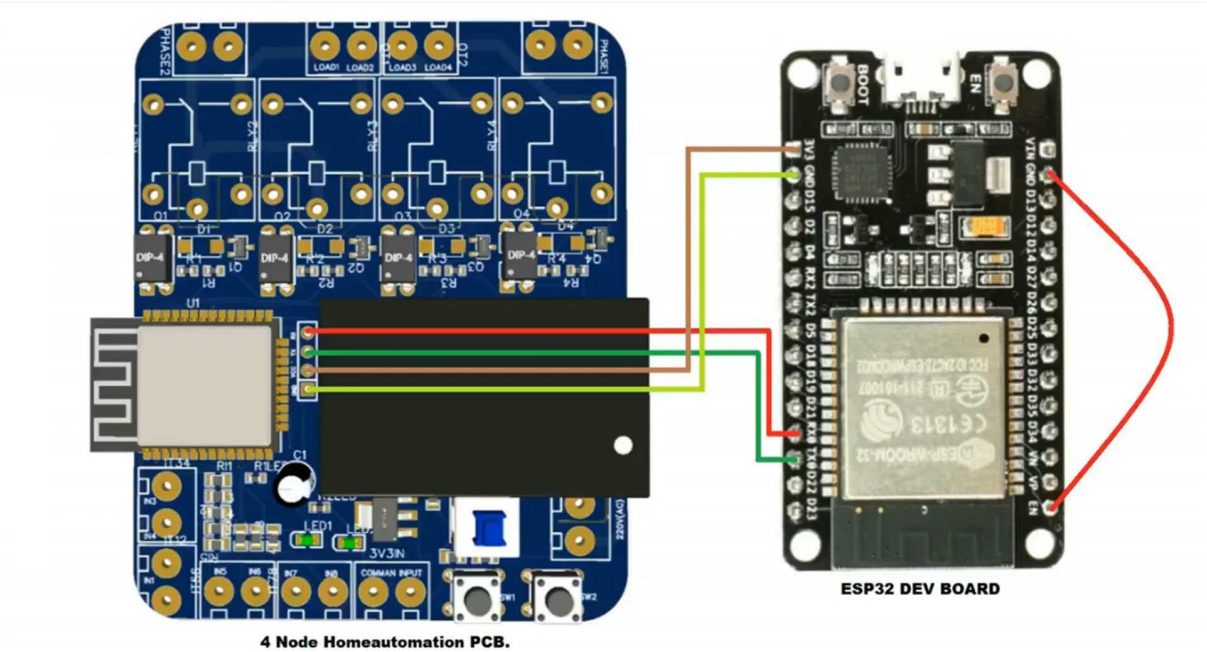
Now in this project, we are going to make an Android app-controlled home automation system using ESP 32's inbuilt Bluetooth feature. Along with this, we can also control our home appliances through regular switch buttons that are generally used in our home. Our Android application will communicate with the ESP 32 via Bluetooth. Hence, no Wi-Fi is required. For the making of this project, I will use my newly designed 4-node SMT home automation PCB. Now let's get into this project. This project is sponsored by JLC PCB. JLC PCB is a well-known PCB prototype company in China. It specializes in quick PCB prototypes and small-batch production. You can now order a minimum of five PCBs for just $2. For more details, check the description. This is the schematic of today's home automation PCB. If you want, you can download this schematic from the description to design your own custom PCB. Now convert this schematic into a PCB.

After completing the design of your PCB, you can directly order the PCB from JLC PCB or just download the Gerber file from here. After that, go to the JLC PCB website, then click on the Quote Now button under the summit assembly. After that, upload the Gerber file of your PCB. After that, select the number of PCBs and color masking of PCB if you want. After that, select the summit assembly service, and here you have to select on which surface you want your components to be soldered, either top surface or bottom. After that, click on the confirm button. Now here you have to upload two more files. One is the CPL, that is the pick and place file, and another one is BOM, that is the Bill of Materials. You can download these files from your EasyEDA account. Just open that PCB project on your EasyEDA account and then click on Fabrication, then BOM. Now click on Export. Now click on Export BOM to download the BOM file. Similarly, download the CPL file. After downloading both the files, just upload both the files here onto this page. After that, select next. Here it will show all the summit components which are to be soldered, and also we can select which components will be soldered or not. Select the components according to your preference. After that, click on the next button, then click save to cart to complete your order. After a week, my PCB arrived at my place in a new blue box from JLC PCB. Let me open the box. The packaging of the PCB in

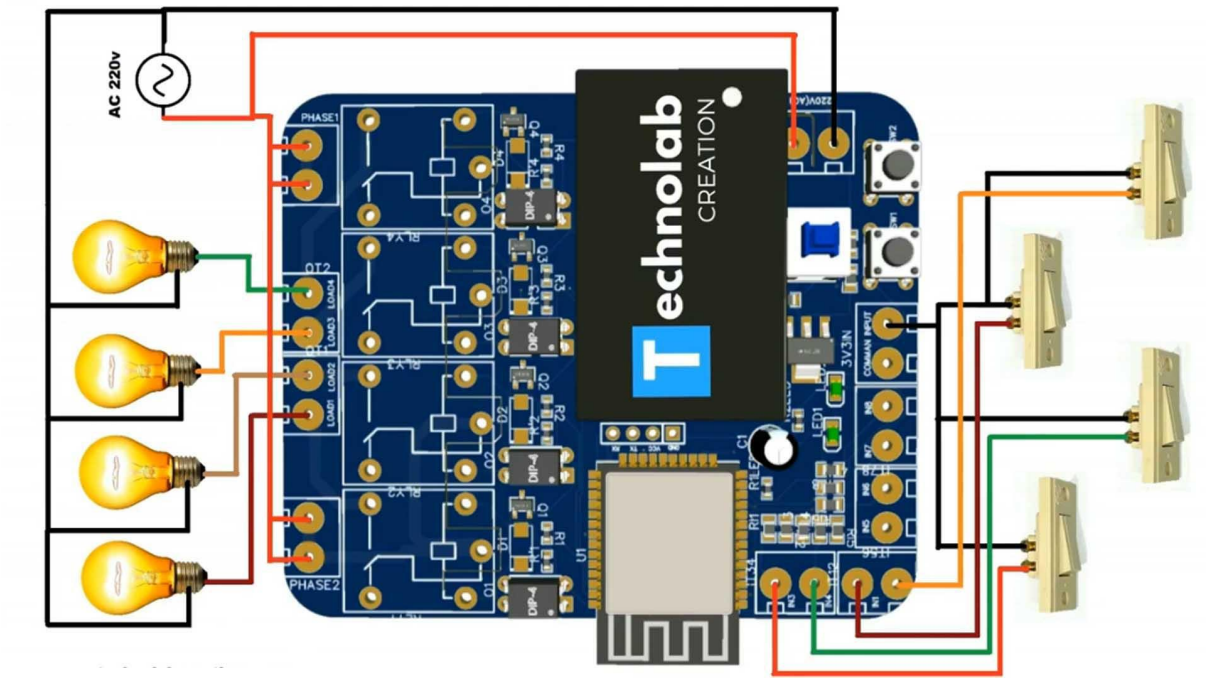
bubble wrap is very good.



Here it is, our home automation PCB. The quality of the PCB is good, and the summit components are soldered well after soldering. The rest of the components of the PCB look like this. Neat, clean, and well-arranged. To flash the code into the ESP 32 chip, I will use an ESP 32 Development Board. Now make the connections according to this schematic. This is the code for our today's home automation project. You can download this code from the link given in the description here. In this section of the code, you have to enter the name of your Bluetooth device.



This Bluetooth device name will appear when we pair our smartphone with ESP 32. Now this code is ready. Hit the upload button after selecting the right board and comport. After clicking the upload button onto the PCB, I will press and hold the boot button and press the reset button once to make this module go into the boot mode. As you can see, the code starts uploading. Download the APK file of this mobile application from the given link in the description and install it on your smartphone. Now open the Bluetooth settings of your phone and click on a new device.



Here, the Bluetooth device with the same name that we gave in the code. Now tap on this pair. Now tap on this and pair your phone with the ESP 32 after successful pairing of the Bluetooth device. Now open the app again and tap on connect. Here, select ESP 32. The Bluetooth is connected message is shown, which means the ESP 32 is successfully connected to the app. Now connect all the bulbs and switches according to this circuit diagram.

I'm trying. To think. About it but I. Would like to. To.

EXAMPLE DUMMY CODE

A home automation system using an ESP32 can be quite complex, as it involves various sensors, actuators, and communication protocols. However, I can provide a basic example to get you started with controlling a single device (e.g., a light) via a web interface. In this example, we'll use the ESP32, a relay module to control a light, and the Arduino IDE. Additionally, you can enhance the system by adding more devices, sensors, and features.

Here's a simple code example for controlling a light via a web interface using the ESP32:

```cpp
#include <WiFi.h>
#include <ESPAsyncWebServer.h>
#include <ArduinoOTA.h>

// Replace with your network credentials
const char* ssid = "Your_SSID";
const char* password = "Your_PASSWORD";

const int relayPin = 2; // Pin connected to the relay module (for controlling the light)

AsyncWebServer server(80);

void setup() {
  Serial.begin(115200);

  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, LOW); // Initialize the relay as OFF
```

```
  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");

  // OTA (Over-the-Air) update setup
  ArduinoOTA.onStart([]() {
    Serial.println("OTA update started");
  });
  ArduinoOTA.onEnd([]() {
    Serial.println("\nOTA update finished");
  });
  ArduinoOTA.onError([](ota_error_t error) {
    Serial.printf("OTA update error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) Serial.println("Auth
Failed");
    else if (error == OTA_BEGIN_ERROR)
Serial.println("Begin Failed");
    else if (error == OTA_CONNECT_ERROR)
Serial.println("Connect Failed");
    else if (error == OTA_RECEIVE_ERROR)
Serial.println("Receive Failed");
    else if (error == OTA_END_ERROR) Serial.println("End
Failed");
  });
  ArduinoOTA.begin();

  // Web server setup
  server.on("/", HTTP_GET, [](AsyncWebServerRequest
*request){
```

```
    String state = (digitalRead(relayPin) == HIGH) ? "ON" :
"OFF";
    String html = "<html><body>";
    html += "<h1>Light Control</h1>";
    html += "<p>Light is currently " + state + "</p>";
    html += "<a href=\"/toggle\">Toggle Light</a>";
    html += "</body></html>";
    request->send(200, "text/html", html);
  });

  server.on("/toggle", HTTP_GET, [](AsyncWebServerRequest
*request){
    digitalWrite(relayPin, !digitalRead(relayPin));
    request->redirect("/");
  });

  server.begin();
}

void loop() {
  ArduinoOTA.handle();
}
```

This code sets up a web server on your ESP32 that allows you to control a light. You can access the ESP32's web interface by navigating to its IP address. The interface will display the current state of the light and allow you to toggle it.

Make sure to replace `"Your_SSID"` and `"Your_PASSWORD"` with your Wi-Fi credentials.

To upload the code to your ESP32, follow the instructions provided in the previous response about setting up the ESP32 in the Arduino IDE.

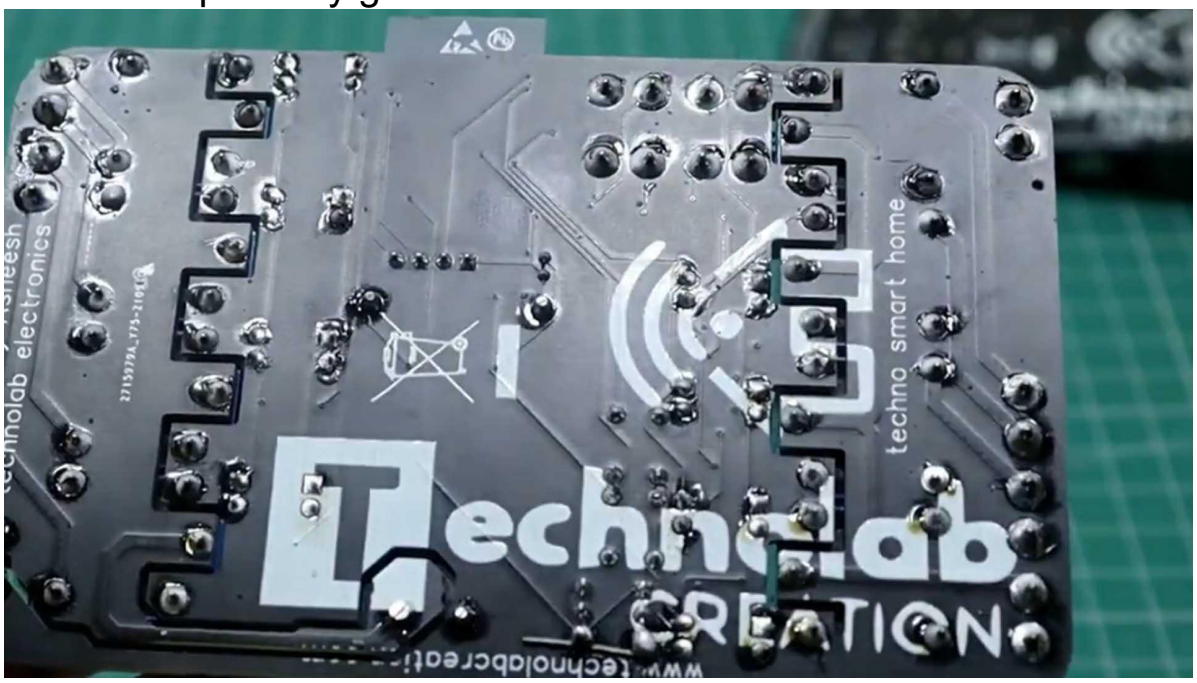
Once the code is uploaded, open the Serial Monitor to find the ESP32's IP address. Use a web browser to access the IP address, and you'll be able to control the light via the web interface.

8 NODE SMT SMART HOME-AUTOMATION PCB

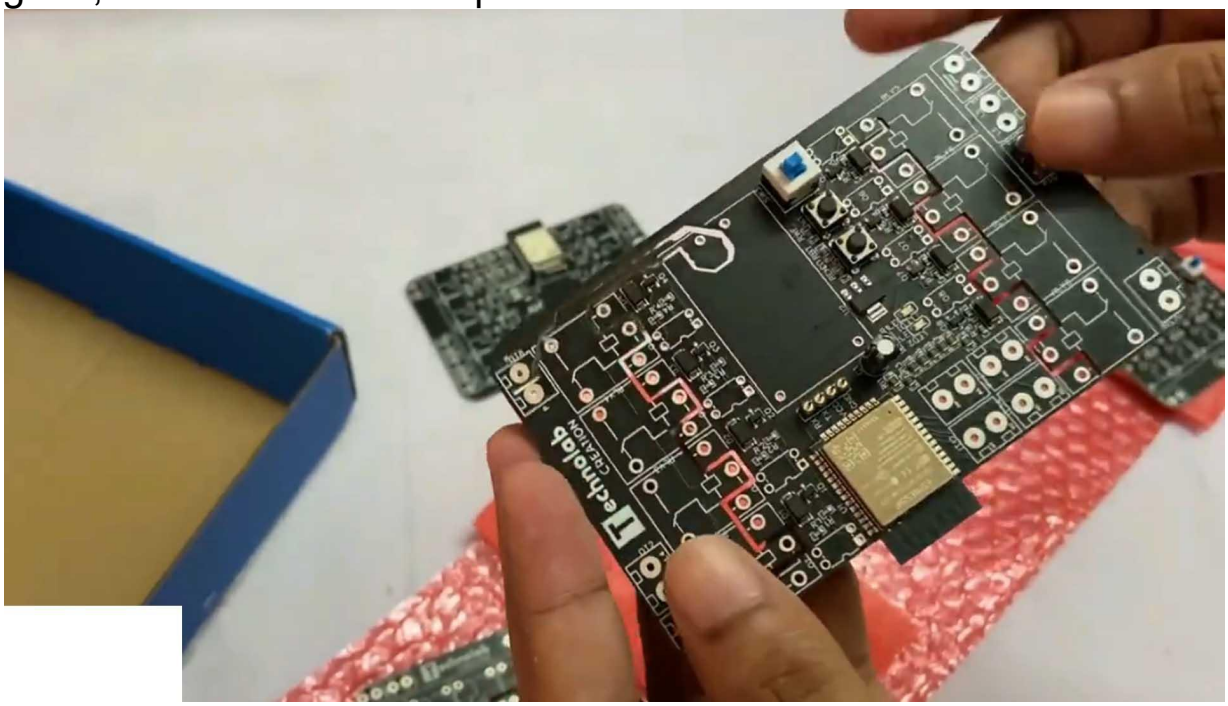
Hey, hello friends, welcome to another project. In this project, we are going to make a home automation project using the new Blink 2.0 and an eight-node SMT home automation PCB. In this home automation project, we can control our home appliances via the Blink smartphone application, Blink dashboard, from anywhere in this world. We can also control our home appliances via manual switch buttons and monitor the real-time status in the Blink app. During the project, I will explain the circuitry, code, and how to set up the Blink dashboard.

Now let's get into this project. This is the schematic of today's home automation PCB. If you want, you can download this schematic from the description to design your own custom PCB. Now convert this schematic into a PCB. After completing the design of your PCB, you can directly order the PCB from JLCPCB or just

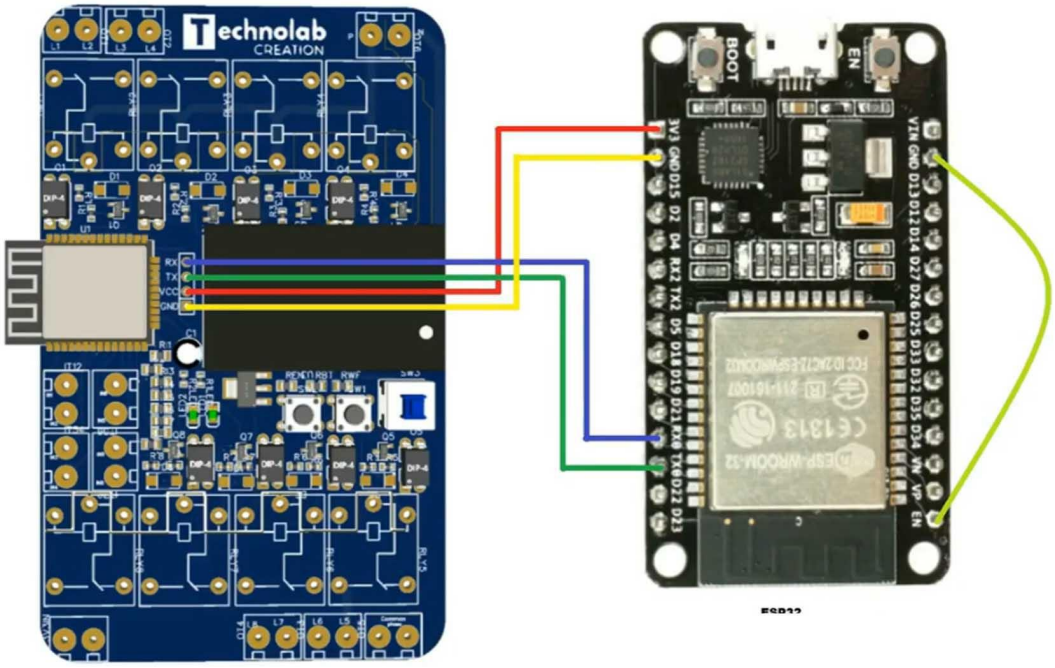
download the Gerber file from here. After that, go to the JLCPCB website. Then click on the Quote Now button under the summit assembly. After that, upload the Gerber file of your PCB. After that, select the number of PCBs and color masking of the PCB if you want. After that, select the summit assembly service, and here you have to select on which surface you want your components to be soldered, either the top surface or bottom. After that, click on the Confirm button. Now here you have to upload two more files. One is the CPL, that is the pick and place file, and another one is BOM, that is the Bill of Material. You can download these files from your EasyEDA account. Just open that PCB project on your EasyEDA account and then click on Fabrication, then BOM. Now click on Export. Now click on Export BOM to download the BOM file. Similarly, download the CPL file. After downloading both the files, just upload both the files here onto this page. After that, select next. Now here it will show all the summit components which are to be soldered and also we can select which components will be soldered or not. Select the components according to your preference. After that, click on the next button. Then click Save to cart to complete your order. After a week, my PCB arrived at my place in a new blue box of JLCPCB. Let me open the box. The packaging of the PCB in bubble wrap is very good.



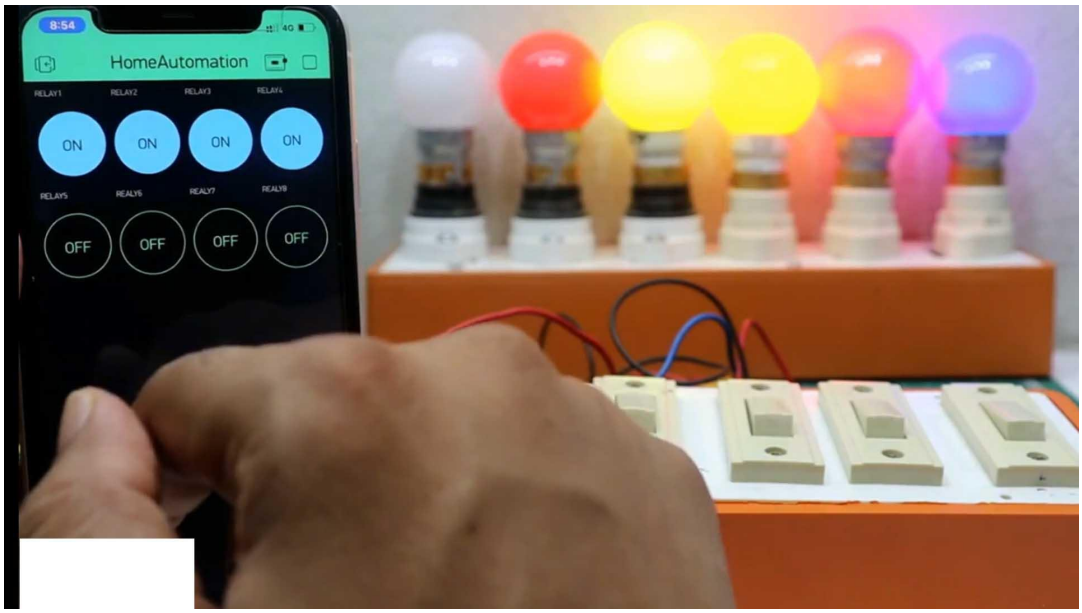
Here it is, our home automation PCB. The quality of the PCB is good, and the summit components are soldered well.



After soldering, the rest of the components on the PCB look like this. Neat, clean, and well-arranged. Open this Blink Cloud webpage from the link given in the project description, and here you have to log in with your email ID and password. In case you are new, just create a new account. It will take a few seconds. I already have an account on Blink, so I am directly logging in. Now click on a new template. Now give the name of your template on which your project is based. I am naming it "8 Node Blink". Then select the hardware type, in my case, it is ESP 32, then connection type, and the connection type is Wi-Fi. After that, click on the done button. Here we have successfully created the template for our project. Now click on data streams.



Then click on new data streams, select Virtual pin. Here you have to give the name of the data stream. Give any conventional name you want. Now select the pin on which you want to control your relay. I am selecting virtual pin V0, and here select data type as integer. Now click on create. Here we have successfully created the first data stream. Again, click on the new data stream, select Virtual pin. Now give the name of the second data stream. Select the pin for the second data stream. I am selecting virtual pin V1, and here again select data type as integer, then click on create button below. Here we have successfully created two data streams.



Now, in a similar manner, create six more data streams as we need to create a total of 8 relays in our project. Here we have successfully created the total eight data streams. Now go to the web dashboard to configure the web dashboard. As we need to control a total of 8 devices, we need a total of 8 switches for this. Drag and drop 8 switches from the widget box, one by one. Now, if you hover on the Switch widget, you'll find a setting icon. Click on this setting icon to set up the switch widget for the web dashboard. Now again, give here any name for this switch widget. Select the data stream for this switch widget and enable the show on/off labels. After that, click on the save button. In a similar way, set up all the rest of the switch widgets.

Designing a complete 8-node Smart Home Automation PCB is a complex task that involves hardware design, component selection, and embedded programming. While I can provide a basic outline of the components you may need, a full design would require detailed knowledge of your specific requirements and constraints. Here's a simplified example outline for an 8-node smart home automation PCB using a microcontroller like the ESP32:

****Components:****

1. **Microcontroller:** ESP32 is a popular choice due to its Wi-Fi and Bluetooth capabilities.

2. **Relays:** Use relay modules for controlling appliances or devices.

3. **Sensors:** Depending on your requirements, you may need various sensors such as PIR motion sensors, temperature and humidity sensors (DHT22), light sensors, etc.

4. **Power Supply:** Ensure a reliable power supply for the PCB, considering the power requirements of the components.

5. **Connectors:** Terminal blocks or connectors for easy wiring of devices.

6. **LED Indicators:** Indicators to show the status of each node.

7. **Wi-Fi/Bluetooth Antenna:** To ensure good connectivity for wireless communication.

8. **Voltage Regulators:** To regulate power for different components.

PCB Layout:

- Plan the layout of your PCB, considering the placement of components, traces, and spacing between them.

- Pay attention to thermal management, especially if relays or other components generate heat.

- Ensure proper grounding and power planes.

Circuit Design:

- Create the circuit diagram using a tool like KiCad or Eagle.

- Connect the microcontroller to the relays, sensors, and other components.

- Design power distribution and regulation circuits.

- Include pull-up/pull-down resistors where needed.

Embedded Software:

- Develop firmware for the ESP32 using the Arduino IDE or the ESP-IDF (Espressif IoT Development Framework).

- Implement Wi-Fi or Bluetooth communication for remote control.

- Write code to read data from sensors and control relays accordingly.

- Implement a user-friendly interface, such as a web server, to control the devices.

Testing:

- Rigorously test your PCB to ensure that all components work as expected.

- Verify the reliability and safety of the system.

Production:

- Once your prototype works as expected, you can proceed to mass production if needed.

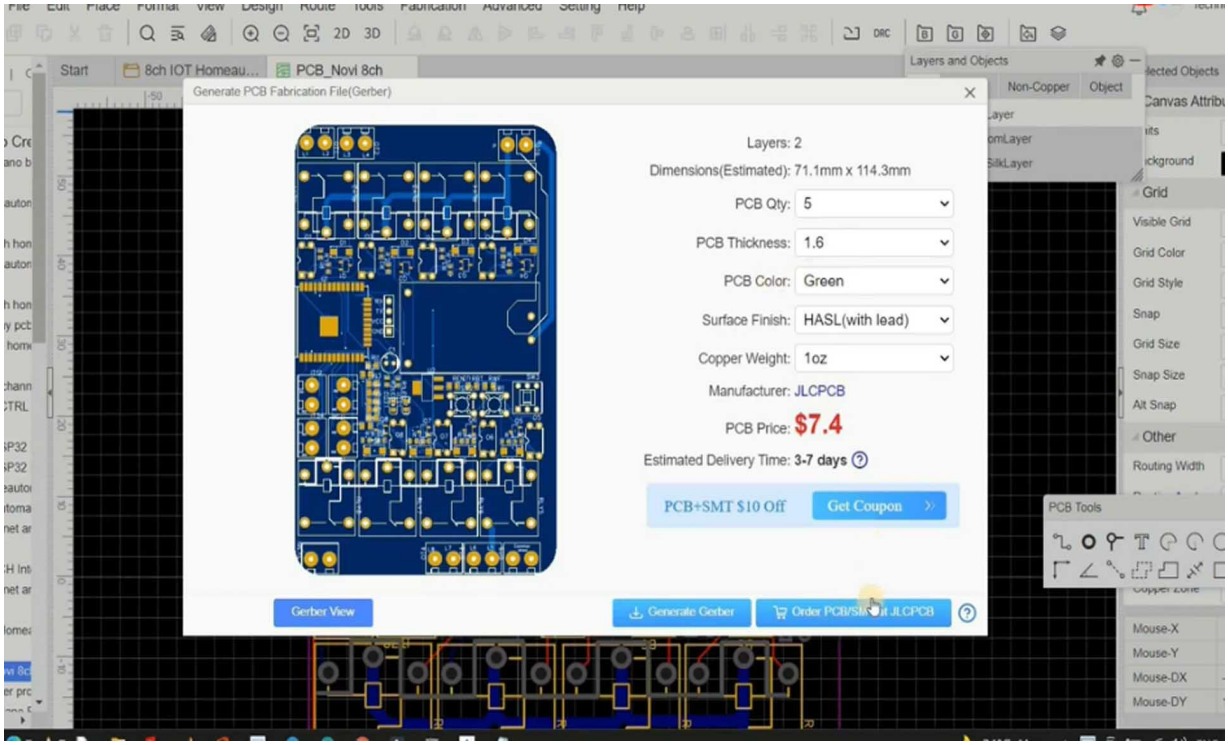
This is a high-level outline to get you started, and creating a full 8-node Smart Home Automation PCB is a significant project. It's crucial to consider safety and

security aspects, as well as relevant industry standards if applicable.

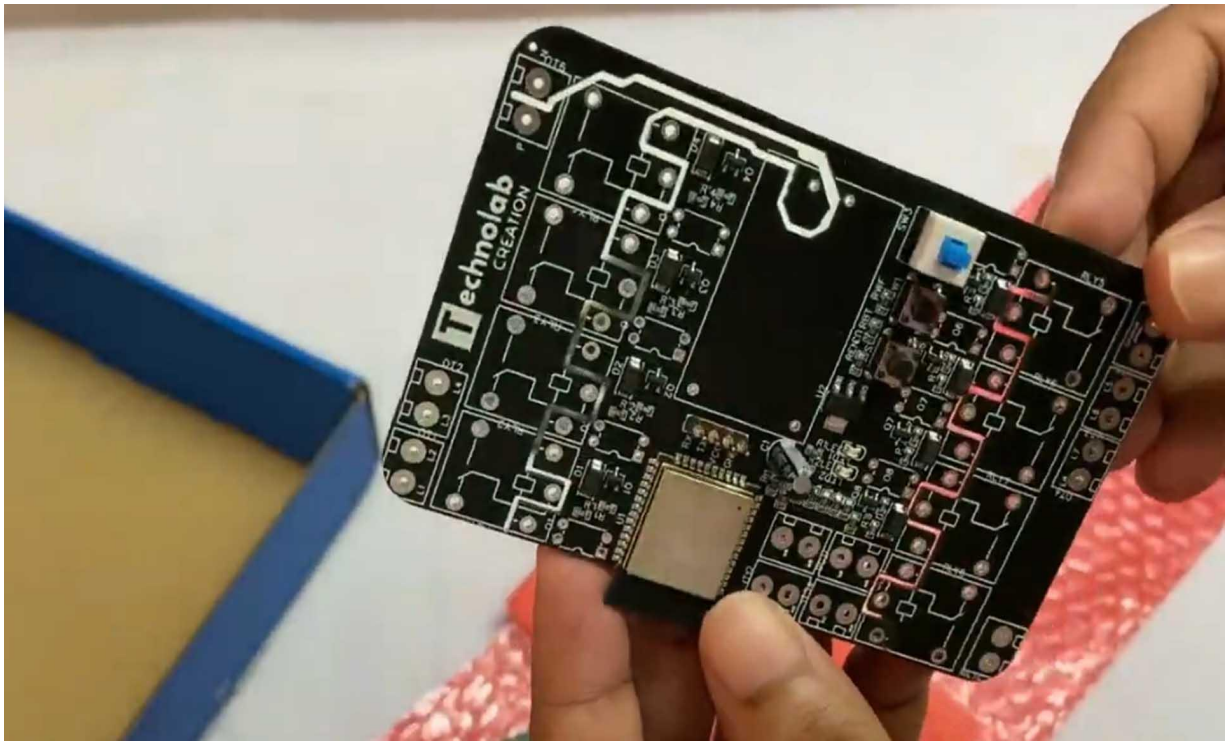
Please note that this project can be quite complex, and it may be advisable to work with a team with expertise in hardware design, embedded programming, and PCB manufacturing. Additionally, ensure that your project complies with all relevant regulations and safety standards in your region.

8 NODE SMT SMART HOME AUTOMATION PCB

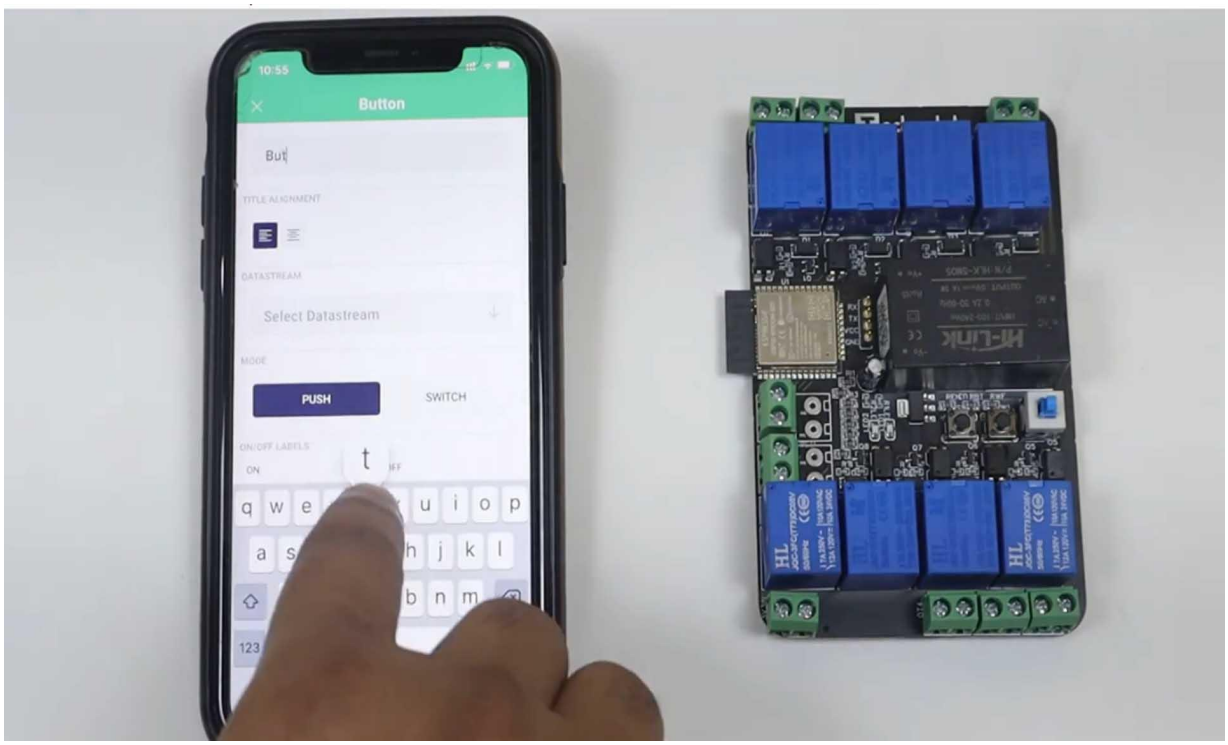
Hey, hello friends, welcome to another project. In this project, I am gonna introduce my newly designed fully summit component used home automation PCB. This is an 8-node home automation PCB. The size of this PCB is very small due to summit components, and it will easily fit inside your electrical switchboard. This PCB has an inbuilt OTA button, meaning you can update the code wirelessly over the air, and there are 2 onboard LEDs. You could use them according to your need, like testing your code or something else. In my case, I have used them as a Wi-Fi indicator. That is, if Wi-Fi connection is available, then both the LEDs will glow. As only a single LED will glow, we can give input to control our devices via manual switch button. This feature is useful when there is no Internet connection available. And we can also monitor the real-time status of appliances in the app, whether it is on or off. So we can control our appliances manually as well as through the Internet from anywhere in this world. This PCB is compatible with all the smart speakers available in the market like Amazon Alexa, Google Home, Apple Siri. If you want, I can make a project on how to control our home appliances using these smart speakers. Just let me know in the comment section below. And now, let's get started with this project. This project is sponsored by JLCPCB. JLCPCB is a well-known PCB prototype company in China. It is specialized in quick PCB prototype and small batch production. You can now order a minimum of five PCBs for just $2. For more details, check the description. This is the schematic of today's home automation PCB. If you want, you can download this schematic from the description to design your own custom PCB.



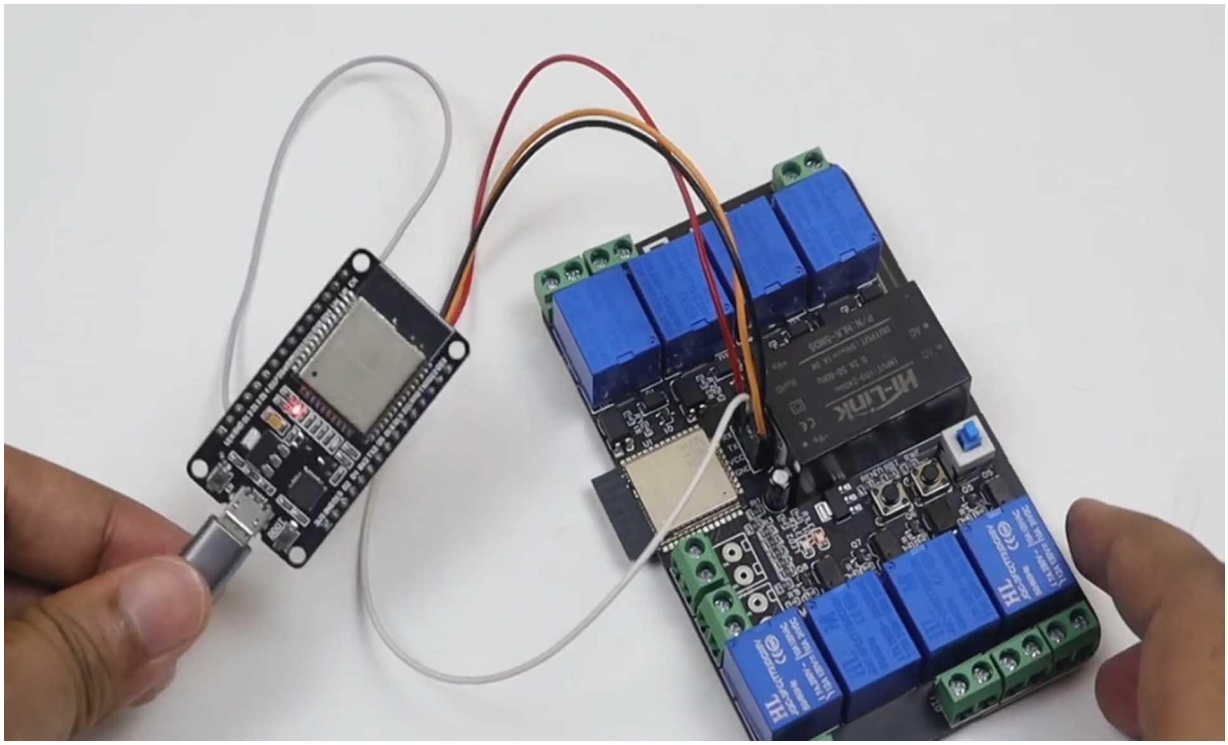
Now convert this schematic into a PCB. After completing the design of your PCB, you can directly order the PCB from JLC PCB or just download the Gerber file from here. After that, go to the JLC PCB website, then click on the Quote Now button under the summit assembly. After that, upload the Gerber file of your PCB. After that, select the number of PCBs and color masking of the PCB if you want. After that, select the summit assembly service, and here you have to select on which surface you want your components to be soldered, either the top surface or bottom. After that, click on the confirm button.



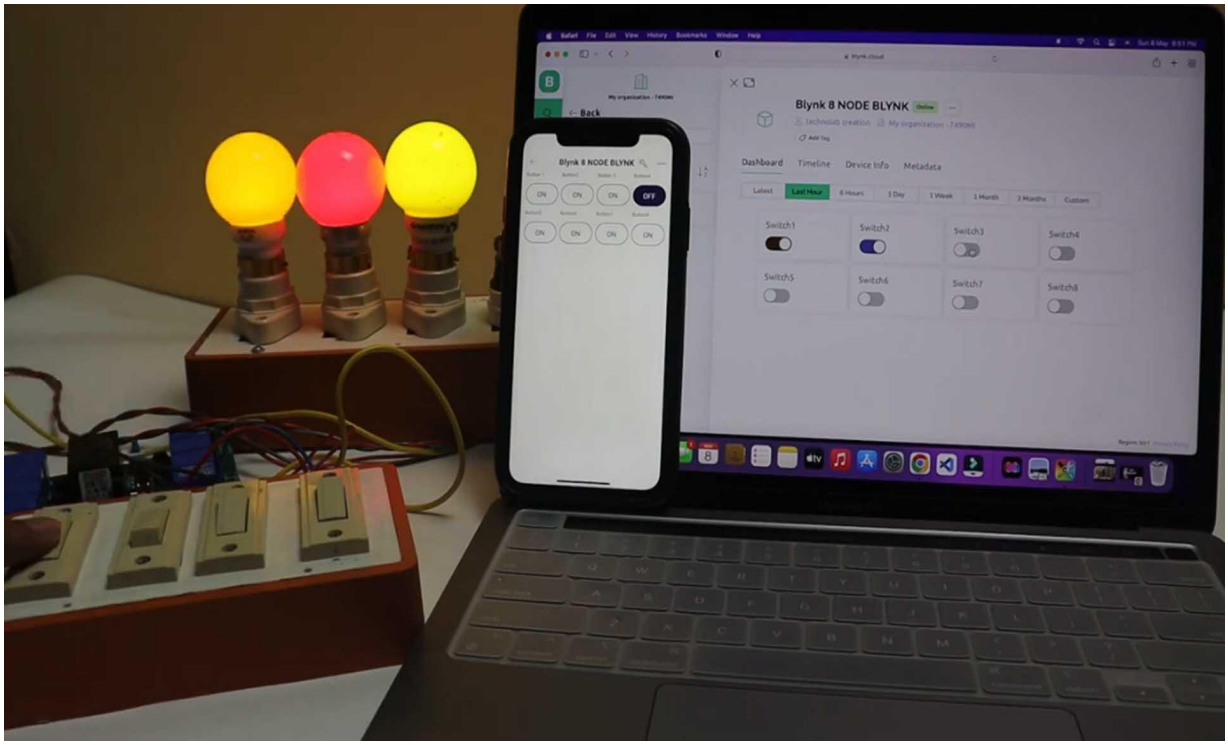
Now here you have to upload two more files. One is the CPL. That is the pick and place file, and another one is BOM. That is the Bill of Material. You can download these files from your easyEDA account. Just open that PCB project on your easyEd account and then click on Fabrication then BOM. Now click on export. Now click on Export BOM to download the BOM file. Similarly, download the CPL file. After downloading both the files, just upload both the files here onto this page. After that, select next. Now here it will show all the summit components which are to be soldered and also we can select which components will be soldered or not. Select the components according to your preference. After that, click on the next button, then click save to card to complete your order. After a week, my PCB arrived at my place in a new blue box of JLC PCB. Let me open the box. The packaging of the PCB in bubble wrap is very good.



Here it is, our home automation PCB. The quality of PCB is good, and the summit components are soldered well. After soldering, the rest of the components PCB will look like this. Neat, clean, and well-arranged. To flash the code into the ESP 32 chip, I will use an ESP 32 Development Board. Now make the connections according to this schematic. This is the code for our today's home automation project. Download this code from the description and open it in the Arduino IDE.



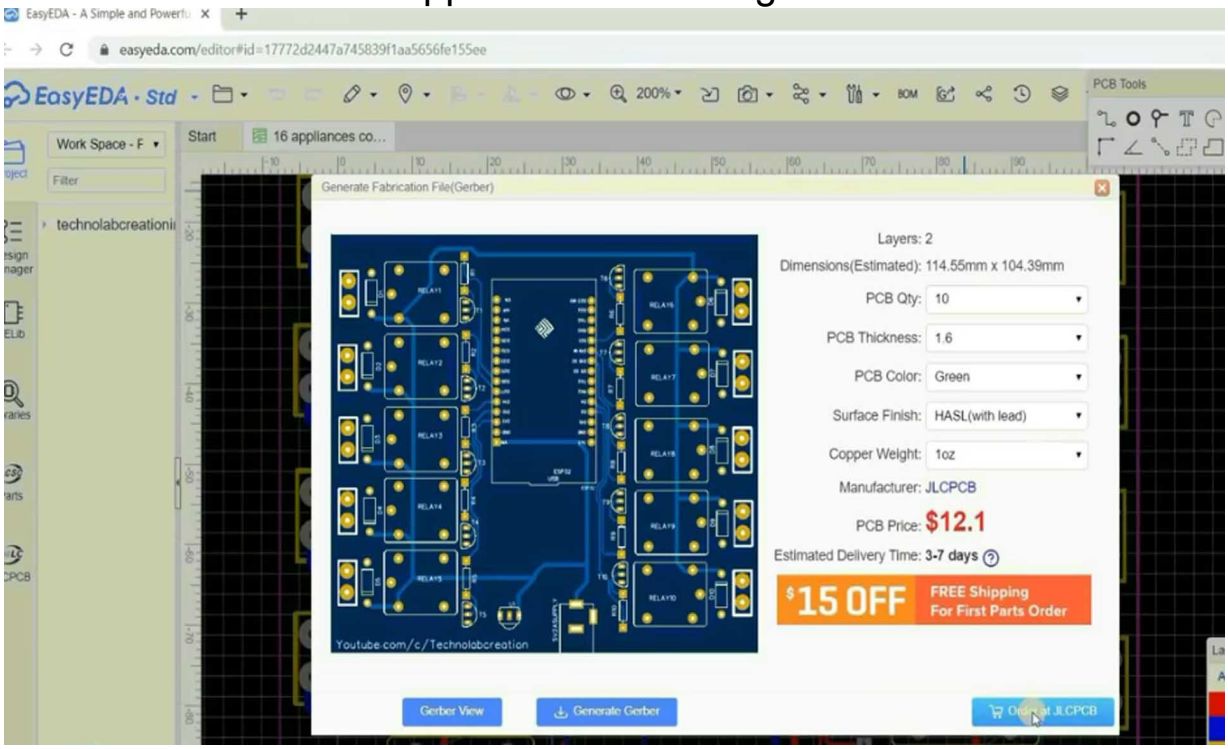
Before you upload the code, you need to make a few changes in it. First, in this section, you need to enter the SSID and password of your router or hotspot. After that, here you have to enter the authentication token sent by Blink on your registered email ID. Just copy and paste it here, and the rest should be okay. After selecting the right board and COM port, hit the upload button.



After clicking the upload button onto the PCB, I will press and hold the boot button and press the reset button once to make this module go inside the boot mode. As you can see, the code starts uploading. Now connect all the bulbs and switches according to this circuit diagram. Now everything is done. Let's see the project in action.

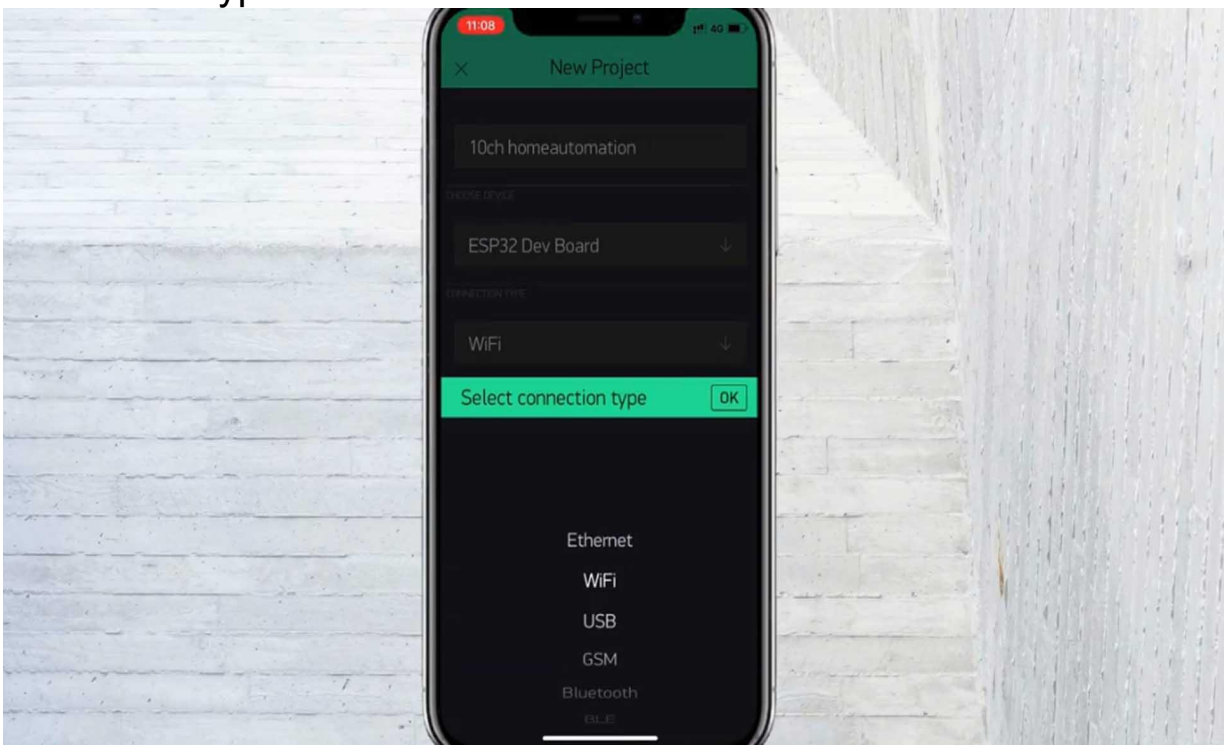
HOME AUTOMATION SYSTEM USING ESP32

This project is sponsored by JLCPCB. JLCPCB is a well-known PCB prototype company in China. It is specialized in quick PCB prototype and small batch production. You can now order a minimum of five PCBs for just $2. For more details, check the description. In my sum of last projects, I had used this 4-channel PCB to control the home appliances. But what if we want to control more than four appliances? So to solve this issue, we have another PCB in which we are able to control a total of 10 devices using this single PCB. Now in this project, we are going to build a 10-channel home automation system using the BlinkIoT cloud platform. Now we can control our home appliances from anywhere in this world using this PCB and Blink mobile application. So let's get started.



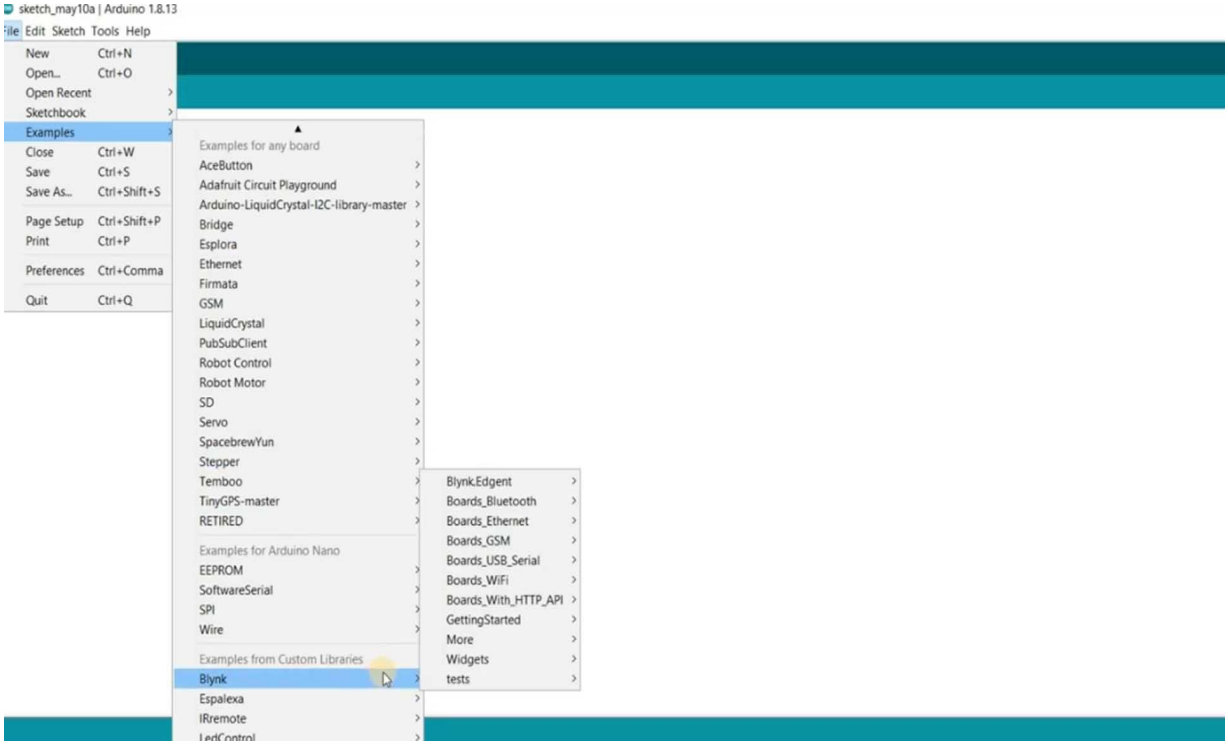
For this project, I am going to use my old PCB. You can download the Gerber file of this PCB from the description. For designing my

PCB, I am using EasyEDA. EasyEDA is very easy and simple for designing the PCB. After designing the PCB, I directly ordered it from GLC PCB for manufacturing of PCB. After uploading the Gerber file, the software automatically detects the default settings. However, if you want, you can change these settings now. Save to cart to complete your order. After seven days, my PCB arrived at my place. Connect all the bulbs in this manner. Download the Blink mobile application on your smartphone. This app is available for both Android as well as for iOS. After installing the app, let's set it up. Tap on a new project. Give the name of your project. I am giving it "10-channel home automation". Now select the hardware type. In my case, it is the ESP 32 Development Board. Now select the connection type.



Choose the theme: Dark or Light, whatever you want. Now tap on create project. We have successfully created the project, and for this project, an authentication code is sent to our registered email ID. We need this code at the time of coding, so keep it. Now tap on the screen to open the widget box. Select button. Tap on the button for its settings. Give an appropriate display name to this button. I am giving it "Button One". After this, select the Digital pin, GP26 mode, switch, and off/on level, leaving it as it is. Now tap on OK. Here we

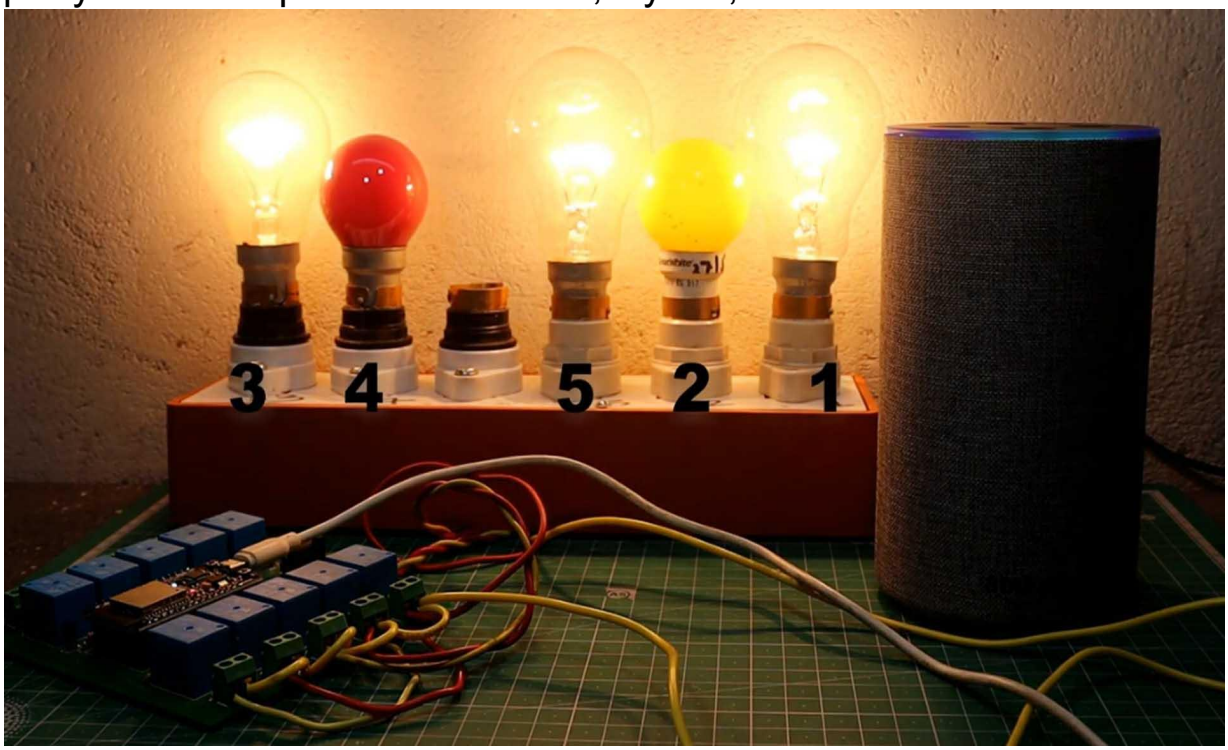
have successfully set up "Button One". In the same way, set up nine more buttons as we need to control 10 relays in our project. Here we have successfully set up all the 10 switches. Coding is the easiest part of this project, and first of all, we need to have the Blink library installed in our Arduino IDE. For that, we need to go to Sketch > Include Library, then click on Manage Libraries. Here we have to search for "Blink". This is the Blink library that we need to install in our Arduino IDE. Select the latest version and click on install.



After installing the library, click on the close button. Now go to Files > Examples. Here we have to navigate to Blink, then BoardsWi-Fi, and then into ESP 32 Wi-Fi. This is the code for today's project. Now here you need to add the authentication token, which is already sent to you on your registered email ID. Just copy and paste it in your code. After that, in this section, you have to enter the access ID and password of your router or hotspot. That's it. Our coding is complete. Now hit the upload button after selecting the right board and COM port. Everything is done. Now let's see our project in action.

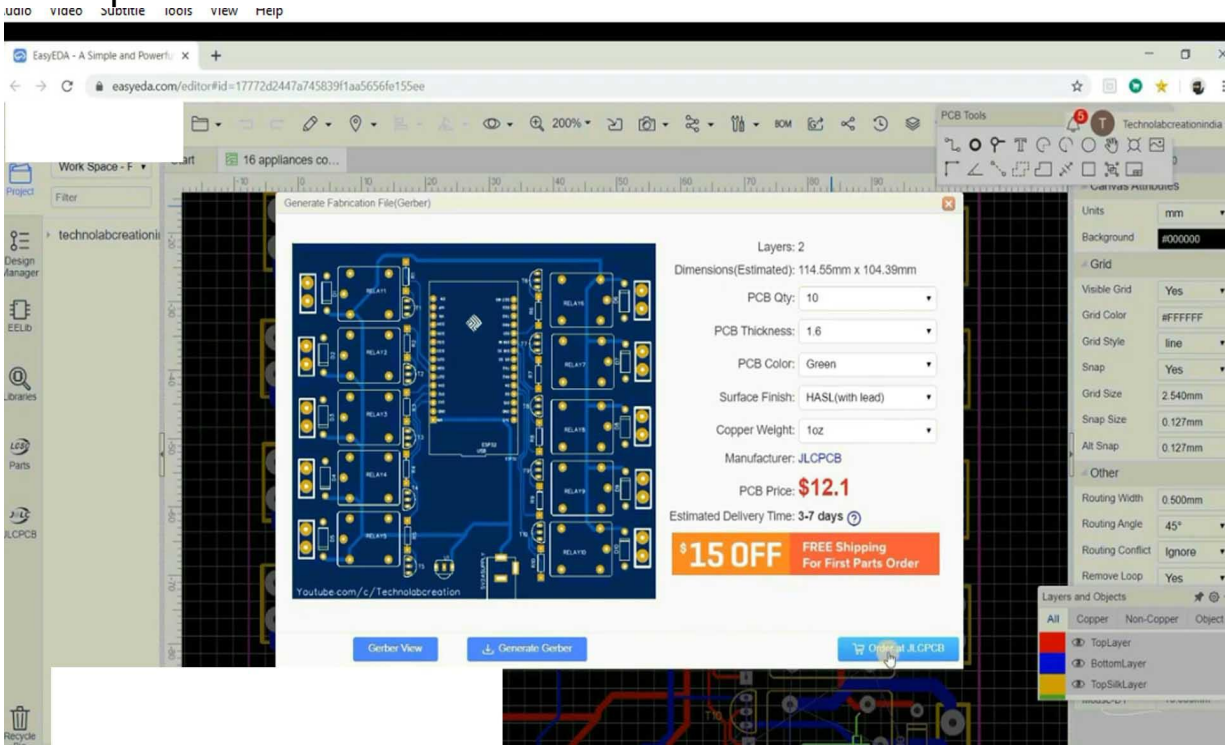
HOME AUTOMATION SYSTEM USING ALEXA ESP32

Hey, hello friends, welcome to another project. Now in this project, we will make a home automation project using Alexa and ESP 32. This is the 10-channel home automation system in which we control our home appliances by giving voice commands to the Alexa smart speaker. And the best part of this project is we don't need any third-party IoT cloud platform like Blink, Cynric, or IFTTT.

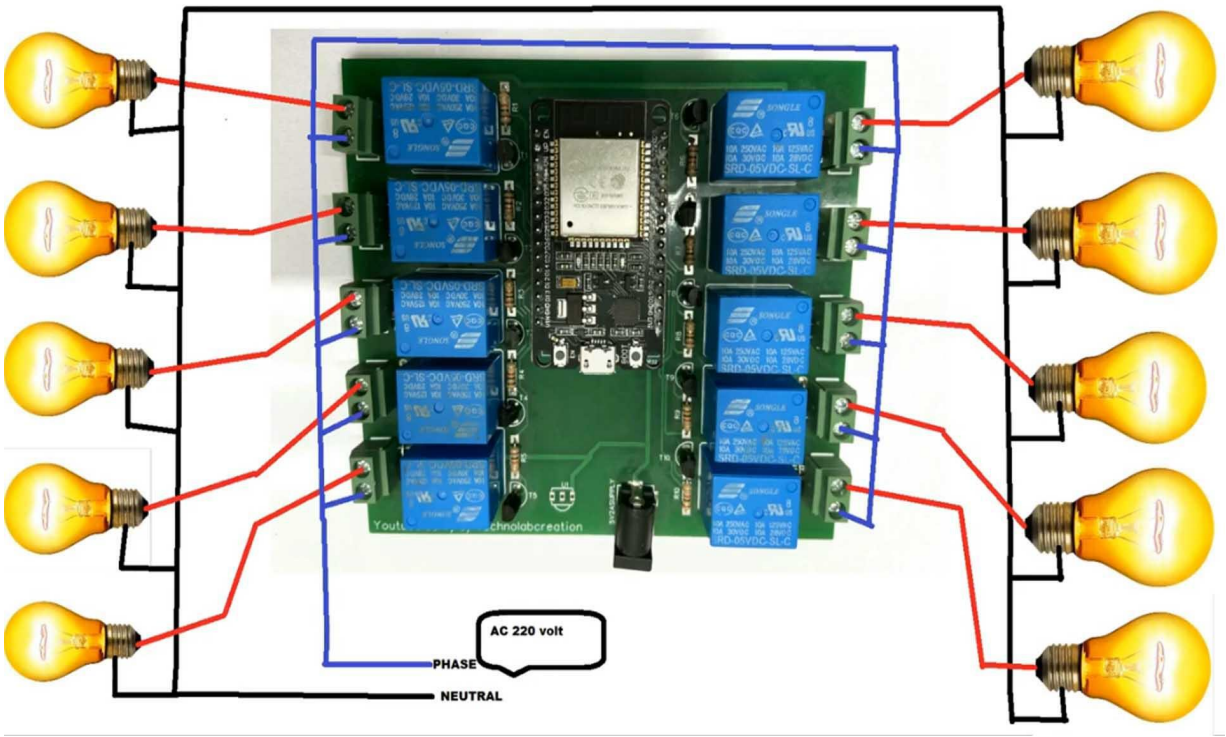


We just need ESP 32 with this custom-made PCB and Alexa smart speaker. Let me show you how it works. "Alexa, all lights on. Alexa, light three off. Alexa, light one off. Alexa, light 5 off. Alexa, light four off. Alexa, light two off." You can give instructions even in Hindi. Let me show you: "Sari light on karo, Alexa. Sari light band karo, Alexa. Light 4 on karo, Alexa. Sari light on karo, Alexa. Sari lights off karo, Alexa." This project is sponsored by JLCPCB. JLCPCB is a well-known PCB prototype company in China. It is specialized in quick

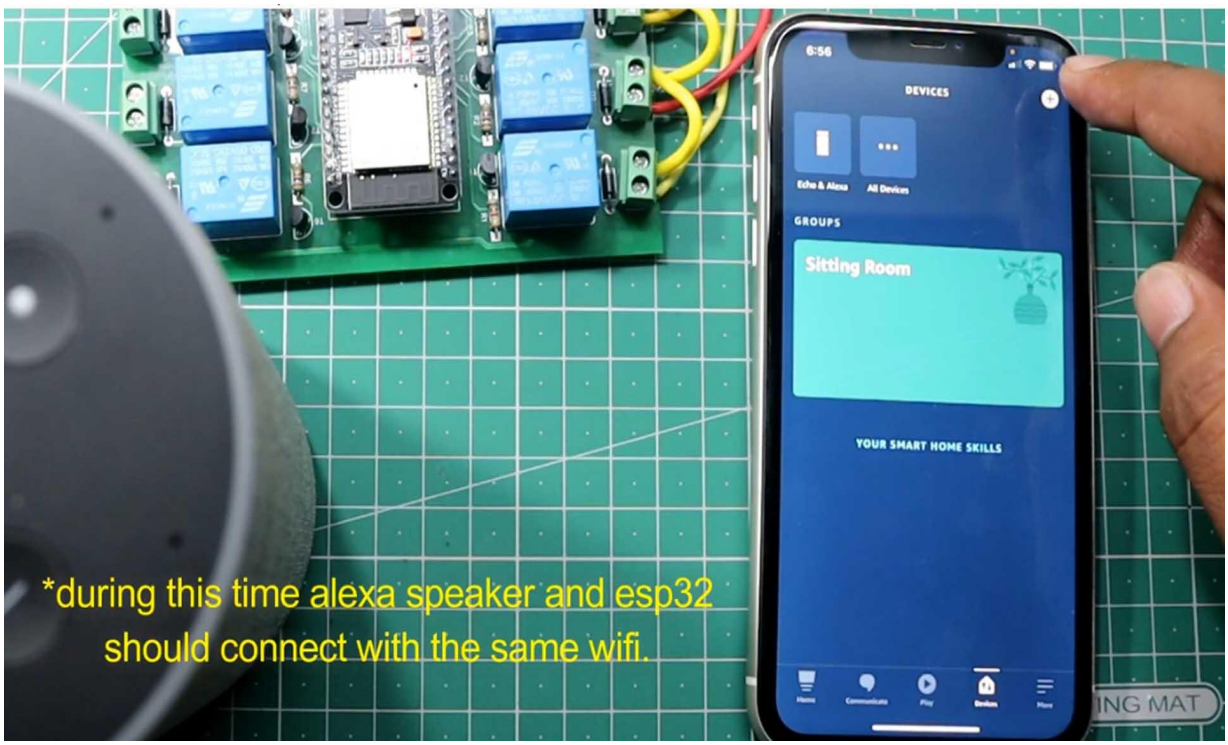
PCB prototype and small batch production. You can now order a minimum of five PCBs for just $2. For more details, check the description.



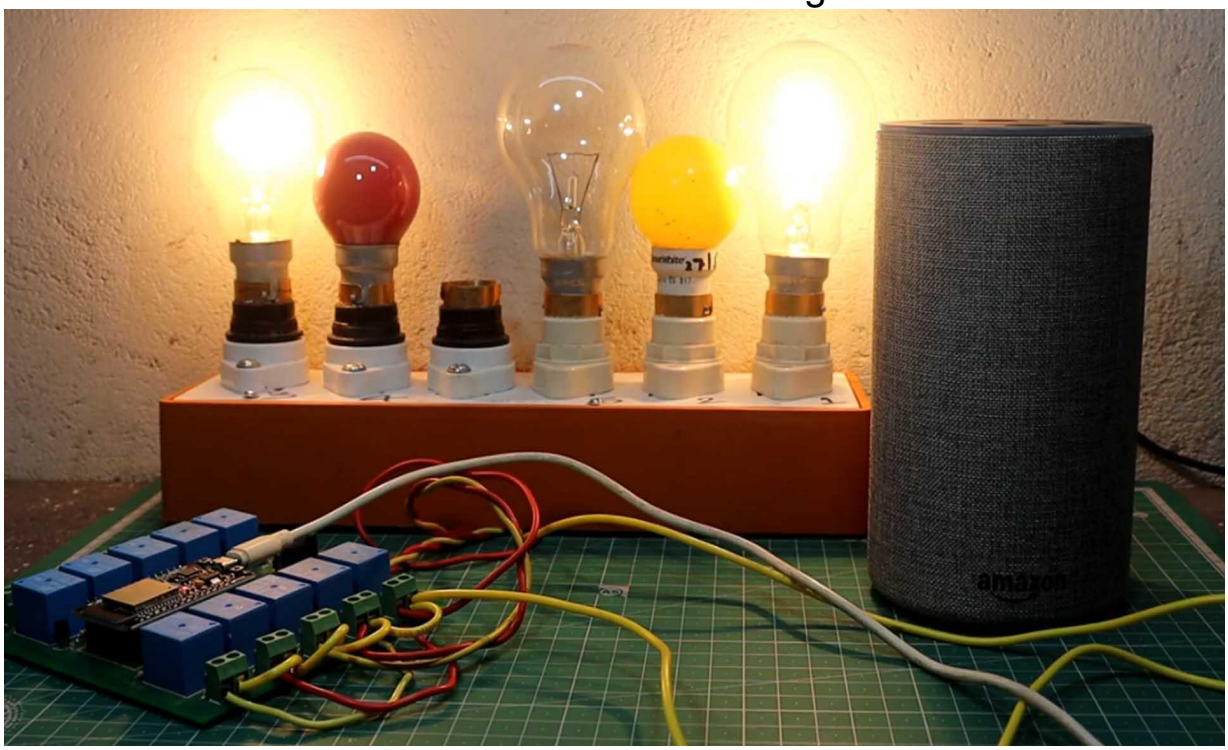
For this project, I'm going to use my old PCB. You can download the Gerber file of this PCB from the description. For designing my PCB, I'm using EasyEDA. EasyEDA is very easy and simple for designing the PCB. After designing the PCB, I directly ordered it from GLC PCB for manufacturing of PCB. After uploading the Gerber file, the software automatically detects the default settings. However, if you want, you can change these settings now. Save to cart to complete your order. After seven days, my PCB arrived at my place. Connect all the bulbs in this manner. This is the code for today's project. Let me explain a little bit about this code. Here, we added some basic libraries that are needed to run the code, like WiFi.h and ESP8266WiFi.h. The ESP8266WiFi.h is used for NodeMCU.



That means you can use the NodeMCU board. If you don't have an ESP32 board, this code will work for both. This is the ESP Alexa library. This library must be added in your Arduino IDE. To add this library, go to Sketch > Include Library > Manage Libraries. A pop-up will come out. In the search field, type "ESP Alexa." Select the latest version and click on install. It will take a few seconds to install the library. Now click on close. In this section, you have to add the SSID and password of your router or hotspot. Here, we give the name of all the 10 devices:



light one, light two, light three, light 4, and so on. That's it. Now, after selecting the right board and COM port, hit the upload button. Open the Alexa app. Tap on devices. Tap on the add button to add devices. Now click on "Add devices." Select light.

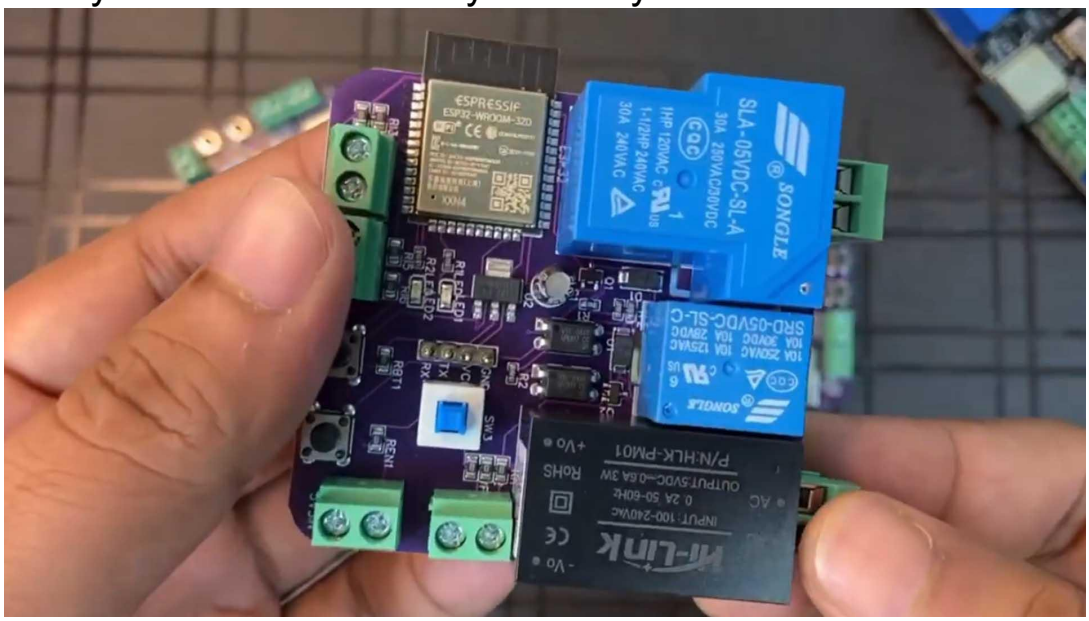


Scroll down and click on "other." Now click on "Discover devices." It

will take some time to connect all the devices. Here, you can see ten devices. Now tap on next. Here, you can see these are the devices that we created in our code. Now set up all the devices one by one.

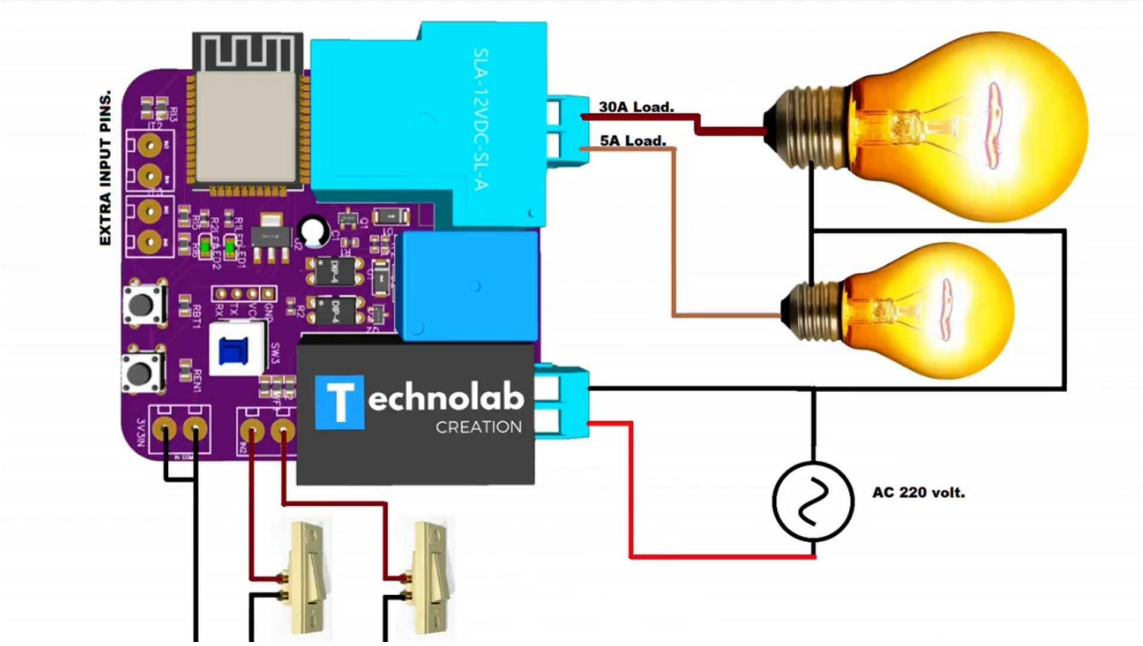
HOME-AUTOMATION PCB FOR HEAVY LOAD APPLIANCES

Hey, hello friends, welcome to another project. In this project, I am going to introduce my newly designed 30 MPR home automation PCB. I have also designed loads of other home automation PCBs like 2 node, 4 node, and 8 node home automation PCBs. These PCBs are fully tested and work very well. These PCBs are best for home automation systems. But these PCBs could handle only up to 10 ampere load. So we can only use this PCB for small load appliances like light bulbs and televisions which consume less power. But if you want to automate heavy-load appliances like air conditioners, washing machines, and more, then in that case, we couldn't connect to these PCBs. These PCBs will not handle such heavy loads and eventually the relay will burn out.



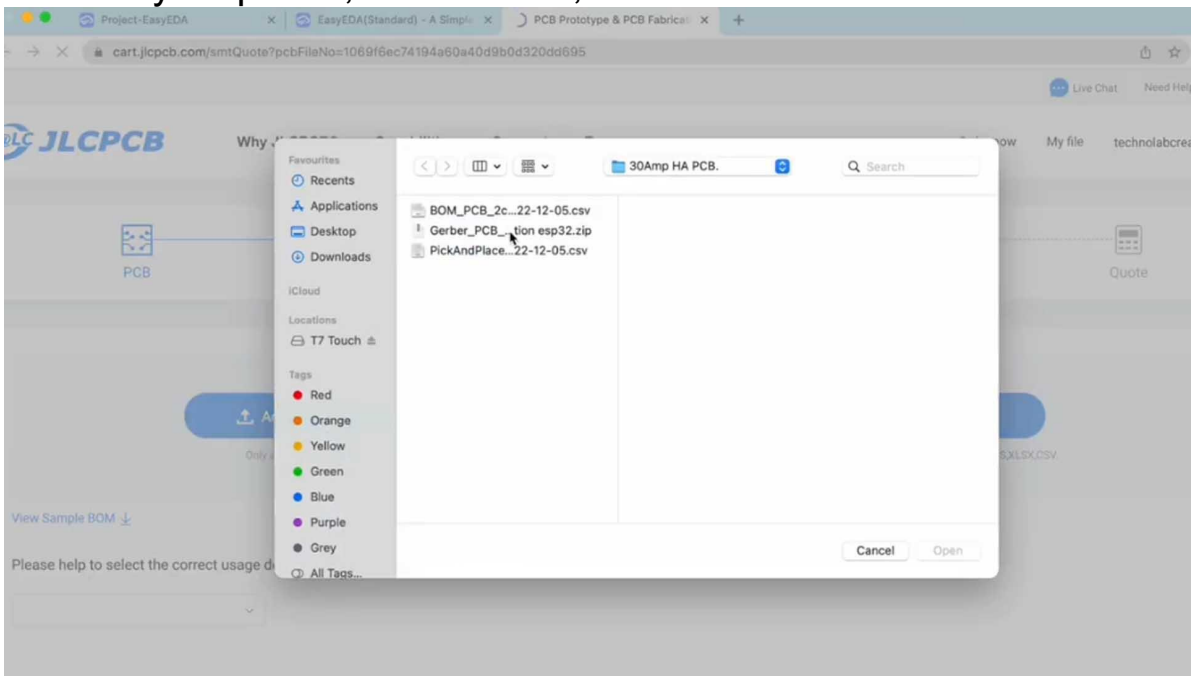
So to solve this problem, I have designed a brand new home automation PCB which can easily handle up to 30 Mpereload. Most of the heavy-load appliances that we use in our homes consume current

under 30 Amperes. So this PCB is perfect for those appliances. In this way, we can connect any heavy-load appliances. This PCB is compatible with all the popular IoT platforms like Blink, ESP Rainmaker, and more. In this way, we can easily integrate this PCB with Alexa and Google Assistant. Apart from this, we can also give manual input to this PCB and connect via Bluetooth to control the appliances from a smartphone application in a local area network. And the size of this PCB is very small and can easily fit inside the electrical switchboard. And there are 2 onboard LEDs which we can use in many ways like for testing code and Wi-Fi indicator.

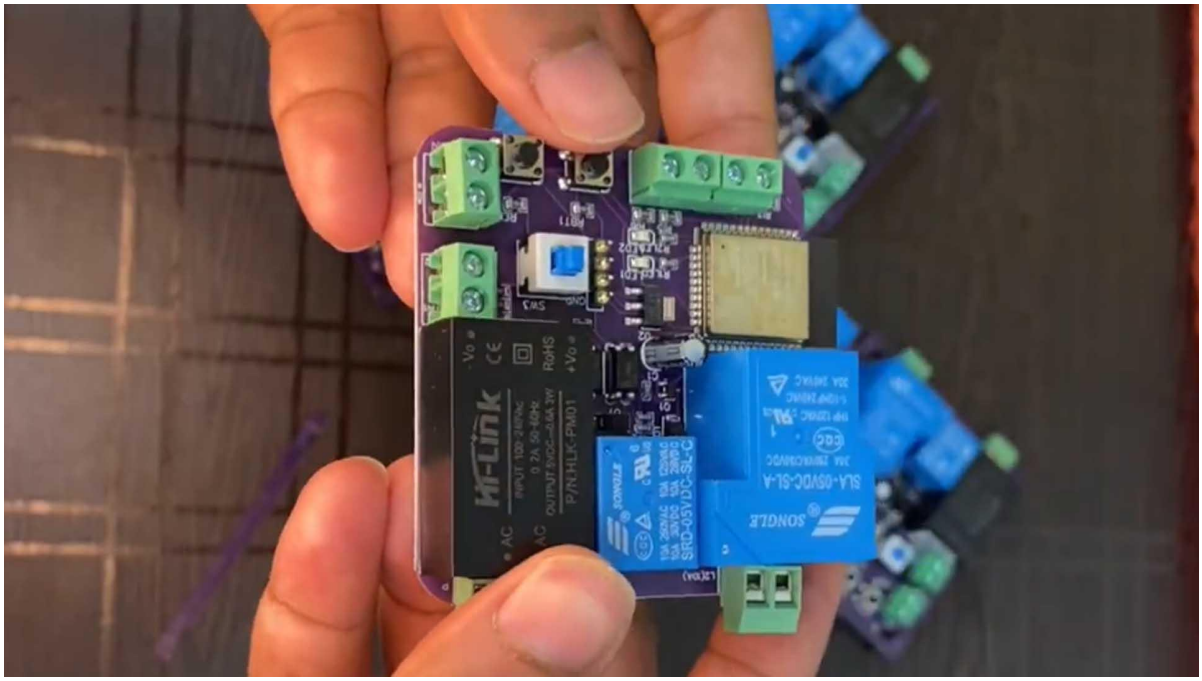


During the project, I will let you know how to upload the code, the circuitry, and the connection of relays and switches. So I recommend that you watch the complete project till the end. Now let's get into this project. This is the schematic of the PCB. If you want your own custom-designed PCB, then you can download this schematic from the link given in the project description. After making the schematic, convert it into PCB. Arrange and place all the components in desirable places. Once the layout is ready, route the wiring and complete the design of the PCB. After the completion of PCB design, you need to download 3 files which will be required during the PCB order. These files are BOM, Gerber, Cpl (pick and place file). Now go to the JLCPCB website and click on the Quote Now button under PCB assembly. Click here to upload the Gerber file of your PCB.

Here, JLC PCB will automatically set all the parameters of the PCB. Select the PCB quantity and color masking of the PCB by yourself. I am selecting the purple color. Scroll down below and select PCB assembly. Here, you have to select on which side you want the PCB assembly: Top side, bottom side, or on both sides.



In my case, I want only the top side. After that, click on the confirm button. For PCB assembly, we need two more files: one is BOM (Bill of Material) and the second one is Cpl (pick and place file). Upload these two files one by one. Here, you need to give the description about your PCB for which purpose you want to make this PCB. I am giving it a "30 MPR Home Automation PCB". After that, click on the next button. All the components were shown here that are to be assembled. In case you do not want to assemble any particular component, then you can deselect that component. After checking all the components, click on the next button. Here, you will see a computer version of components placement which seems not accurate. This is only for reference purposes. Now click on "save to cart" to complete your order after seven days. PCBs arrived at my place.



As usual, the quality of the PCB is very premium and the components are soldered very well. Traces are perfect, silk screen is fine, purple color, PCB masking looks very beautiful, and PCBs look pretty professional. To upload the code into ESP 32 chip, I will use ESP 32 Development Board. Connect the PCB to the ESP 32 board as per the circuit diagram. This is the example code which I am going to upload in the PCB. This code is for controlling 2 relays using the ESP Rainmaker app. I have explained this code in detail and also explained how to add the latest ESP 32 library in your Arduino IDE in my previous project. The link of that project is available in the description as well as in the card section. If you want, you can check out that project. Now go to tools and select the right board that is ESP32 development module and then select the partition scheme as Rainmaker. And in the last, select the right communication port, then click on the upload button to upload the code. After clicking the upload button on the PCB, press and hold the boot button and press the reset button once to make the module go inside the boot mode. Here, as you can see, the code is successfully uploaded. After uploading the code, you need to configure the ESP Rainmaker app. But I am skipping this configuration part because the configuration is the same as in the two-channel relay. Just check out that project. You can find the link to

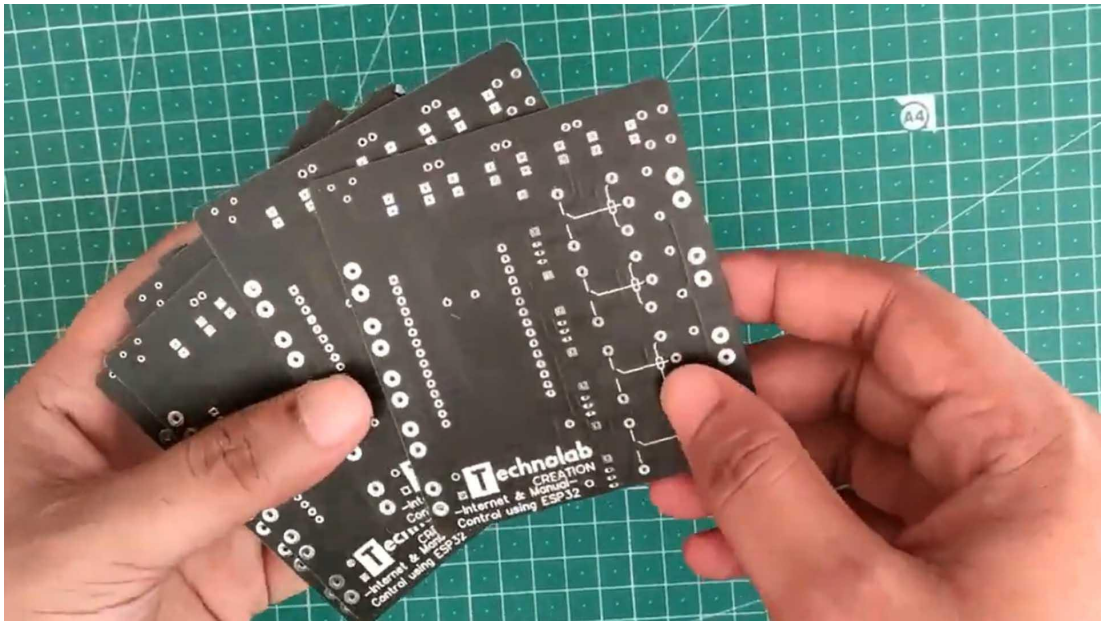
that project in the description as well as in the card section. After configuring the ESP Rainmaker app, you will be able to control your heavy-load appliances from the smartphone application as well as from the smart speakers. Now connect all the switches and appliances as per this connection diagram. Here, I am using a light bulb to demonstrate this project. Now everything is done. Let's see the project in action. "Turn on the light one." "Got it, Turning the light one on."

ALEXA MANUAL CONTROL HOME AUTOMATION SYSTEM USING ESP32

Hey, hello friends, welcome to another project. Now in this project, we will be going to make a voice control plus manual control home automation system. We will use Alexa as a Voice Assistant, and for manual control, we will use switch buttons that we regularly use in our homes. And for the making of this project, we don't need any third-party IoT cloud platforms like IFTTT, Cylindrical Blink. And the best part of this project is we can give instructions even in English as well as in Hindi. Let me show you.

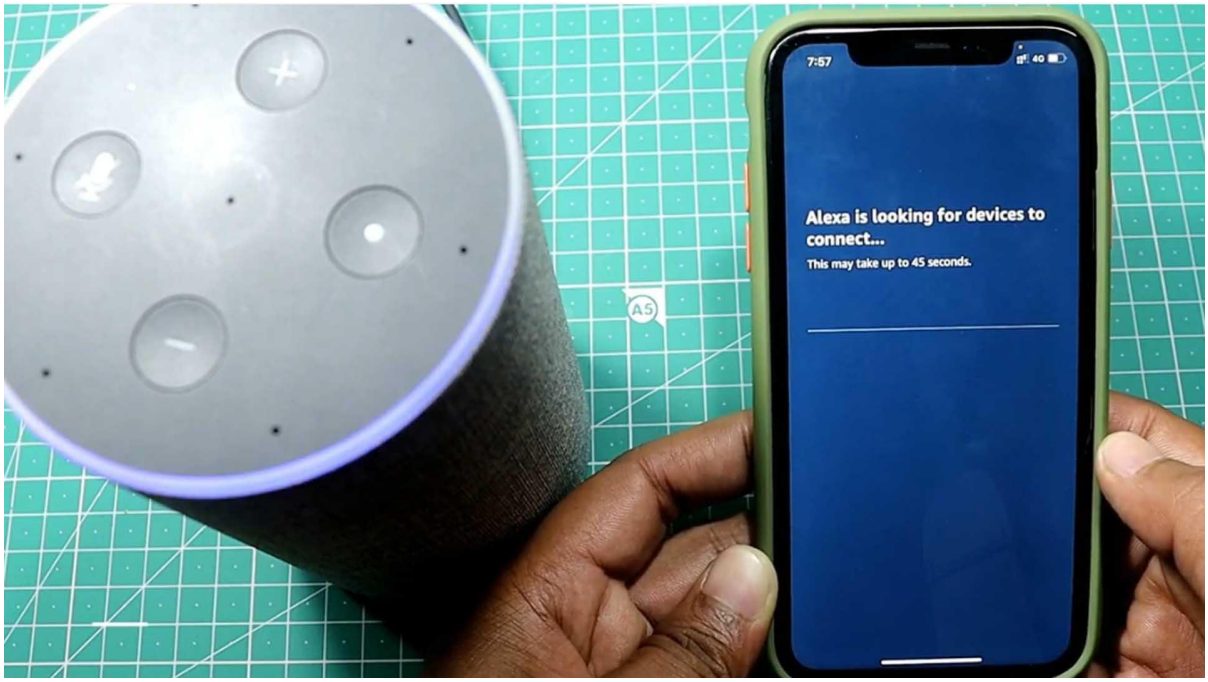
"Alexa, all lights off." "Alexa, light 2 on." "Alexa, light 4 on." "Alexa, light 2 on." "Alexa, light 3 on Karo." "Alexa, saari light on Karo." This project is sponsored by JLCPCB. JLCPCB is a well-known PCB prototype company in China. It specializes in quick PCB prototypes and small-batch production. You can now order a minimum of five PCBs for just $2. For more details, check the description. For this

project, we need ESP32, high link A/C to DC converter, 5V relay, BC547 NPN transistor, 540007 diode, 10K and 1K ohm resistors, 2 terminal connectors, female header, and a custom-designed PCB. Ordering PCBs from JLCPCB is very easy. Just upload your Gerber file. JLCPCB automatically recognizes all the features of the PCB. Choose the PCB color and other extra features, and after that, click on "Save to Cart" to complete your order. After a few days, here are the PCB boards in the new blue box of JLCPCB.



Now solder the components on the PCB. After soldering the components, the PCB looks like this, neat and clean. Connect all the bulbs and switches in this manner. This is the code for today's project. Let me explain a little bit about this code. Here, we added some basic libraries that are needed to run the code, like WiFi.h and ESP8266WiFi.h. ESP8266WiFi.h is used for NodeMCU. That means you can use a NodeMCU board if you don't have an ESP32 board.

This code will work for both. This is the ESP Alexa library. This library must be added in your Arduino IDE. To add this library, go to Sketch, then Include Library, Manage Libraries. A pop-up will come out. In this search field, type "ESP Alexa," select the latest version, and click on "Install." It will take a few seconds to install the library. Now click on "Close." In this section, you have to add the SSID and password of your router or hotspot. Here, we give the name of all the four devices: Light one, Light two, Light three, and Light 4. You can give it any name you want according to your need. That's it. Now after selecting the right board and COM port, hit the upload button.



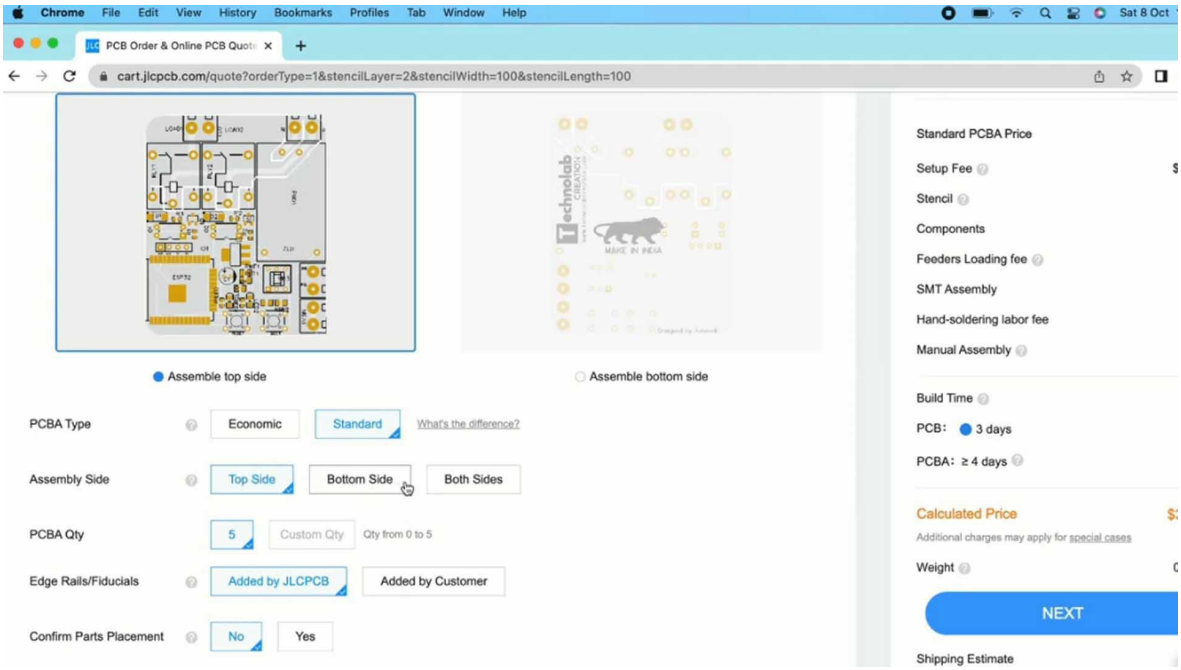
Open the Alexa app. Tap on Devices. Tap on the Add button to add devices. Now click on Add devices. Select light. Scroll down and click on Other. Now click on Discover devices. It will take some time to connect all the devices. Here you can see it found four devices. Now tap on Next. Here you can see these are the devices that we created in our code. Now set up all devices one by one. Now, if you click on Lights, here you can see all the devices are added. With Alexa, everything is done. Let's see how it works. "Alexa, all light on." "Alexa, all light off." "Alexa, saari light on Karo." "Alexa, saari light band Karo." "Alexa, saari light band Karo."

MANUAL CONTROL HOME-AUTOMATION SYSTEM USING ESP RAIN-MAKER

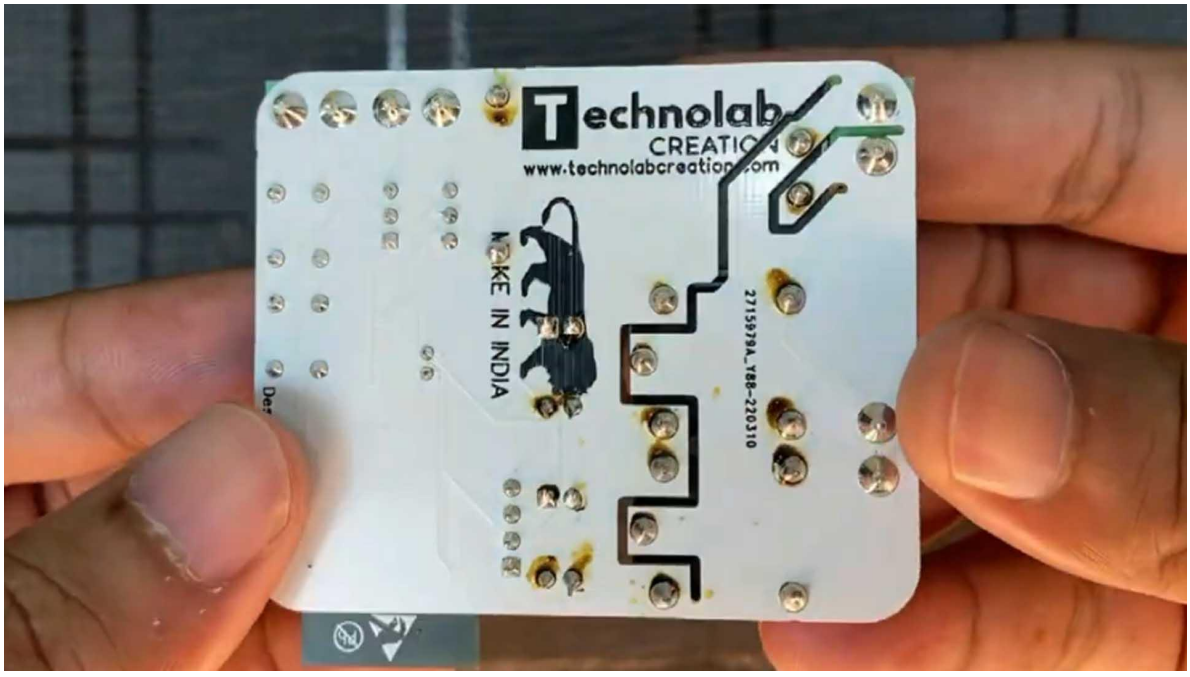
Controlling our home appliances from Alexa and Google Assistant is fun, and these smart speakers are very popular nowadays, almost everyone has them in their houses. Controlling home appliances from these smart speakers is pretty amazing and also very convenient. So in this project, I am going to make a home automation system in which we will control our home appliances from these smart speakers. "Alexa, turn on the light one." Turn on the light, too. And apart from these smart speakers, we can also control our home appliances from a smartphone application and also with the regular manual switches. And the best part of this home automation project is that you can easily make this home automation system even if you don't know anything about coding. This project is as simple as that. In this home automation project, I am going to use the ESP RainMaker app, which is free and available for both iOS and Android.

In this project, we will control 2 relays. As this PCB has only two relays, and apart from this, I also have eight-node and four-node home automation PCBs. The codes for all these PCBs are available in the description. These PCBs are fully tested and work very well. If you purchase this PCB, you will get a QR code along with it. Just scan this QR code with the ESP RainMaker app and easily integrate the Alexa smart speaker and Google Assistant to it. Make your devices smart. During the project, I will show you these steps in detail. I will also explain the code, circuitry, and connection of all the switches. So I recommend you watch the complete project till the end. Now, let's go into this project. This is the schematic of the PCB. If you want your own custom-designed PCB, then you can download this schematic from the link given in the project description. After making the schematic, convert it into a PCB. Arrange and place all the components in desirable places. Once the layout is ready, route the wiring and complete the design of the PCB. After the completion of the PCB design, you need to download 3 files which will be required during the PCB order. These files are BOM, Gerber, Cpl (pick-and-place) files. Now, open the JLC PCB website and click on the "Quote Now" button under PCB assembly. Click here to upload the Gerber file of your PCB. Here, JLC PCB will automatically set all the parameters of the PCB. Select the PCB quantity and color masking of the PCB by yourself. I am selecting white color. Scroll down and

select "PCB Assembly." Here, you have to select on which side you want the PCB assembly: top side, bottom side, or on both sides. In my case, I want it only on the top side. After that, click on the "Confirm" button.

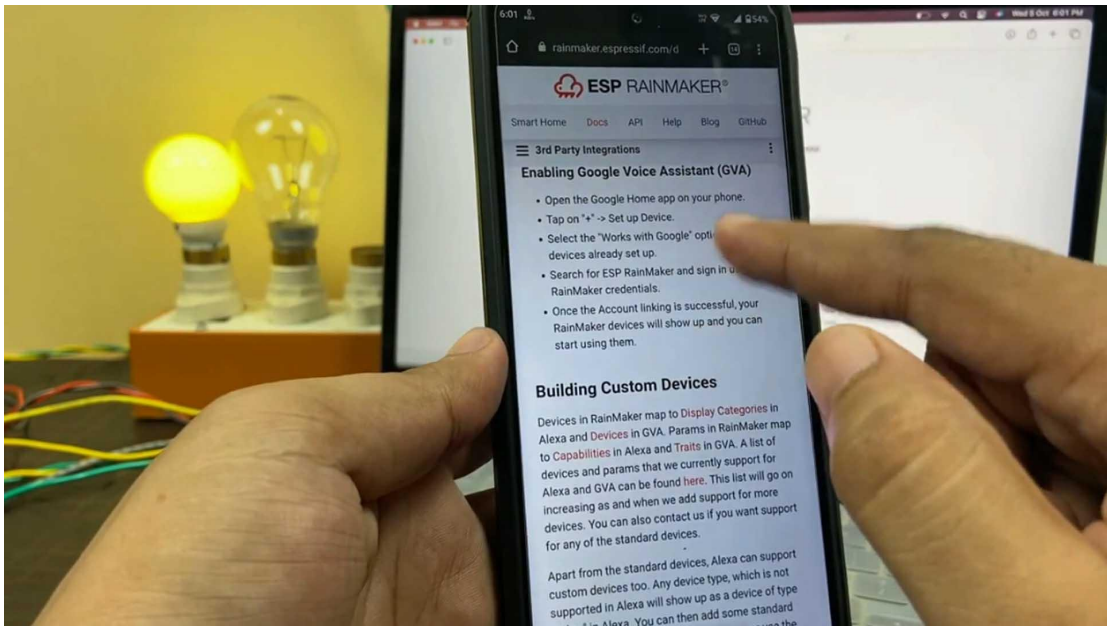


For PCB assembly, we need two more files: BOM (Bill of Materials) and Cpl (pick-and-place) file. Upload these two files one by one. Here, you need to give a description about your PCB, for which purpose you want to make this PCB. I am giving it as "2-node SMT." After that, click on the "Next" button. All the components were shown here that are to be assembled. In case you want to not assemble a particular component, then you can deselect that component here. After checking all the components, click on the "Next" button. Here, you will see a computer version of components placement, which seems not accurate. This is only for reference purposes. Now click on "Save to Cart" to complete your order. After seven days, the PCBs arrived at my place



As usual, the quality of the PCBs is very premium and the components are soldered very well. Traces are perfect, the silk screen is fine, and the white-colored PCB masking looks very beautiful. The PCBs look pretty professional. Make all the connections of the bulbs and switches as per this circuit diagram. To upload the code into the ESP 32 chip, I will use the ESP 32 Development Board. Connect the PCB to the ESP 32 board as per the circuit diagram. This is the code for our today's home automation project. Before you upload the code, first, you need to update the ESP 32 boards' library in your Arduino IDE. Now copy this link provided in the code. Now open Arduino Preferences and paste the copied link here. Then click "OK." Now go to "Tools" and then click on "Board Manager." Here, search for ESP 32. Install the latest ESP 32 boards in your Arduino IDE. Close this window after installation. Here, I have to find the names of devices, like Light one, Light 2, and so on. You can give any name you want. Here, I have defined the pins for relays and switches. If you are using my PCB, then there's no need to change anything; just upload the code as it is. One thing you could change here is the setup part of the code, the name of the node. This name will appear in the app as the name of the device. After that, there is no need to change anything. Now go to "Tools" and select the right board, which is the ESP 32 Dev

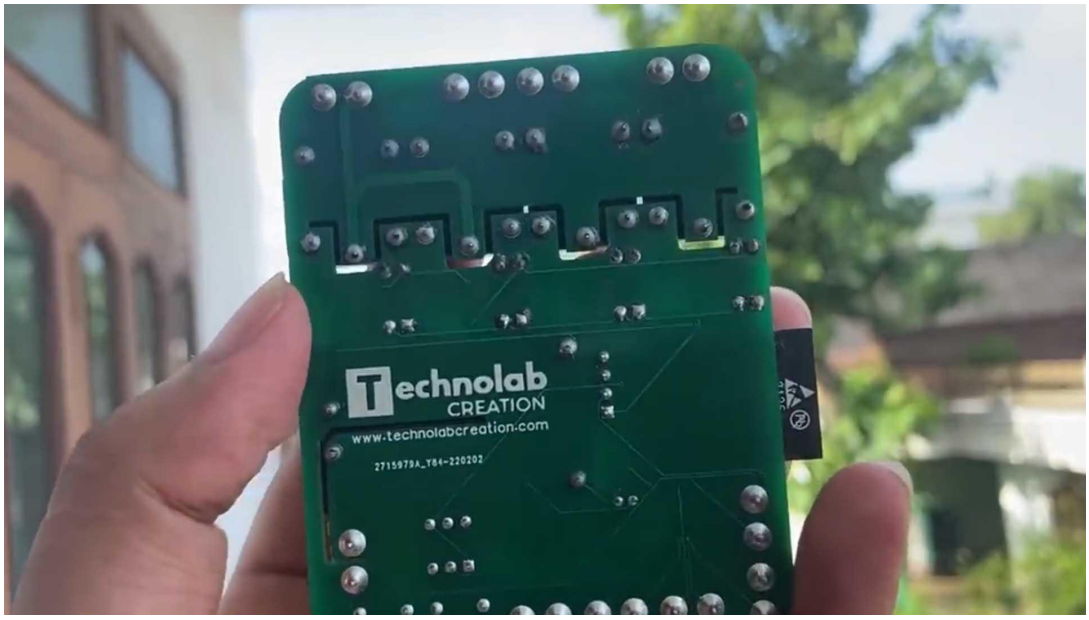
module. Now click on the "Partition Scheme" and select "Trendmaker."



Now select the right port. Here, we are good to go. Now click the "Upload" button. Once the code is successfully uploaded, open the serial monitor. Now press the reset button on the PCB for five seconds. As you can see, a QR code is printed on the serial monitor. I have to scan this QR code, but this QR code is not clearly visible. To view this QR code clearly, copy this link and open it in your browser. Now with this QR code, we can easily enter the Wi-Fi credentials to the ESP 32 chip using the ESP RainMaker app. And this ESP RainMaker app is available for both Android and iOS. Now open this app and click on "Add Device." Scan the QR code. It will take a few seconds to connect with the ESP 32 chip. Now select your Wi-Fi network and enter the password of your Wi-Fi. It will take a few sequences to configure the Wi-Fi credentials in ESP 32. As you can see, all the devices are successfully added. Now tap on "Done." Here, both the devices that we have defined in the code are added and ready for control.

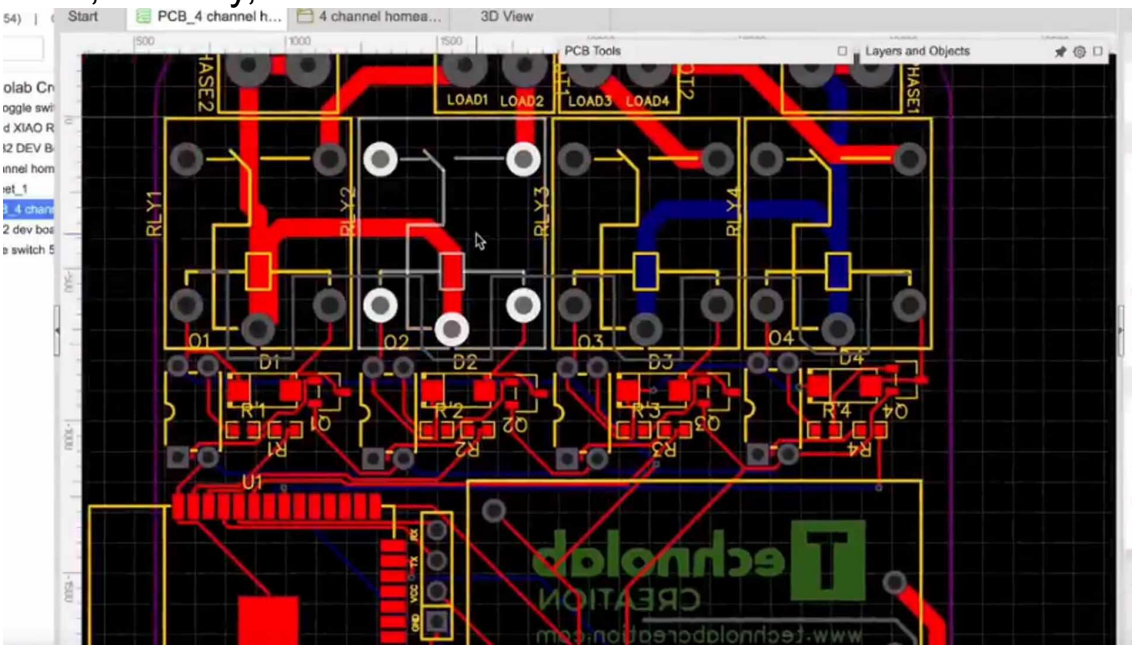
AUTOMATION SYSTEM WITH MANUAL CONTROL

"Alexa, all lights off." Hey, hello friends, welcome to another project. Nowadays, almost everyone has smart speakers in their houses, like Amazon Alexa or Google Assistant. I have Amazon Alexa in my house. The quality of the Voice Assistant is amazing. You can ask anything to Alexa, and Alexa will give responses in no time. Talking to Alexa is always fun. Apart from this, we can also use Alexa to automate our home appliances. That means we can control our home appliances by just giving voice commands to Alexa. Just like this, "Alexa, all lights on." "Alexa, light one off." "Alexa, light two off." And apart from this, we can also control these appliances using manual switch buttons that we traditionally do. We can also control these appliances using the Alexa smartphone application. In this project, we will learn how to make a home automation system using Alexa and ESP 32.



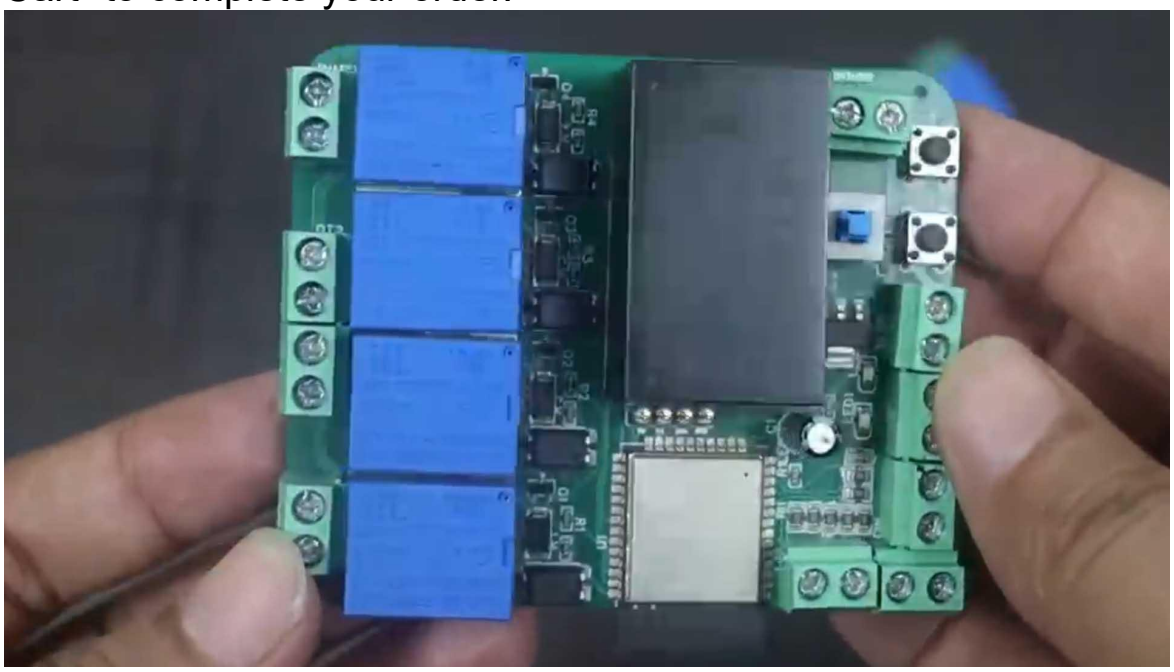
To make this project, I will use my 4-node safety home automation PCB. Apart from this 4-node PCB, I also have eight-node and two-node home automation PCBs. These PCBs are fully tested and work very

well. These PCBs have all the safety features like A/C electrical isolation. So if you want to purchase these PCBs, just check the link given in the project description. During the project, I will explain the code, circuitry, and connection of bulbs and switches.



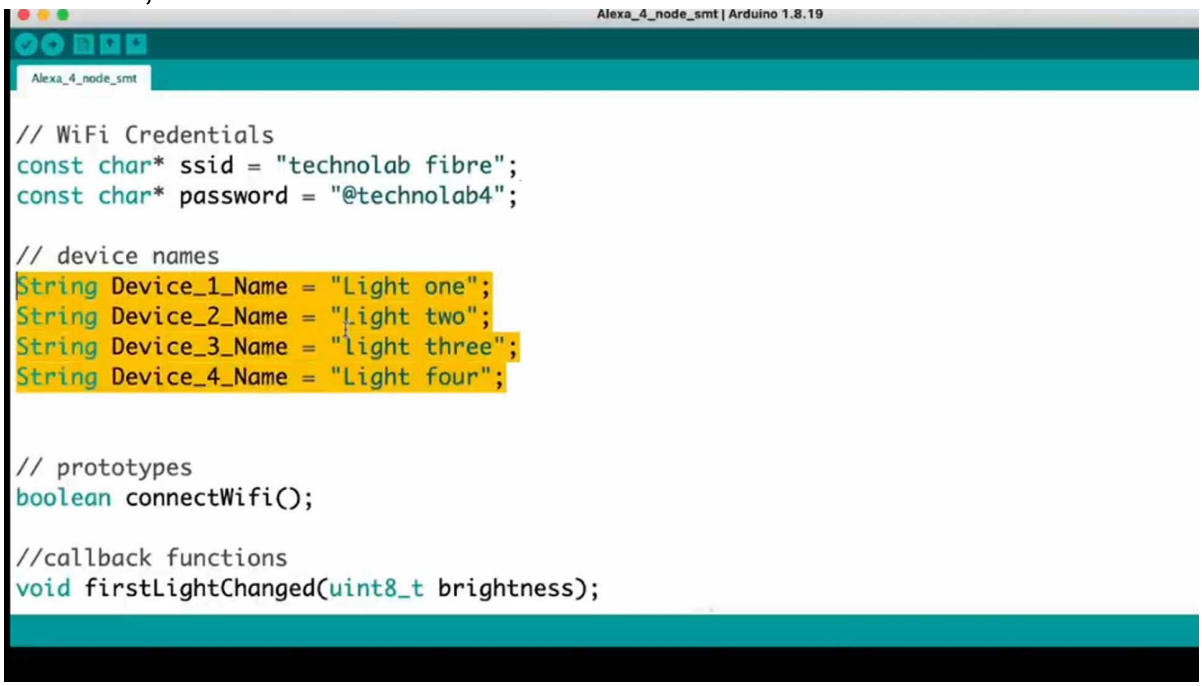
So, I will recommend that you watch the complete project till the end. Now, let's get into this project. This is the schematic of the PCB. If you want your own custom-designed PCB, then you can download this schematic from the link given in the project description. After making the schematic, convert it into a PCB, arrange and place all the components in desirable places. Once the layout is ready, route the wiring and complete the design of the PCB. After the completion of the PCB design, you need to download 3 files which will be required during the PCB order. These files are BOM, Gerber, and Cpl (pick-and-place) files. Now, open the JLC PCB website and click on the "Quote Now" button under PCB assembly. Click here to upload the Gerber file of your PCB. Here, JLC PCB will automatically set all the parameters of the PCB. Select the PCB quantity and color masking of the PCB by yourself. Scroll down below and select "PCB Assembly." Here, you have to select on which surface you want the PCB assembly, either on the top side or bottom side. In my case, it is the top side. After that, click on the "Confirm" button. For PCB assembly, we need two more files. One is BOM (bill of materials), and the second one is Cpl (pick-and-place) file. Upload these two

files one by one. Here, you need to give a description about your PCB, for which purpose you want to make this PCB. I am giving it as "ESP 32 Home Automation." After that, click on the "Next" button. All the components were shown here that are to be assembled. In case you want to not assemble a particular component, then you can deselect that component. After checking all the components, click on the "Next" button. Here, you will see all the components that have to be assembled by JLCPCB along with pricing. Now click on "Save to Cart" to complete your order.

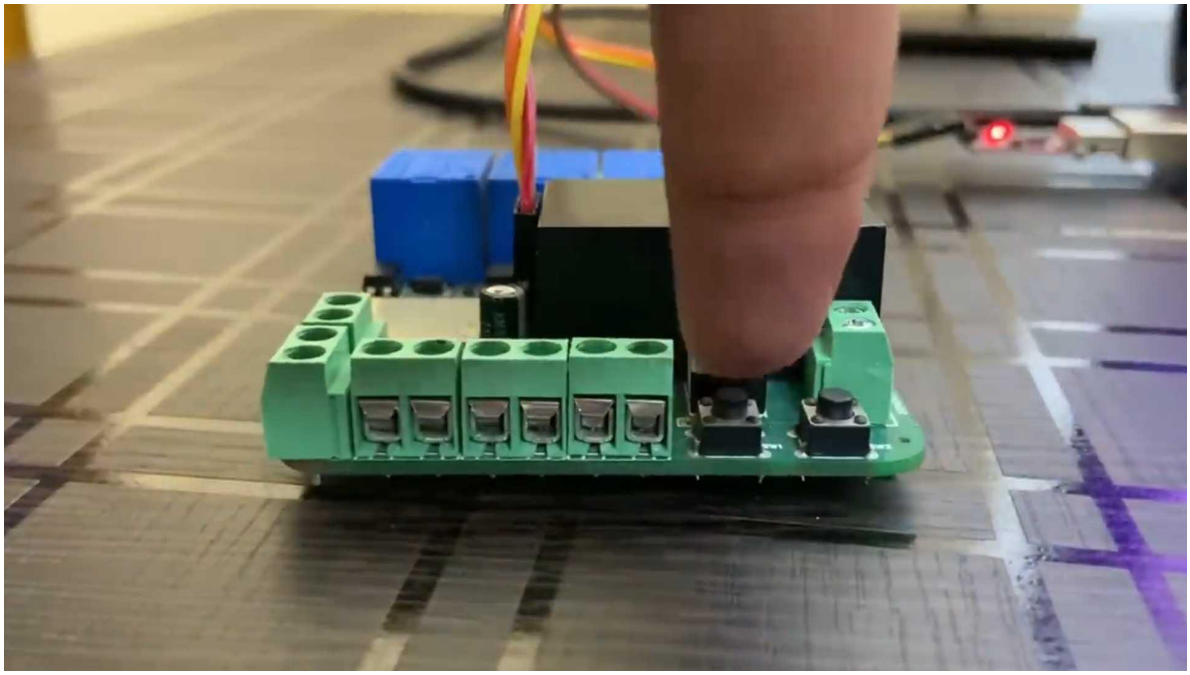


After seven days, PCBs arrived at my place. As usual, the quality of the PCBs is very premium, and the components are soldered very well. To upload the code into the ESP 32 chip, I will use a USB to TTL converter. Connect the PCB to the USB to TTL converter as per this circuit diagram. In case you don't have a USB to TTL converter, you can upload the code using the ESP32 Development Board. Just make the connections as per this circuit diagram, and the rest of the process is the same. This is the code for this home automation project. Before you upload the code, you need to add the ESP Alexa and ACE button libraries in your Arduino IDE. To add these libraries, go to "Tools" and click on "Manage Libraries." A pop-up will come up. Here, in the search box, type "ESP Alexa" and install this library after selecting the latest version. I have already installed this library, so I am leaving it. After installing this library, search for the "AceButton"

library by typing "AceButton" in the search box. Now install this library also by selecting the latest version of it. After installing both libraries, close this window.



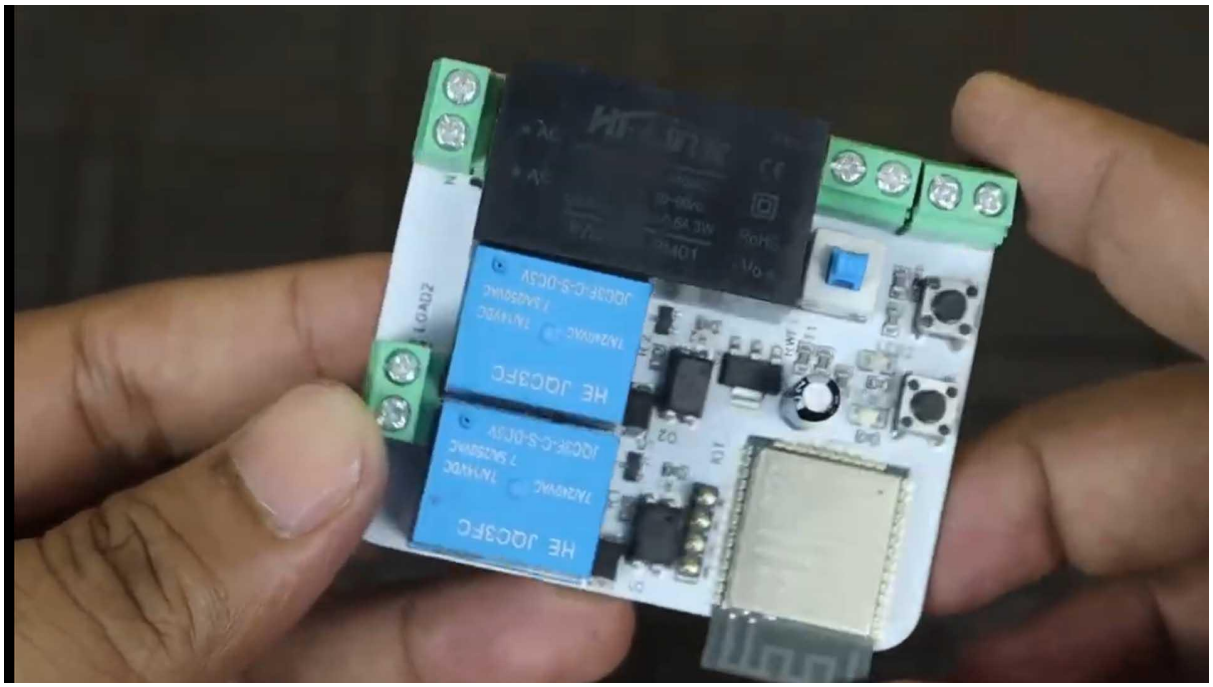
Here, in this section of the code, you have to enter the SSID and password of your router or hotspot. And here, we have defined a total of 4 devices and also we give the name of each device. These names will appear in the Alexa app when we connect the ESP 32 with Alexa. After that, the code will remain the same. No need to change anything. Just download this code from the project description and make the changes that I just told you and straightforwardly upload this code after selecting the right board and COM port. While uploading the code, press and hold the boot button and press the reset button once to make this module go into the boot mode. Download the Amazon Alexa app on your smartphone. This app is available for both iOS and Android. In the app, tap on "Devices," then tap on the plus icon on the top right corner of the app to add devices.



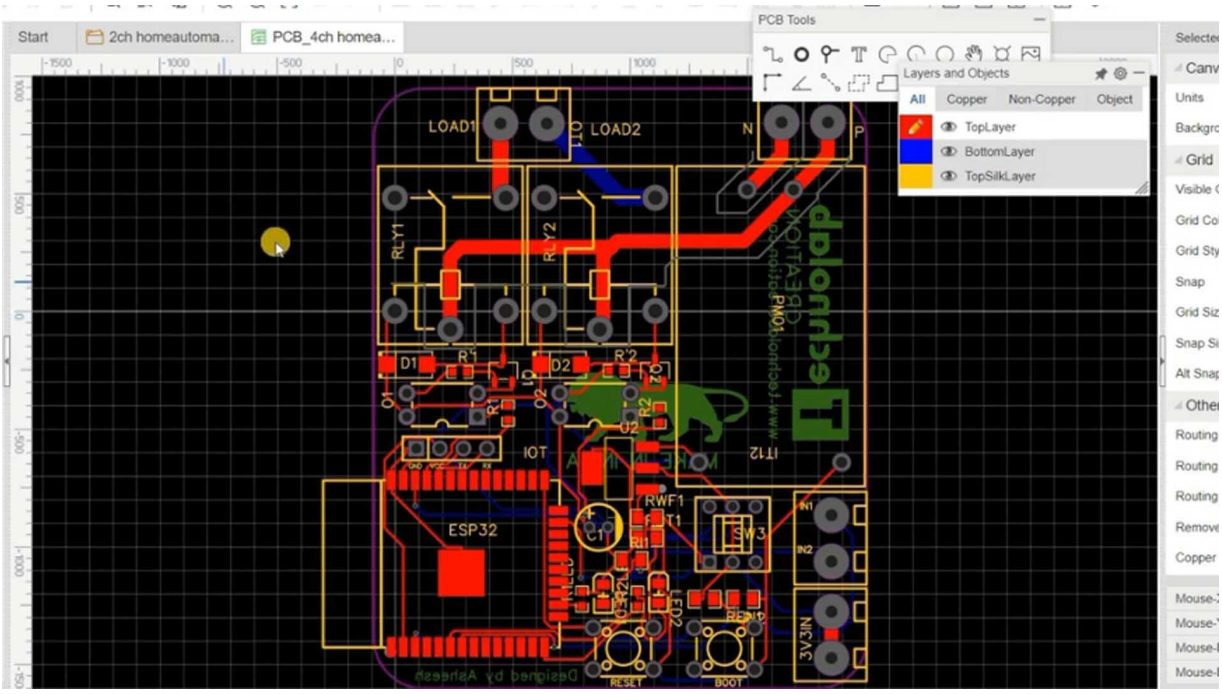
Here, you scroll down and select "Other." Here, click on "Discover Devices." Here, the Alexa speaker is searching for the devices. It will take a few seconds to find the devices. Here, you can see Alexa found a total of 4 devices. Now tap on the "Next" button. Here, a total of 4 devices are connected to Alexa. Make the connections of all the bulbs and switches as per this schematic diagram. Now, everything is done. Let's see the project in action.

ANDROID APP BLUETOOTH CONTROLLED HOME-DEVICES USING ESP32

Hello, friends! Welcome to another home automation project project. In this project, we are going to make an Android app-controlled home automation system using Bluetooth. Now, we can control our home appliances through our smartphone using the Bluetooth feature of the ESP32. Apart from this, we can also control our devices using manual switch buttons that we conventionally do. For the making of this home automation system, I will use my two-node SMT home automation PCB and a custom-designed Android app. And the best part of this project is that we don't need any Internet connection or local server to connect the Android application to ESP32. This is especially useful for people who don't have an Internet connection, making this home automation system perfect for them. The app will directly communicate with the ESP32 via Bluetooth. Using Bluetooth is a very convenient way of creating a home automation system for local control.

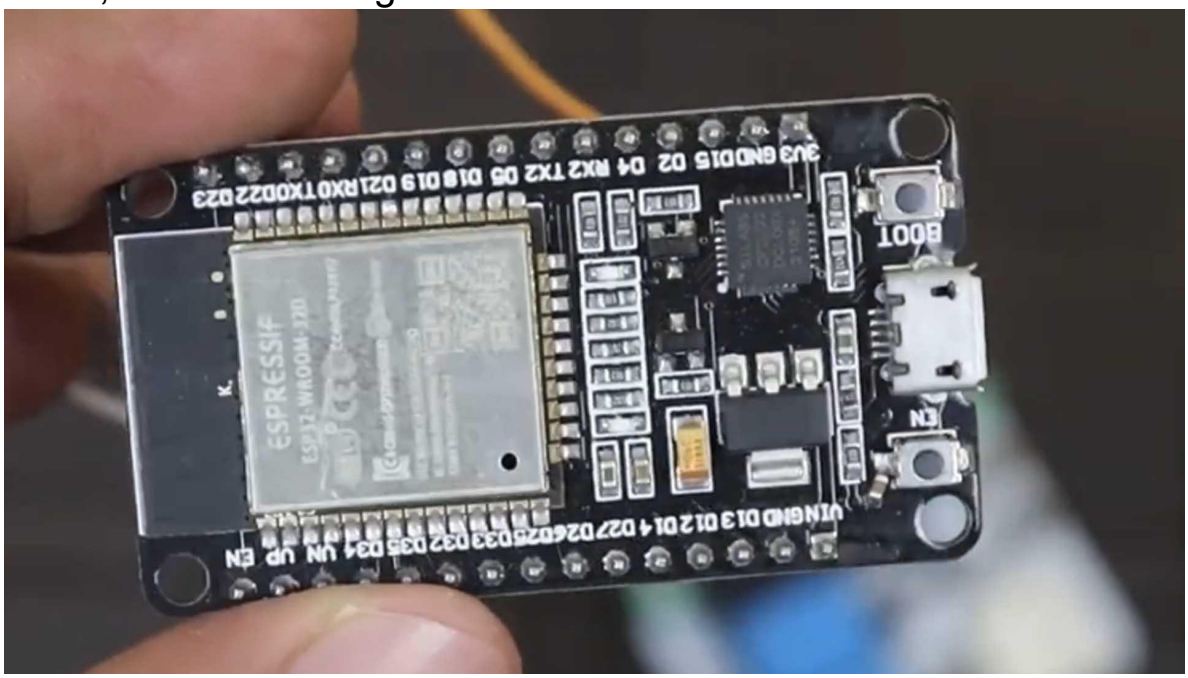


And the only downside of this project is that it is limited to a local range. Apart from this two-node version, 4-node and 8-node versions of the home automation PCB are also available. If you want to purchase this PCB, please check the project description. All the necessary links are available there. Now, let's get into this project. This is the schematic of today's home automation PCB. If you want, you can download this schematic from the description to design your own custom PCB.



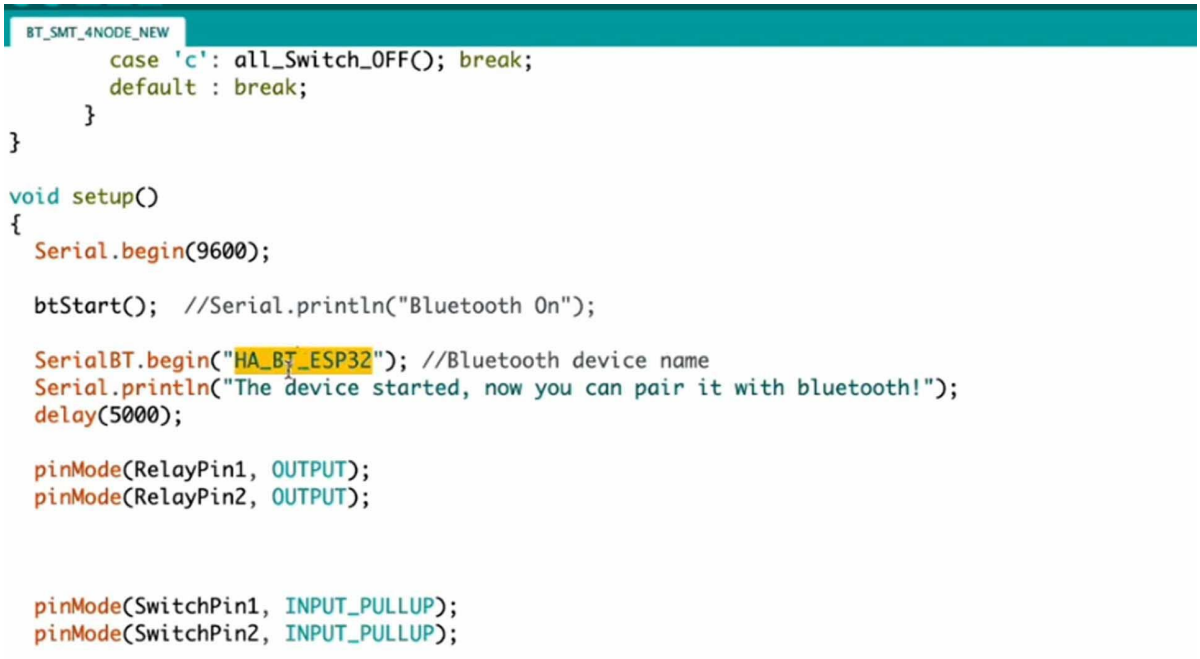
Now, convert this schematic into a PCB. After completing the design of your PCB, you can directly order the PCB from JLC PCB, or you can download the Gerber file from here. After that, go to the JLC PCB website and click on the "Quote Now" button under the PCB assembly section. Upload the Gerber file of your PCB. After that, select the number of PCBs and the color masking of the PCB if you want. Then select the PCB assembly service, and here, you have to choose whether your components will be soldered on the top or bottom surface of the PCB. After that, click on the "Confirm" button. Now, here, you need to upload two more files: one is the Cpl (pick-and-place) file, and another one is the BOM (Bill of Material) file. You can download these files from your EasyEDA account. Just open the PCB project on your EasyEDA account, go to Fabrication, then BOM. Click on "Export BOM" to download the BOM file. Similarly, download the Cpl file. After downloading both files, upload them here onto this page. Then select "Next." Now, here, it will show all the components that are to be soldered. You can also select which components will be soldered or not. Select the components according to your preference. After that, click on the "Next" button, and then click "Save to Cart" to complete your order. After a week, my PCB arrived at my place in a new blue box from JLCPCB. Let me open the box. The packaging of the PCB in bubble wrap is very

good. Here it is, our home automation PCB. The quality of the PCB is good, and the surface-mounted components are soldered well. After soldering the rest of the components, the PCB looks neat, clean, and well-arranged.



To flash the code into the ESP32 chip, I will use an ESP32 Development Board. Now, make the connections according to this schematic. This is the code for today's project. Download this code from the link given in the project description. Before you upload the code, you need to make some changes in the code. First of all, you need to add the ESP32 boards in your Arduino IDE. Also, you need to add the AceButton library in your Arduino IDE to run this code. To do this, go to "Tools," then "Manage Libraries." A pop-up will come up. Type "AceButton" in the search box. Now install this library by selecting the latest version of it. I have already installed this library, so I am skipping it. After installing the library, close this window. The rest of the code is okay; no need to change it. There is one little change you could make here in the section of the code that specifies the Bluetooth device name. The name we give here will be the name of the Bluetooth device, and this name will appear when we pair the ESP32 with our smartphone. Now, upload this code after selecting the right board and COM port. After clicking the upload button, I will press and hold the boot button and press the reset button once to

make this module go into boot mode. As you can see, the code starts uploading. I have made this custom Android app using Code Ruler.



I will put the AIA and APK files of this app in the project description. If you want, you can download and customize it according to your needs. Download the APK file of this app and install it on your Android smartphone. After installing the app, open the Bluetooth settings of your phone. Click on "Pair new device." A Bluetooth device with the same name that we mentioned in the code will appear. Now tap on this device and click on "Pair" to pair the ESP32 with our smartphone.

EXAMPLE DUMMY CODE

Creating a complete Android app for Bluetooth-controlled home devices using an ESP32 involves both Android app development and programming the ESP32. It's a relatively complex project, so I'll provide a simplified example outline and code snippets to get you started. You will need to have some knowledge of Android app development using Java or Kotlin and the Arduino IDE for ESP32 programming.

Android App Development (Android Studio):

1. **Set Up Your Android Project:**
   - Create a new Android project in Android Studio.
   - Set up your app's user interface (UI) with buttons, switches, or other controls to control your home devices.

2. **Bluetooth Setup:**
   - Implement Bluetooth connectivity. Request Bluetooth permissions and enable Bluetooth on the Android device.

3. **Device Discovery:**
   - Discover available Bluetooth devices (ESP32) and list them in your app.

4. **Device Pairing:**
   - Pair your Android device with the ESP32.

5. **Bluetooth Communication:**
   - Establish a Bluetooth connection with the ESP32 and send commands to control your devices.

6. **User Interface (UI):**
   - Update the UI to reflect the status of your home devices and provide controls for turning them on/off or adjusting settings.

7. **App Logic:**
   - Implement the app's logic to send commands to the ESP32 based on user interactions.

Here's a simplified example of Android app code for Bluetooth control (Java):

```java
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

import java.io.IOException;
import java.util.UUID;

public class MainActivity extends Activity {
    Button connectBtn, onBtn, offBtn;
    BluetoothAdapter bluetoothAdapter;
    BluetoothDevice bluetoothDevice;
    BluetoothSocket bluetoothSocket;

    private static final String DEVICE_ADDRESS =
"00:00:00:00:00:00"; // Replace with your ESP32's Bluetooth MAC address
    private static final UUID UUID_SERIAL =
UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
        connectBtn = findViewById(R.id.connectBtn);
        onBtn = findViewById(R.id.Btn);
        offBtn = findViewById(R.id.offBtn);

        bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

        connectBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                if (!bluetoothAdapter.isEnabled()) {
                    Intent enableBT = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                    startActivityForResult(enableBT, 1);
                } else {
                    bluetoothDevice =
bluetoothAdapter.getRemoteDevice(DEVICE_ADDRESS);
                    try {
                        bluetoothSocket =
bluetoothDevice.createRfcommSocketToServiceRecord(UUID_SERIA
L);
                        bluetoothSocket.connect();
                        Toast.makeText(getApplicationContext(),
"Connected to device", Toast.LENGTH_SHORT).show();
                    } catch (IOException e) {
                        Toast.makeText(getApplicationContext(),
"Connection failed", Toast.LENGTH_SHORT).show();
                    }
                }
            }
        });

        onBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                sendCommand("ON");
            }
```

```
        });

        offBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                sendCommand("OFF");
            }
        });
    }

    private void sendCommand(String command) {
        if (bluetoothSocket != null) {
            try {
                bluetoothSocket.getOutputStream().write(command.get
Bytes());
                Toast.makeText(getApplicationContext(), "Sent: " +
command, Toast.LENGTH_SHORT).show();
            } catch (IOException e) {
                Toast.makeText(getApplicationContext(), "Error sending
command", Toast.LENGTH_SHORT).show();
            }
        }
    }
}
```

ESP32 Programming (Arduino IDE):

Here's a simplified example of Arduino code for ESP32 to receive commands over Bluetooth and control a device (e.g., an LED):

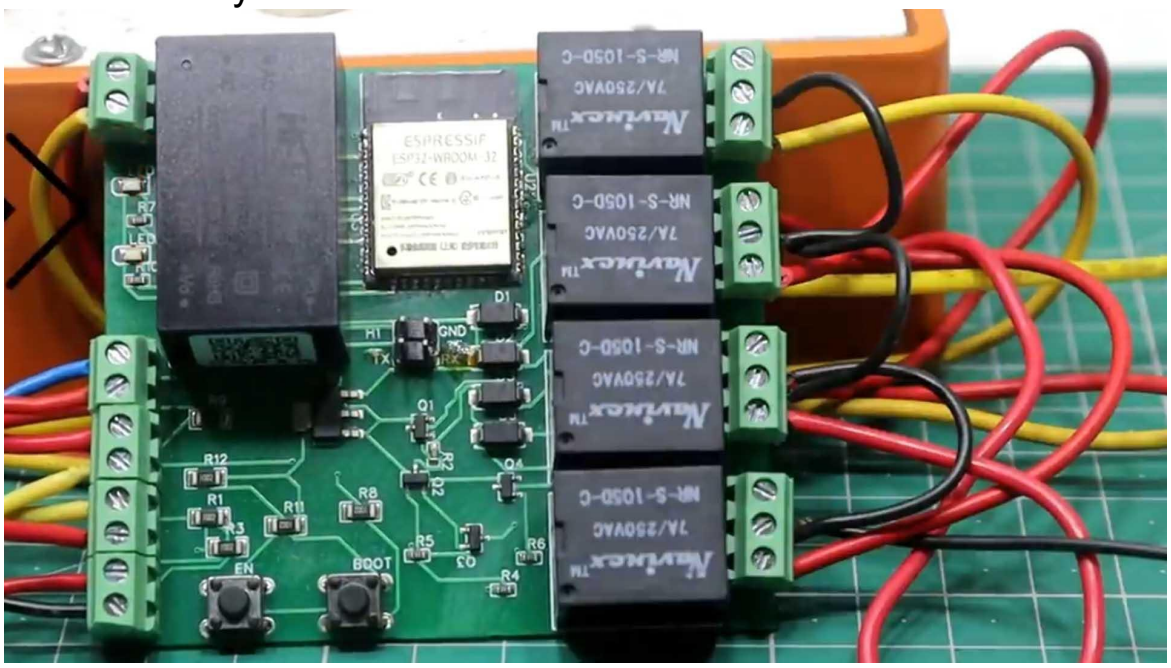
```cpp
#include <BluetoothSerial.h>

BluetoothSerial SerialBT;

void setup() {
  Serial.begin(115200);
```

```
  SerialBT.begin("ESP32_BT"); // Bluetooth device name

  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);
}

void loop() {
  if (SerialBT.available()) {
    char cmd = SerialBT.read();
    if (cmd == 'ON') {
      digitalWrite(LED_BUILTIN, HIGH); // Turn the LED on
    } else if (cmd == 'OFF') {
      digitalWrite(LED_BUILTIN, LOW); // Turn the LED off
    }
  }
}
```

Please note that this is a simplified example to get you started. In a real-world scenario, you would have to handle exceptions, error recovery, and possibly implement more secure communication between the app and ESP32, especially if you're controlling critical home devices.

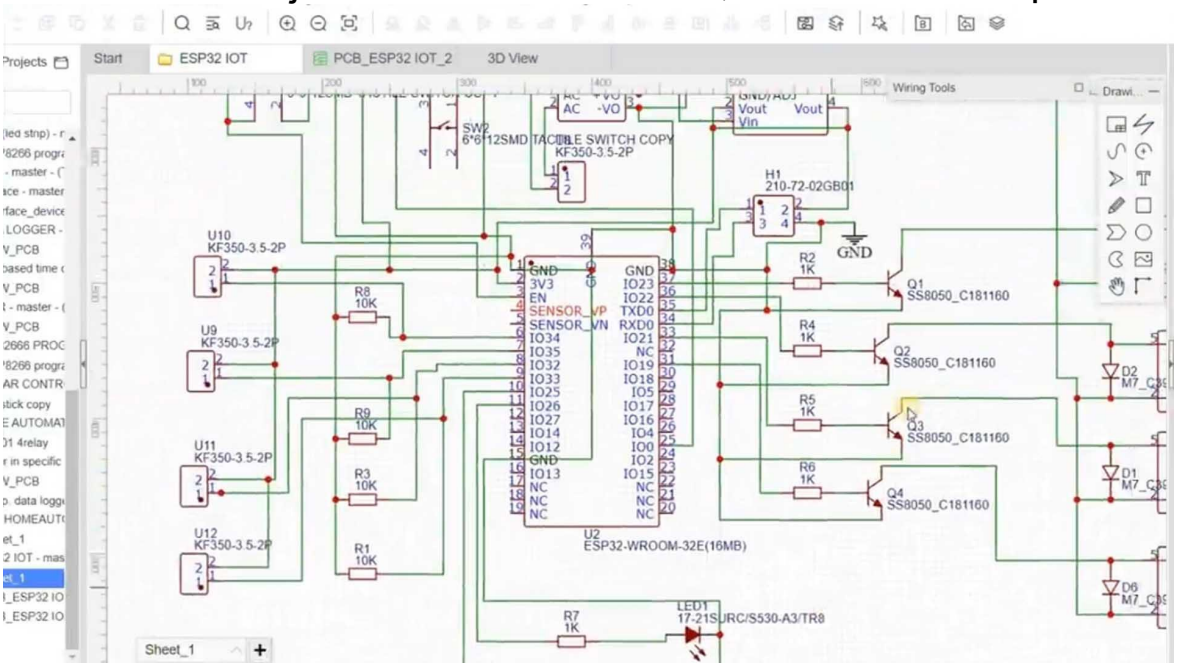
BEST HOME AUTOMATION PCB WITH SMD COMPONENTS_ ESP32 CHIP

Hey, hello friends! Welcome to another project. Now in this project, I will introduce a home automation PCB, which is the smallest home automation PCB I have ever used. By using the ESP32 chip and the summit components, I am able to reduce the size of the PCB. With this PCB, we can create an Internet and manual control home automation system.

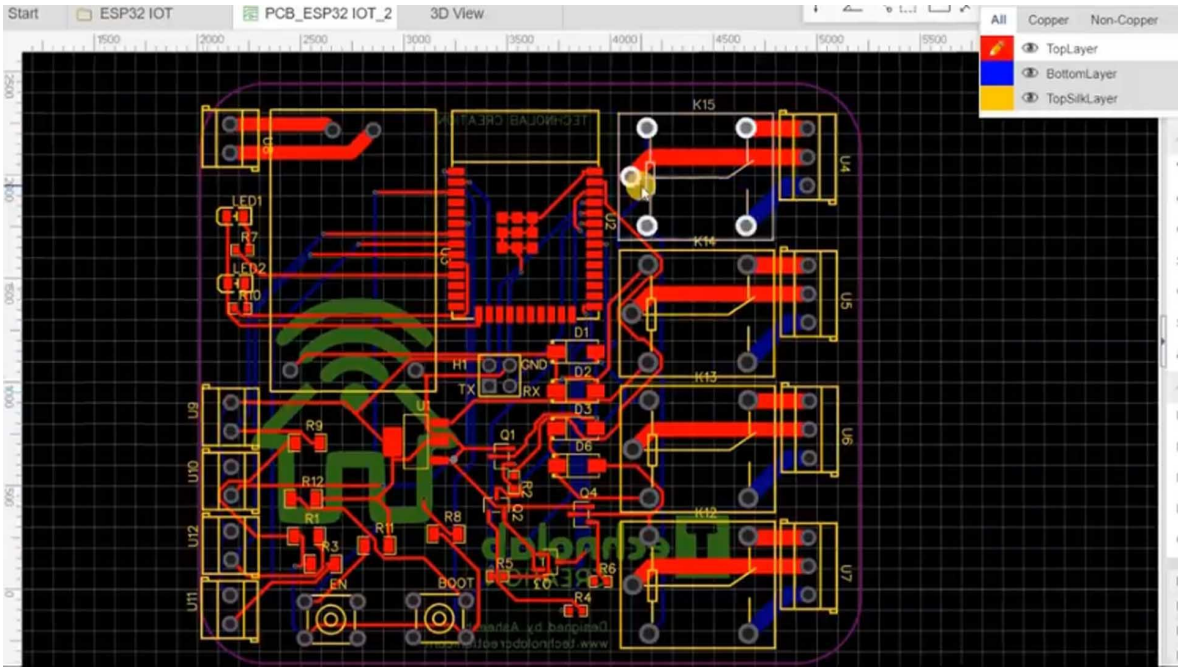


We can control our appliances via the Blink app as well as through manual switches. Additionally, we can monitor the real-time status of appliances in the Blink app. The two onboard LEDs are used for Wi-Fi status. If the ESP32 is connected to Wi-Fi, both LEDs will glow. But if Wi-Fi is not available, only a single LED will glow. Let me show you. Now, I am turning off the hotspot. Here, you can see only one LED is turned on. Let me turn on the hotspot again. Now you can see both LEDs are glowing, indicating that the ESP32 is connected

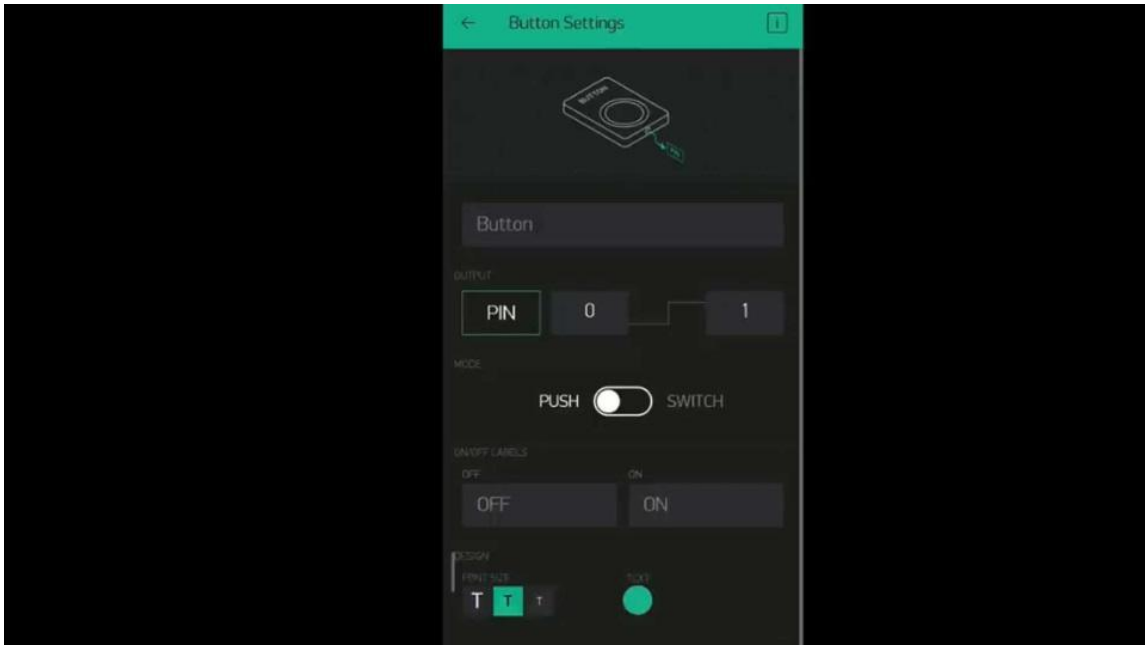
to Wi-Fi. This project is sponsored by JLCPCB. JLCPCB is a well-known PCB prototype company in China. It specializes in quick PCB prototype and small-batch production. You can now order a minimum of five PCBs for just $2. For more details, check the description.



This is the schematic of today's home automation PCB. If you want, you can download this schematic from the description to design your own custom PCB. Now, convert this schematic into a PCB. After completing the design of your PCB, you can directly order the PCB from JLC PCB or just download the Gerber file from here. After that, go to the JLC PCB website, then click on the "Quote Now" button under the summit assembly section. Upload the Gerber file of your PCB. After that, select the number of PCBs and color masking of the PCB if you want. Then select the summit assembly service, and here, you have to select on which surface you want your components to be soldered, either the top surface or bottom. After that, click on the "Confirm" button. Now, here, you need to upload two more files: one is the Cpl (pick-and-place) file, and another one is the BOM (Bill of Material).



You can download these files from your EasyEDA account. Just open the PCB project on your EasyEDA account, then click on "Fabrication," then "BOM." Click on "Export BOM" to download the BOM file. Similarly, download the Cpl file. After downloading both files, upload them here onto this page. Then select "Next." Now, here, it will show all the summit components that are to be soldered, and you can also select which components will be soldered or not. Select the components according to your preference. After that, click on the "Next" button, and then click "Save to Cart" to complete your order. After a week, my PCB arrived at my place in a new blue box from JLCPCB. Let me open the box. The packaging of the PCB in bubble wrap is very good. Here it is, our home automation PCB. The quality of the PCB is good, and the summit components are soldered well. The summit assembly service of JLCPCB is great. After soldering the rest of the components, the PCB looks neat, clean, and well-arranged.

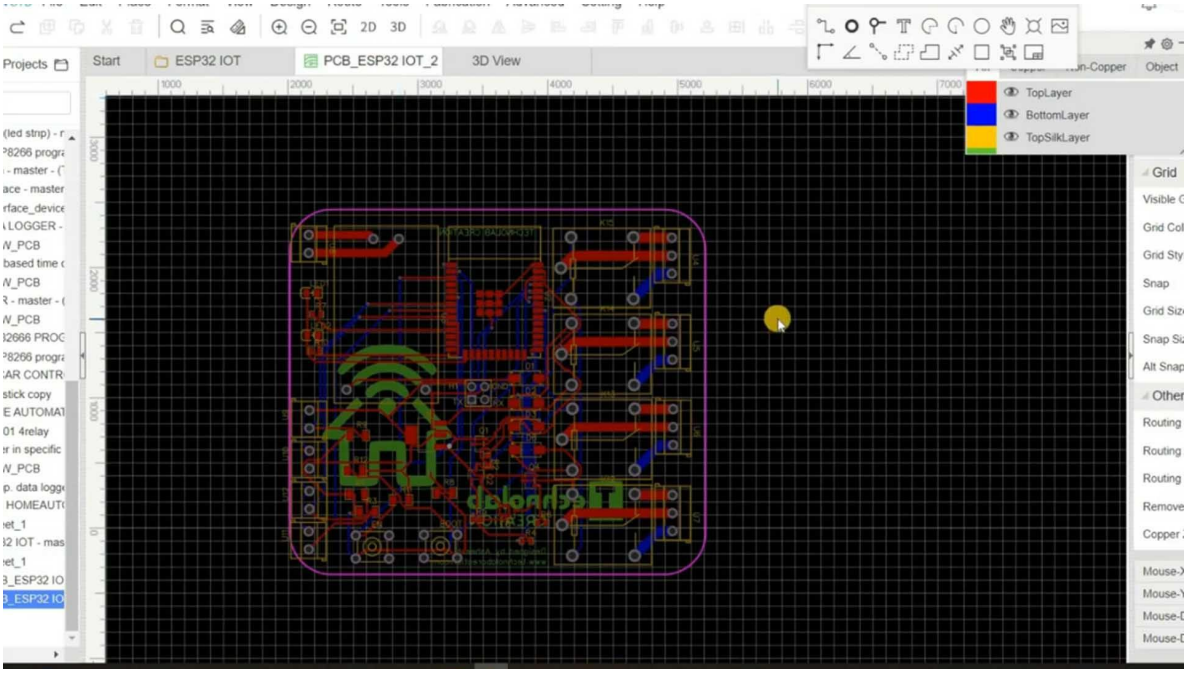


Now, open the Blink app. Click on "New Project." Give it any name; I'm calling it "Home Automation." Select the board as ESP32 Development Board and connection type as Wi-Fi. Click OK. An authentication token will be sent to your email ID, which you'll require for coding. After that, click here and add a button. Likewise, add three more buttons. After that, set up a button for configuration, give it a name like "Relay One." Select the pin as virtual, select virtual pin V1, change 0 to 1 and 1 to 0 for reverse logic, and select the mode as switch. That's it. Likewise, to configure all the buttons To flash the code into the ESP32 chip, I will use an ESP32 Development Board. Now, make the connections according to this schematic. This is the code for our today's home automation project. Download this code from the description and open it in the Arduino IDE. Before you upload the code, you need to make a few changes in it.

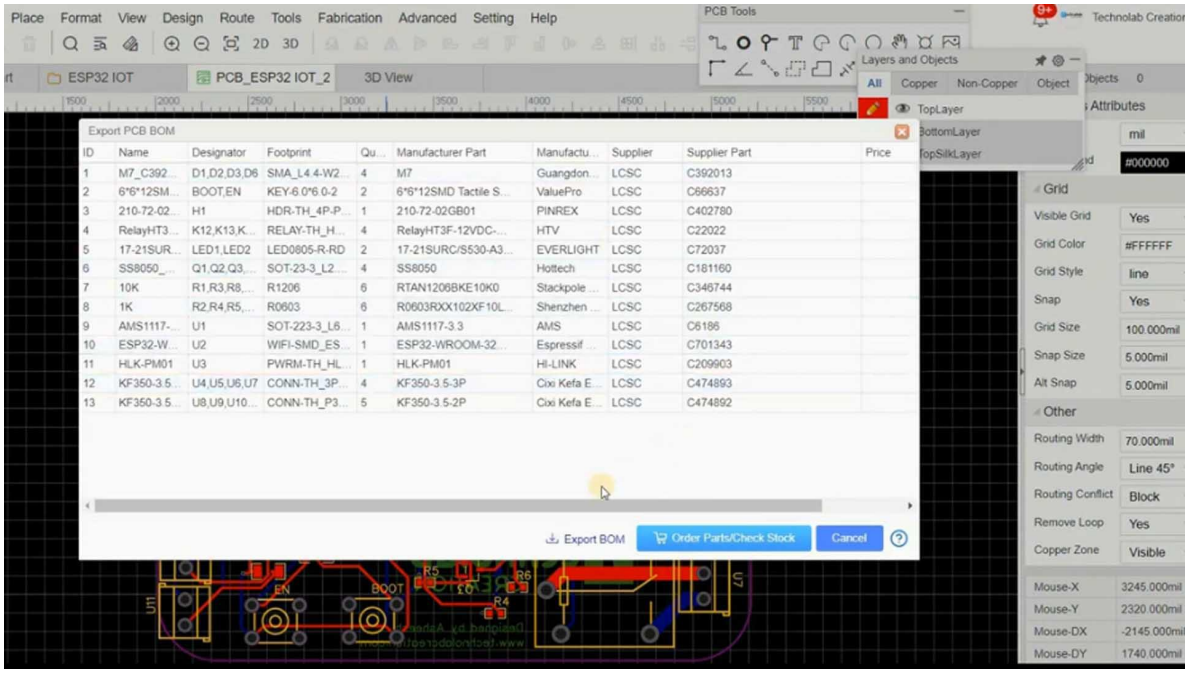
First, in this section, you need to enter the SSID and password of your router or hotspot. After that, here, you have to enter the authentication token sent by Blink to your registered email ID. Just copy and paste it here, and the rest of the code is okay. After selecting the right board and COM port, hit the upload button. After clicking the upload button, on the PCB, I will press and hold the boot button and press the reset button once to make this module go into boot mode. As you can see, the code starts uploading. Now, connect all the bulbs and switches according to this circuit diagram. Now, everything is done. Let's see the project in action.

BIOMETRIC FINGERPRINT DOOR LOCK CONTROL

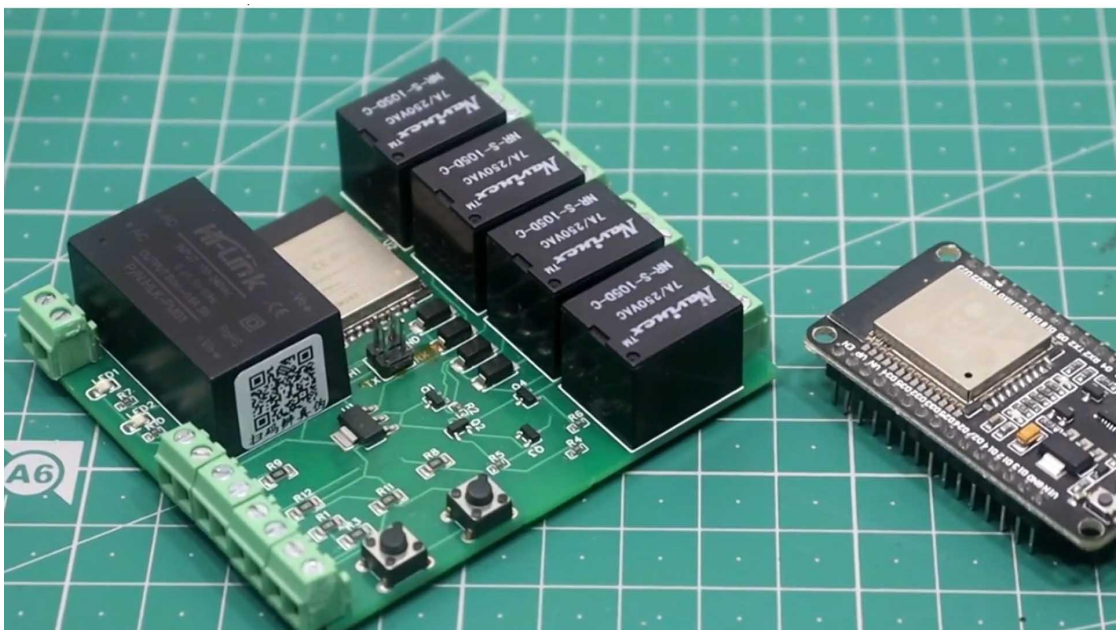
Hey, hello friends! Welcome to another project. In this project, we are going to make a home automation system in which we are able to control our appliances through a smartphone. This project has one more interesting addition, that is a biometric door unlocking system. Now, we can easily unlock the door using our fingerprint through the CM app from which we control our home appliances. So, we don't need separate apps. To unlock the door, we will use our smartphone's fingerprint sensor. Let me show you how this will work. To unlock the door, just put your registered finger on the smartphone sensor. As you see, the solenoid door lock is unlocked by biometric fingerprint. If, in any case, an unauthorized person tries to unlock the door, their fingerprint will not be recognized by the app. Consequently, they are not able to unlock the door. As I told you, we can control our appliances through a smartphone. Apart from this, we can also control our appliances through switch buttons. So, without any further delay, let's get started with this project. This project is sponsored by JLCPCB. JLCPCB is a well-known PCB prototype company in China. It specializes in quick PCB prototype and small batch production. You can now order a minimum of five PCBs for just $2. For more details, check the description.



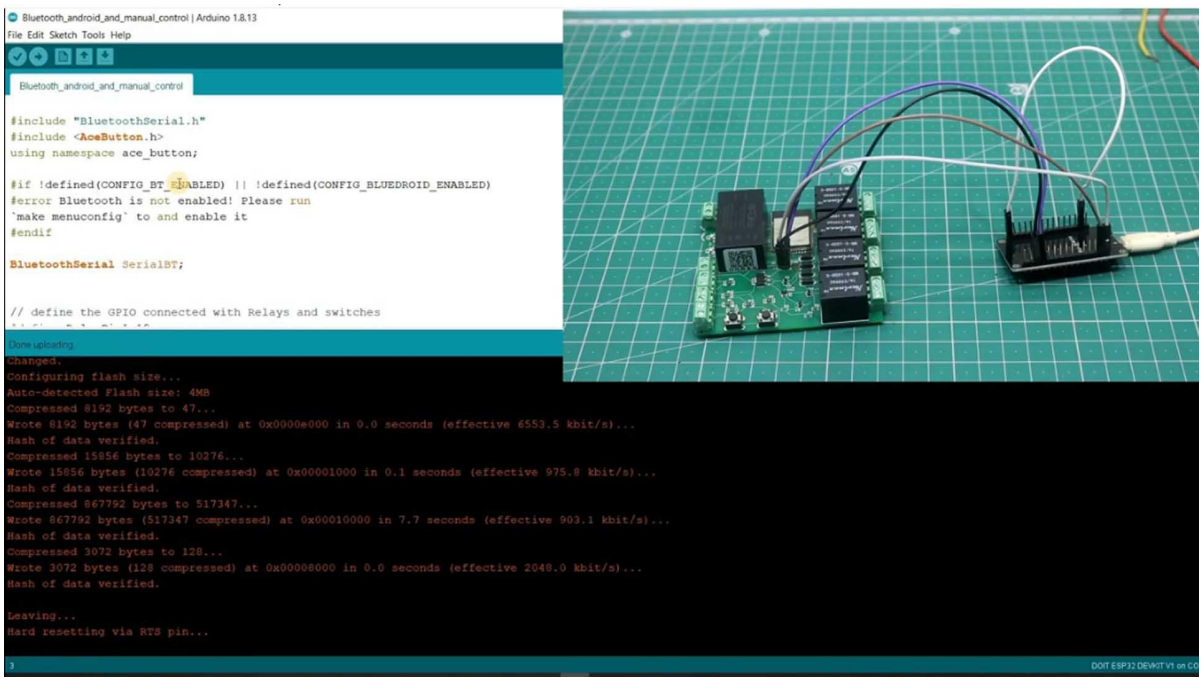
This is the schematic of today's home automation PCB. If you want, you can download this schematic from the description to design your own custom PCB. Now, convert this schematic into a PCB. After completing the design of your PCB, you can directly order the PCB from JLC PCB or just download the Gerber file from here. After that, go to the JLC PCB website, then click on the "Quote Now" button under the summit assembly section. Upload the Gerber file of your PCB. After that, select the number of PCBs and color masking of the PCB if you want. Then select the summit assembly service, and here, you have to select on which surface you want your components to be soldered, either the top surface or bottom. After that, click on the "Confirm" button. Now, here, you need to upload two more files: one is the Cpl (pick-and-place) file, and another one is the BOM (Bill of Material).



You can download these files from your EasyEDA account. Just open the PCB project on your EasyEDA account, then click on "Fabrication," then "BOM." Click on "Export BOM" to download the BOM file. Similarly, download the Cpl file. After downloading both files, upload them here onto this page. Then select "Next." Now, here, it will show all the summit components that are to be soldered, and you can also select which components will be soldered or not. Select the components according to your preference. After that, click on the "Next" button, and then click "Save to Cart" to complete your order. After a week, my PCB arrived at my place in a new blue box from JLCPCB.



Let me open the box. The packaging of the PCB in bubble wrap is very good. Here it is, our home automation PCB. The quality of the PCB is good, and the summit components are soldered well. The summit assembly service of JLCPCB is great. After soldering the rest of the components, the PCB looks neat, clean, and well-arranged. To flash the code into the ESP32 chip, I will use an ESP32 Development Board. Now, make the connections according to this schematic. This is the code for today's project. You don't need to make any changes in this code if you are using the same PCB that I used in the project. One thing, if you want, you can change the name of the Bluetooth device. You can give it any name you want, and the rest of the code is okay. After selecting the right board and COM port, hit the upload button.



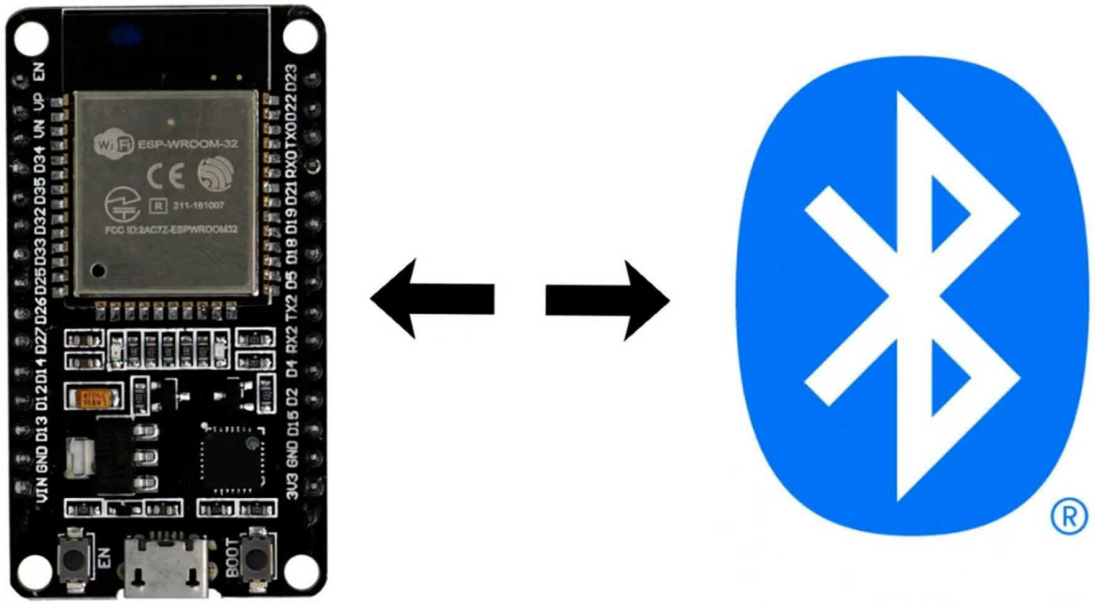
After clicking the upload button onto the PCB, I will press and hold the boot button and press the reset button once to make this module go into boot mode. Connect the solenoid, door lock, and all the bulbs and switches according to this schematic. You can download this schematic from the description, download the APK file of this app from the description, and install it on your phone.

After installing the app, we need to pair the ESP32 with our smartphone. Now open the Bluetooth setting of your phone and pair

with the ESP32. After that, open the app and connect it with the ESP32 via Bluetooth. Now, everything is done. Let's see the project in action.

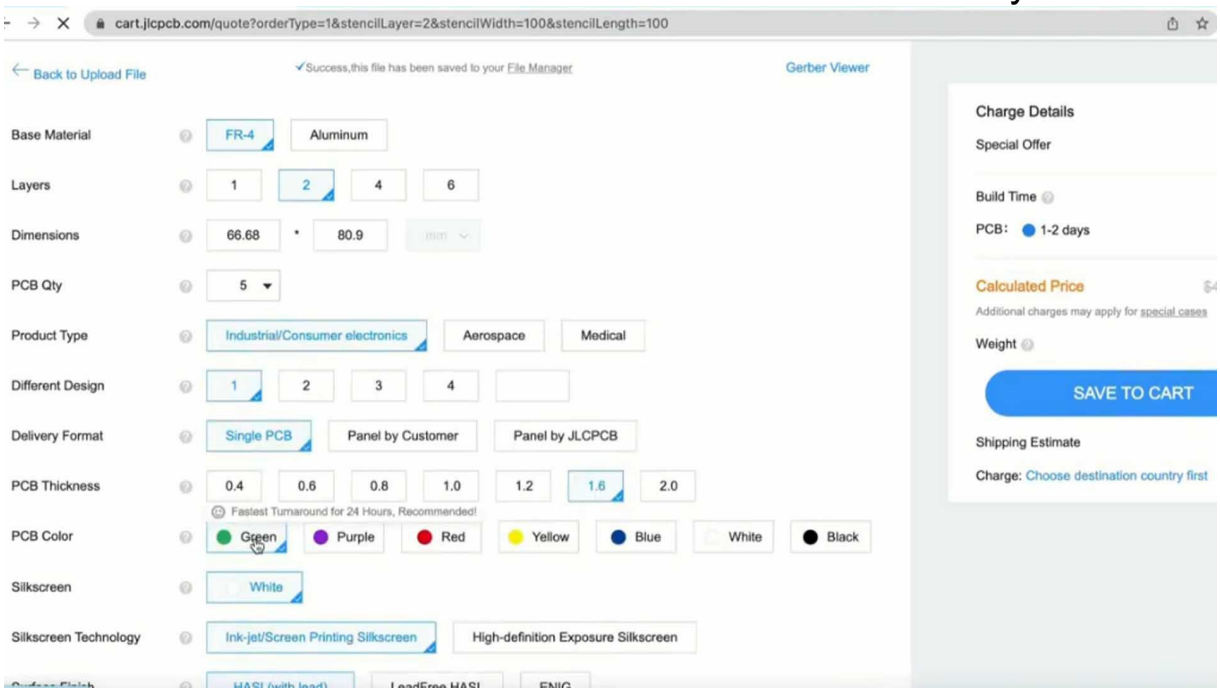
BLUETOOTH _ MANUAL CONTROL HOME AUTOMATION

Hey, hello friends! Welcome to another project. In this project, I am going to make a very useful and very easy home automation system. In this home automation system, we will control our home appliances from the Android app through Bluetooth, and we can also control the appliances via manual switch buttons that we regularly do. Now you are thinking, "We already made this project." Yes, you were right.



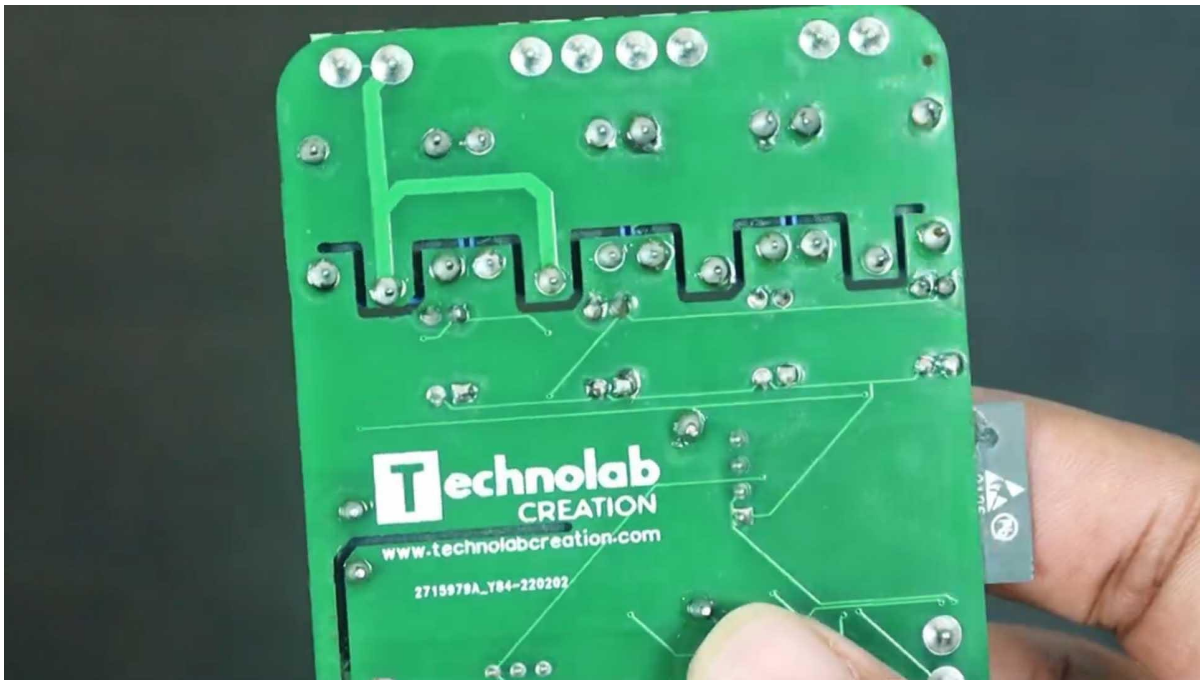
But in this home automation project, we have updated something. Now we can get real-time feedback in the app. Yes, you had that right. Apart from controlling through the app, now we can also get real-time feedback of appliances in the app. To get real-time feedback, we have modified the app. Here I am using the inbuilt Bluetooth connectivity of the ESP32. So, Internet or Wi-Fi is not required in this project. And also, we don't require any IoT or cloud platform. Most people don't have Internet or Wi-Fi connectivity, so

this home automation project is perfect for them. The only downside of this project is that we are using Bluetooth. So in this project, we can only control our appliances in the local area network, not globally. Apart from this, everything is great in this project. This app is absolutely free, and you can download it from the Play Store. I will be linking all the important links down in the description of this project. During the project, I will show you how to pair the app with ESP32, and also I will explain the code, circuitry, and connection of bulbs and switches. So, I recommend that you watch the entire project. Now let's get into this project. This is the schematic of the PCB. If you want your own custom-designed PCB, then you can download this schematic from the link given in the project description. After making the schematic, convert it into a PCB, arrange and place all the components in desirable places. Once the layout is ready, route the wiring and complete the design of the PCB. After the completion of PCB design, you need to download three files which you will require during the PCB order. These files are BOM, Gerber, Cpl (pick-and-place file). Now open the JLCPCB website and click on the "Quote Now" button under PCB assembly.



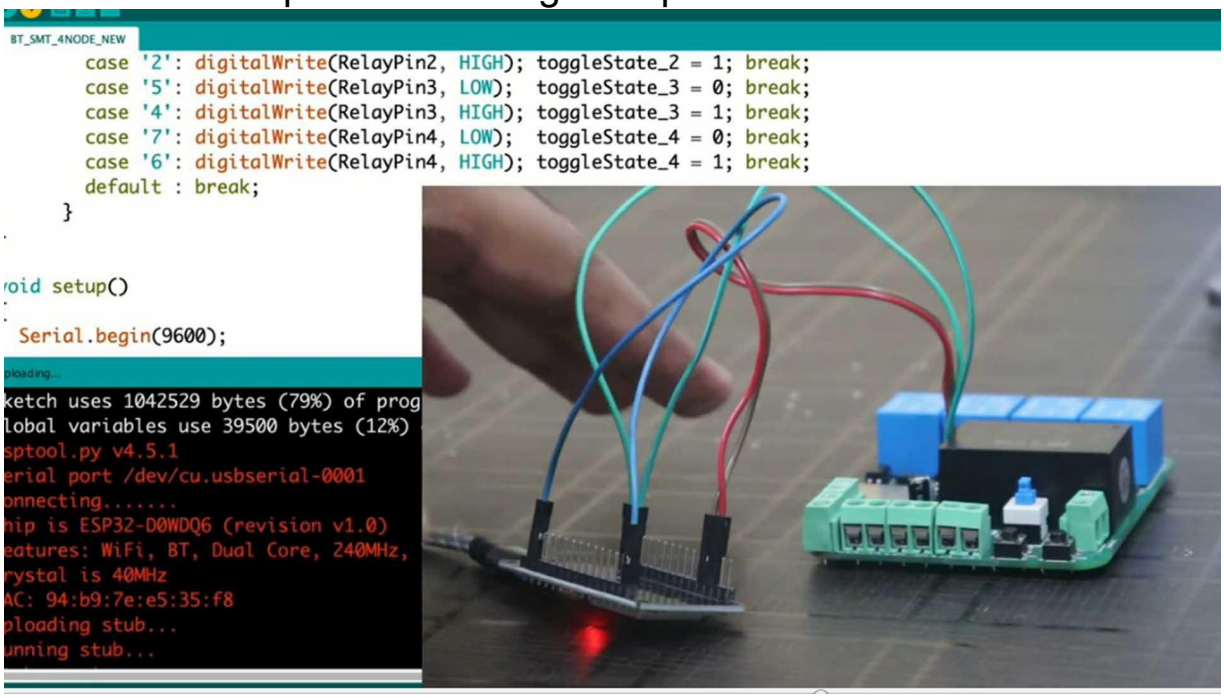
Click here to upload the Gerber file of your PCB. Here, JLCPCB will automatically set all the parameters of the PCB. Select the PCB quantity and color masking of the PCB by yourself. I am selecting the

green color. Scroll down below and select the PCB assembly options. Here you have to select on which side you want to do PCB assembly—top side or bottom side, or both sides. In my case, I want only the top side. After that, click on the "Confirm" button. Now, for PCB assembly, we need two more files. One is BOM (Bill of Material), and the second one is Cpl (pick-and-place file). Upload these two files one by one. Here, you need to give the description about your PCB for which purpose you want to make this PCB. I am giving here that the 4-node is empty. After that, click on the "Next" button. All the components were shown here that are to be assembled. In case you want to not assemble a particular component, then you can deselect that component. After checking all the components, click on the "Next" button. Here, you will see a computer version of component placement which seems not accurate. This is only for reference purposes now. Click on "Save to Cart" to complete your order. After seven days, PCBs arrived at my place. As usual, the quality of PCBs is very premium and the components are soldered very well. Traces are perfect. Silk screen is fine.



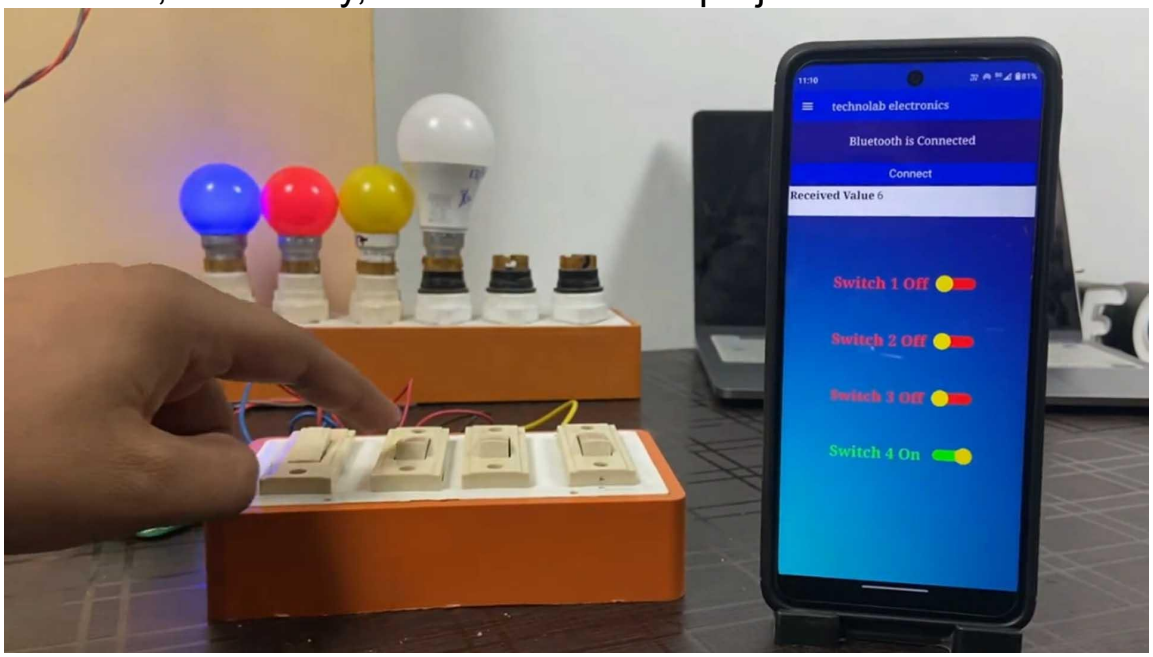
And green PCB masking looks very beautiful, and PCB looks pretty professional. The PCB assembly service of JLCPCB is fabulous. To upload the code into the ESP32 chip, I will use an ESP32

Development Board. Connect the PCB to the ESP32 board as per this circuit diagram. This is the code for our today's home automation project. And this code is very simple. Before you upload the code, in this section of the code, if you want, you can change the Bluetooth device name. This name will appear as the name of your Bluetooth device when we pair ESP32 with our smartphone. After this, no need to change anything. Now, straightforwardly upload the code to your ESP32 board after selecting the right board and the right communication port after hitting the upload button.



After clicking the upload button onto the PCB, I will press and hold the boot button and press the reset button once to make this module go into the boot mode. Here, as you can see, the code is successfully uploaded. Go to the Play Store and download this app. The download link of this app is available in the description. After installation, open the app and give the required permission. Now, open the Bluetooth setting of your phone, then click on "Pair New Device." As you can see, a Bluetooth device having the same name that we mentioned in the code is now showing. Now, tap on this device to pair with your phone. Once the ESP32 is paired with the phone, open the app again. Now, in the app, click on "Connect." Select your Bluetooth device. As you can see, Bluetooth is connected. A message is shown here. This means now our ESP32

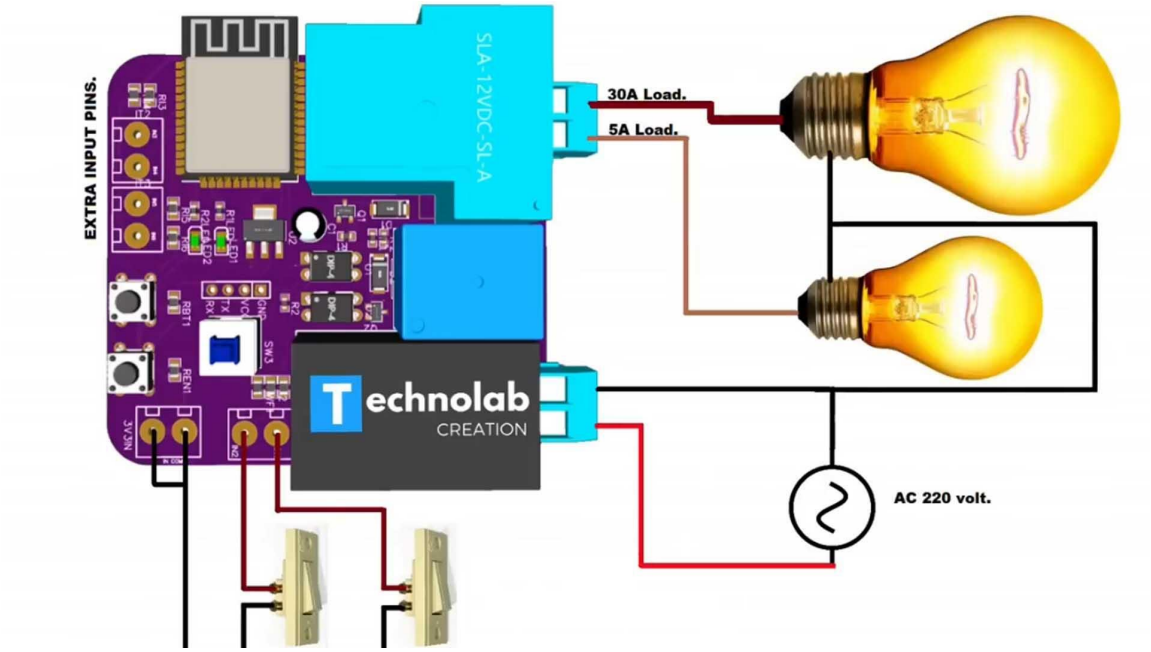
is connected with the app. Here, we did all the configuration, code uploading, and app pairing. Now, let's connect our devices and switches, and finally, we'll see how this project works.



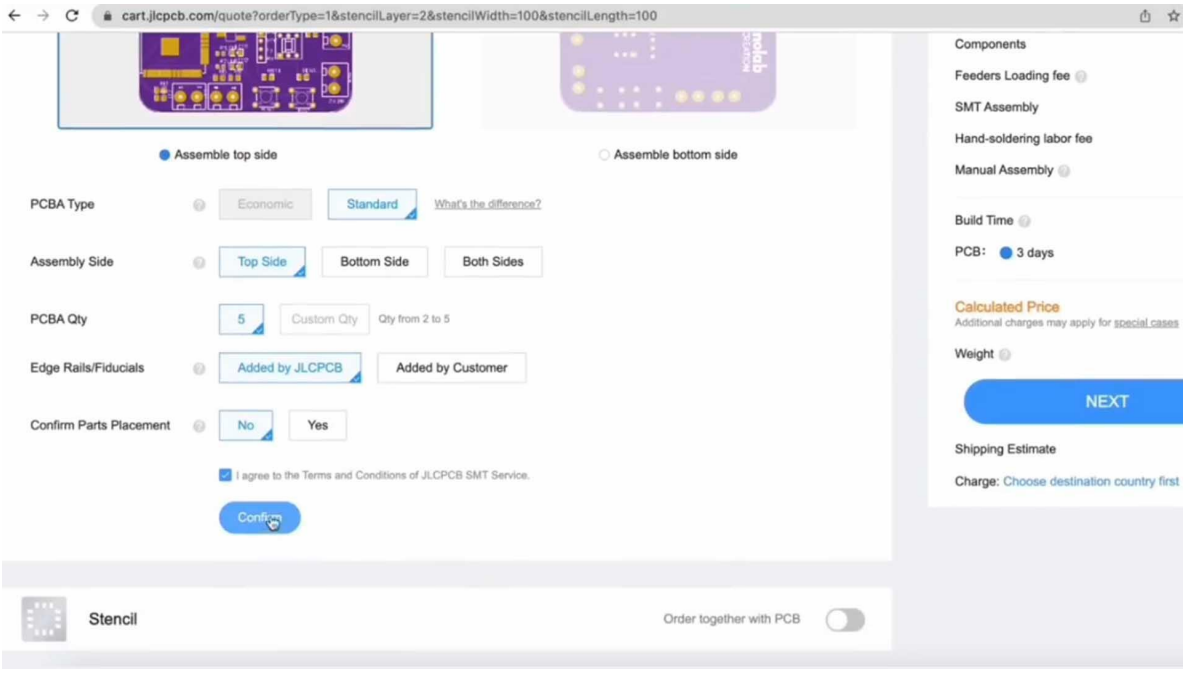
Now, connect all the bulbs or any of your home appliances and switches as per this circuit diagram. Here, everything is done. Let's see this project in action. As you can see, I am able to control the bulbs through the Android app and manual switches, and we are also getting real-time feedback in the app. This home automation project is very interesting and very useful. This home automation PCB is available for sale. You can easily purchase this board from our website. And apart from this module, we have many other interesting and useful modules in our store. Do check them out. Use coupon code "Welcome30" to get a 30% extra discount on all the products.

BLUETOOTH _ MANUAL CONTROLLED HEAVY LOAD DEVICES

In this project, I am going to make a home automation system in which we control heavy-load appliances like A/Cs and washing machines from an Android smartphone. And for the making of this home automation system, we don't require any IoT platform like Blink or ESP Rainmaker. Instead, we use the inbuilt Bluetooth feature of ESP32. So, in this way, we can easily connect the smartphone application to the ESP32 via Bluetooth and control the appliances from the app. This home automation system is very useful for those people who live in rural areas or don't have an Internet connection. We all know very well that automating devices using IoT platforms like Blink and ESP Rainmaker requires an Internet connection, so for people who live in areas where they don't have Internet access, this home automation system is very useful. They can automate their appliances from this PCB. The only drawback of this home automation system is that we can only control the appliances from the local area network. In this home automation system, we can easily control the appliances from the Android smartphone application, and apart from this app, we can also control it through the manual switch buttons, like we usually do. So here, we can easily control the devices from the smartphone application as well as from the manual switch buttons. Apart from this PCB, I have other home automation PCBs. These PCBs are fully tested and work very well. These PCBs are best for any kind of home automation. These PCBs are available for sale. Please check the description for more details, and during the project, I will let you know how to upload the code, circuitry, and connection of bulbs and switches.

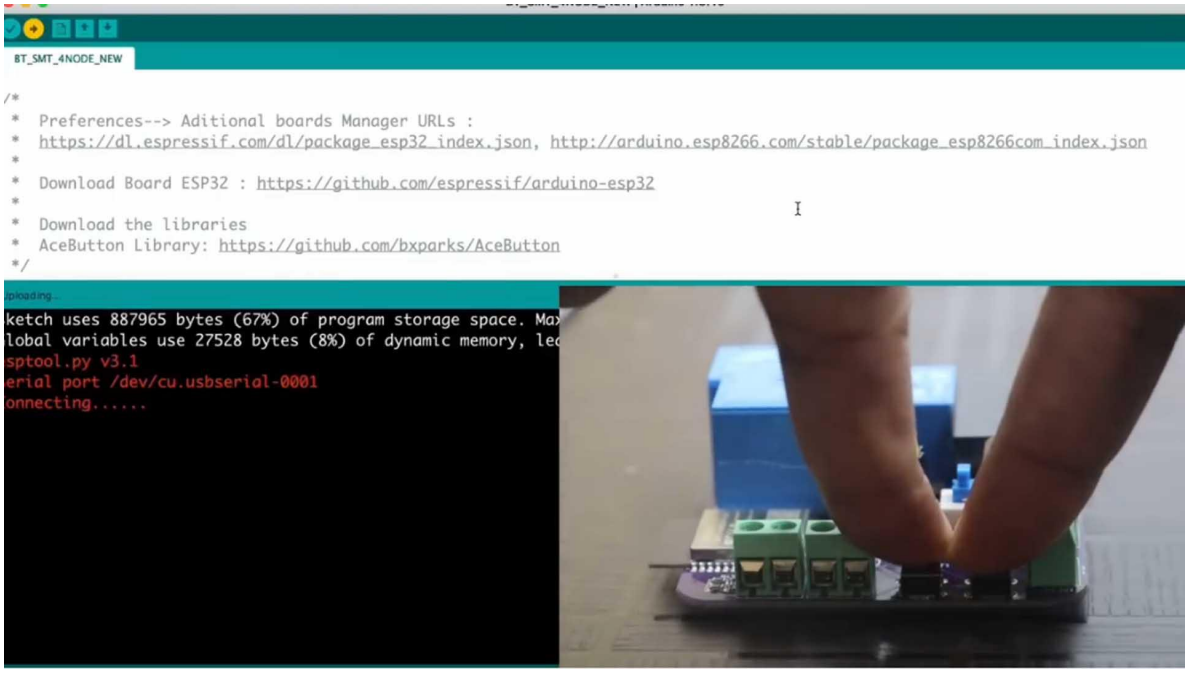


So, I recommend that you watch the complete project until the end. Now, let's get into this project. This is the schematic of the PCB. If you want your own custom-designed PCB, then you can download this schematic from the link given in the project description. After making the schematic, convert it into a PCB, arrange and place all the components in desirable places. Once the layout is ready, route the wiring and complete the design of the PCB. After the completion of PCB design, you need to download 3 files which will be required during the PCB order. These files are BOM, Gerber, and Cpl (pick and place file). Now, go to the JLCPCB website and click on the "Quote Now" button under PCB assembly. Click here to upload the Gerber file of your PCB. Here, JLC PCB will automatically set all the parameters of the PCB.

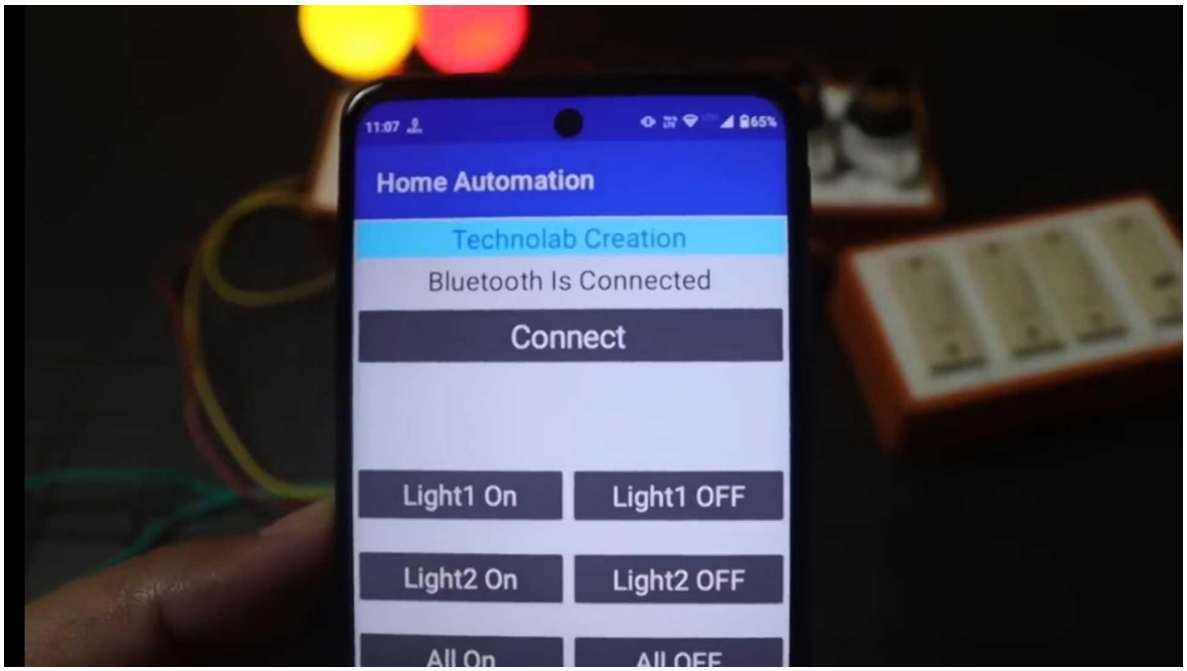


Select the PCB quantity and color masking of the PCB by yourself. I am selecting the purple color. Scroll down below and select PCB assembly options. Here, you have to select on which side you want to do PCB assembly—top side, bottom side, or on both sides. In my case, I want only the top side. After that, click on the "Confirm" button. For PCB assembly, we need two more files. One is BOM (Bill of Material), and the second one is Cpl (pick and place file). Upload these two files one by one. Here, you need to give the description about your PCB for which purpose you want to make this PCB. I am giving here "30 MPL Home automation PCB." After that, click on the "Next" button. All the components were shown here that are to be assembled. In case you want to not assemble any particular component, then you can deselect that component. After checking all the components, click on the "Next" button. Here, you will see a computer version of components placement which seems not accurate. This is only for reference purposes now. Click on "Save to Cart" to complete your order. After seven days, PCBs arrived at my place. As usual, the quality of the PCB is very premium and the components are soldered very well. Traces are perfect, Silk screen is fine, purple color PCB masking looks very beautiful, and PCBs look pretty professional. To upload the code into the ESP32 chip, I will use an ESP32 Development Board. Connect the PCB to the

ESP32 board as per the circuit diagram. This is the code for today's project. Download this code from the link given in the visual description. Before you upload the code, you need to make some changes in the code. First of all, you need to add the ESP32 boards in your Arduino IDE, and also, you need to add the "is button" library in your Arduino IDE to run this code. For this, go to tools, then manage libraries. A pop-up will come here, type "is button." Now, install this library by selecting the latest version of it. I have already installed this library, so I am skipping it. After installing the library, close this window. Rest of the code is OK, no need to change. One little change. If you want, you could make changes in the section of code that is the Bluetooth device name.



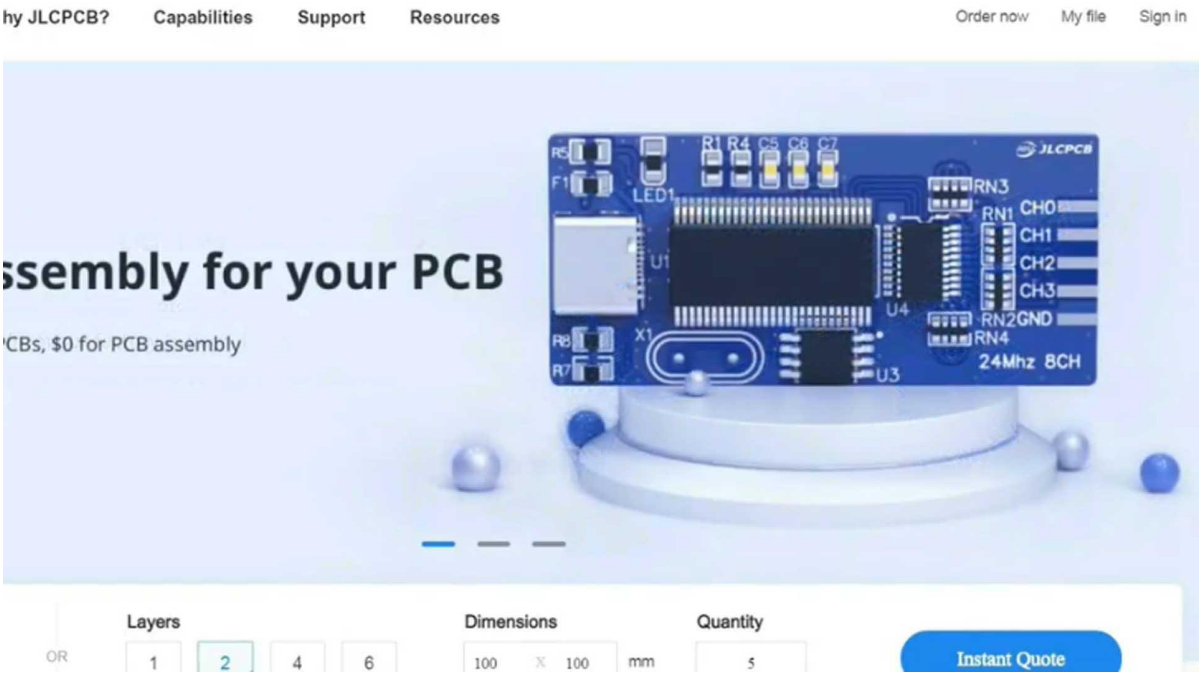
The name we give here will be the name of the Bluetooth device, and this will appear when we pair ESP32 with our smartphone. Now, upload this code after selecting the right board and COM port. After clicking the upload button onto the PCB, I will press and hold the boot button and press the reset button once to make this module go into the boot mode. As you can see, the code starts uploading. I have made this custom Android app from Code Ruler.



I will put the AIA and APK file of this app in the project description. If you want, you can download and customize it according to your needs. Download the APK file of this app and install this app on your Android smartphone. After installing the app, open the Bluetooth setting of your phone. Click on "Pair New Device." A Bluetooth device with the same name that we mentioned in the code will appear. Now, tap on this device, click on "Pair"

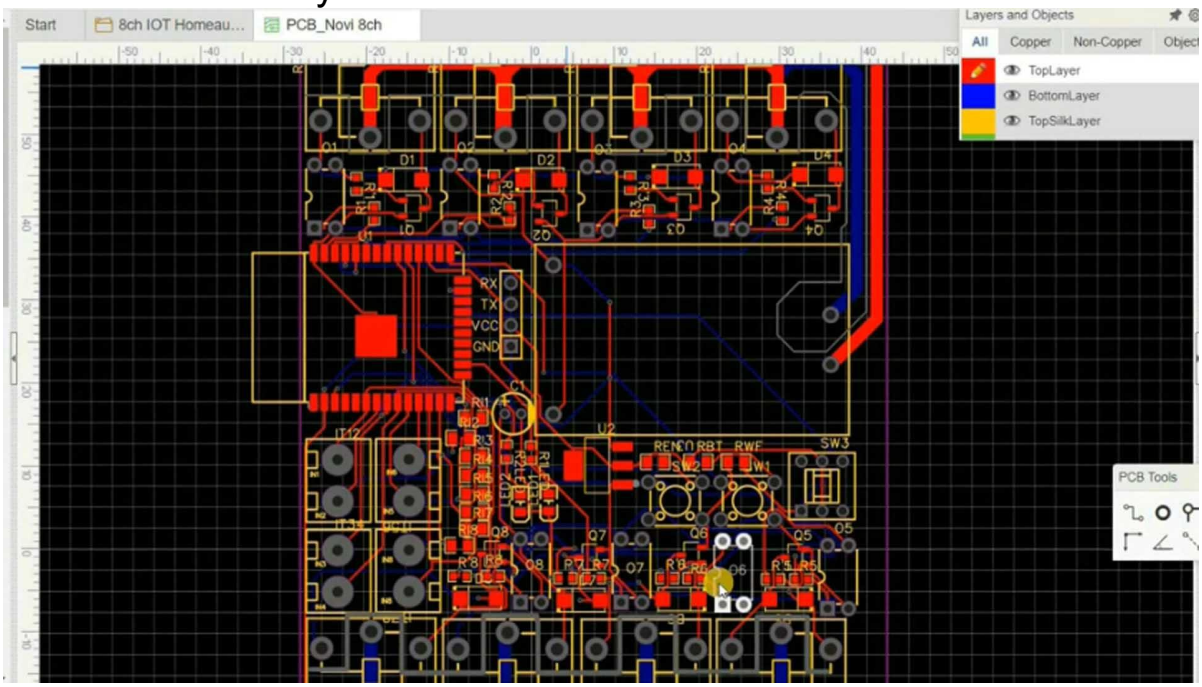
8 NODE SMT HOME AUTOMATION PCB

Hey, hello friends! Welcome to another home automation project project. In this project, we are going to make an Android app Bluetooth controlled home automation system. For making this home automation system, I will use my 8-node Summit home automation PCB and a custom-designed Android app. Everything is available in the description. You can download them from there. If you wish to make this home automation system, the best part of this project is that we don't need any Internet connection or any local server to connect the Android application and ESP32. The app will directly communicate with the ESP32 via Bluetooth. The only downside of this home automation system is that it will only work in a local range. If you want to purchase this 8-node Summit home automation PCB, then contact me. All the necessary details are available in the description. Now let's get into this project.



This project is sponsored by JLCPCB. JLCPCB is a well-known PCB

prototype company in China. It is specialized in quick PCB prototype and small batch production. You can now order a minimum of five PCBs for just $2. For more details, check the description. This is the schematic of today's home automation PCB.

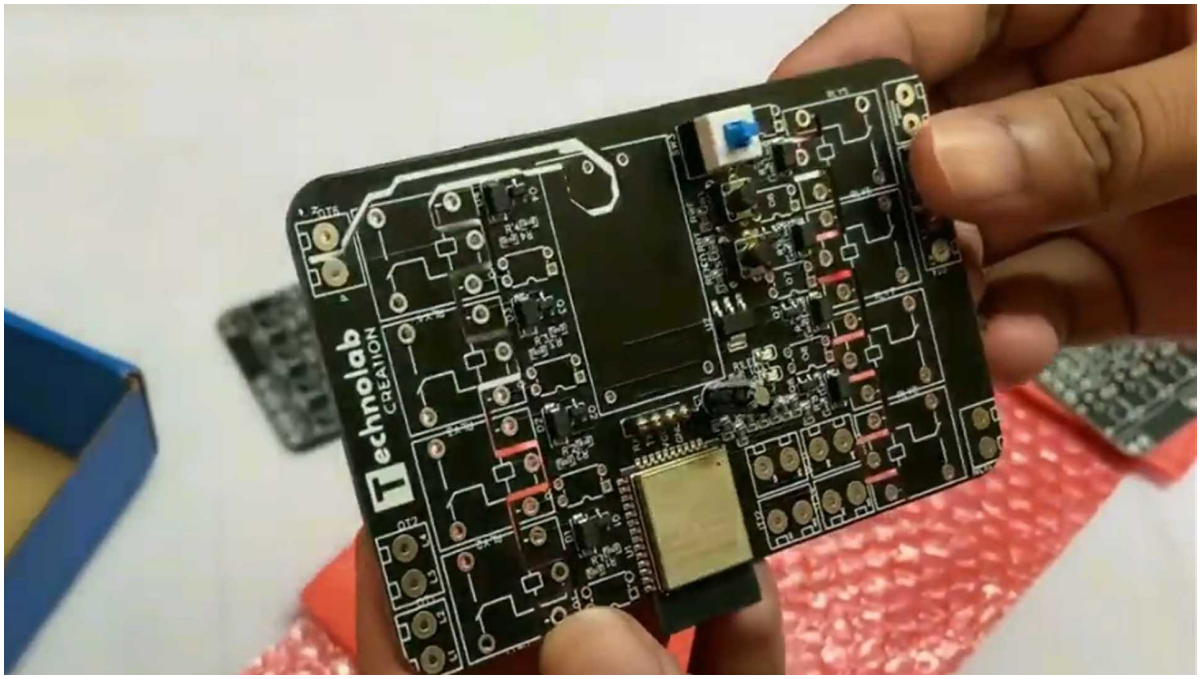


If you want, you can download this schematic from the description to design your own custom PCB. Now, convert this schematic into a PCB. After completing the design of your PCB, you can directly order the PCB from JLCPCB or just download the Gerber file from here. After that, go to the JLCPCB website, then click on the "Quote Now" button under the summit assembly. After that, upload the Gerber file of your PCB. Then, select the number of PCBs and color masking of the PCB if you want. After that, select the summit assembly service, and here, you have to select on which surface you want your components to be soldered—either top surface or bottom. After that, click on the confirm button. Now, here you have to upload two more files. One is the Cpl (pick and place file), and another one is BOM (Bill of Material). You can download these files from your easyEDA account. Just open that PCB project on your easyEDA account and then click on "Fabrication," then "BOM." Now, click on "Export," and then "Export BOM" to download the BOM file.

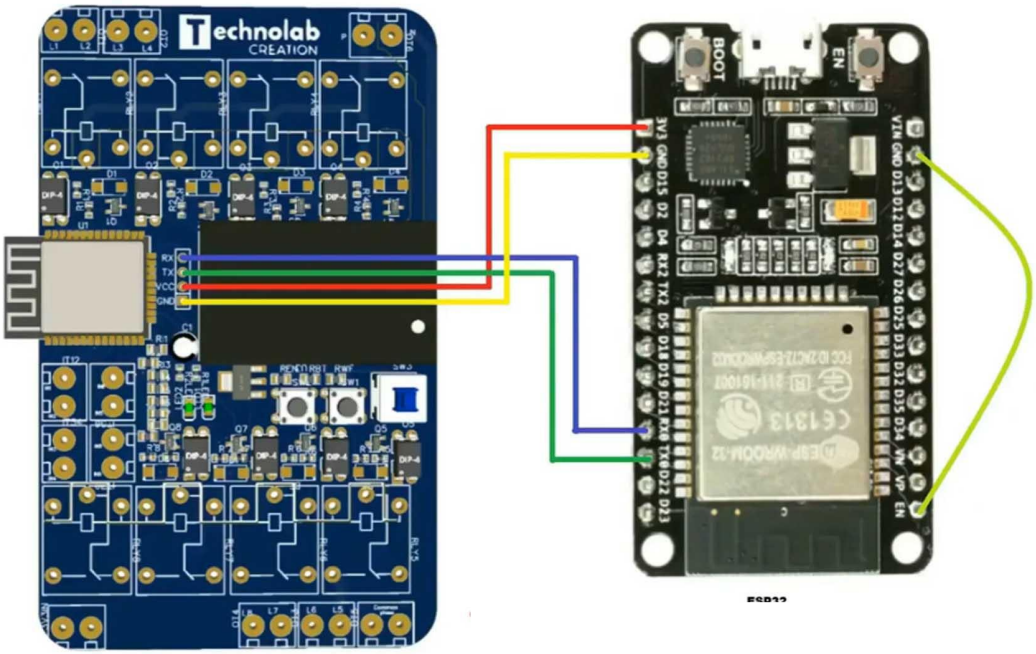
R1,R2,R3,R...	1K	R0603	RC0603JR-071KL 75V - ±5% - - - Thick Film Resistor...	100	LCSC	$0.1000
C1	220uF	CAP-TH_BD5.0-P2...	KM227M010D11RR0VH2FP0 220uF 10V 2mm ±20% 5mm 11mm 1000hrs...	50	LCSC	$0.5950
SW1,SW2	UK-B0202-G5.5-2...	SW-TH_4P-L6.0-W...	UK-B0202-G5.5-250 Top Actuated 50mA @ 12VDC Round But...	10	LCSC	$0.3940
3V3IN,IT12...	WJ126V-5.0-2P	PIT_WJ126V-5.0-...	WJ126V-5.0-2P P=5.0mm Screw terminal ROHS	55	LCSC	$2.5300
SW3	XKB8080-Z	SW-TH_6P-L8.0-W...	XKB8080-Z LATCHING 13.65mm 100mA 160±80gf Thr...	6	LCSC	$0.6534
U2	AMS1117-3.3	SOT-223-3_L6.5-...	AMS1117-3.3 15V 1.3V @ 800mA - - Fixed 3.3V SOT...	6	JLCPCB	$0.8178
RLY1,RLY2,...	SRD-05VDC-SL-C	RELAY-TH_SRD-XX...	SRD-05VDC-SL-C 5VDC SPDT - General Purpose Non Lat...	41	LCSC	$11.8818
U1	ESP32-WROOM-32D	WIFIM-SMD_39P-L...	ESP32-WROOM-32D SMD-38 WiFi Modules ROHS	5	LCSC	$16.0395
Q1,Q2,Q3,Q...	SS8050_C181160	SOT-23-3_L2.9-W...	SS8050 1.5A 0.3W NPN 25V SOT-23(SOT-23-3) ...	50	LCSC	$0.6250
O1,O2,O3,O...	817	DIP-4_L4.6-W6.5...	UPC817CG-D04-T - 1 5000Vrms Transistor Output Opto...	41	LCSC	$1.6400
RX	MH254V-11-04-10...	HDR-TH_4P-P2.54...	MH254V-11-04-1000 P=2.54mm Pin Header & Female Header...	6	LCSC	$0.5370
D1,D2,D3,D...	M7_C392013	SMA_L4.4-W2.6-L...	M7 1.1V 1A 1kV 5uA 1kV - 1A SMA(DO-214...	50	LCSC	$0.5250
RBT,REN,RI...	10K	R0805	0805W8F1002T5E ±1% ±100ppm/°C 10kΩ 0.125W 0805 Chip...	60	JLCPCB	$0.2580
U3	HLK-5M05_C91766...	PWRM-TH_HLK-5M1...	HLK-5M05 Plugin Power Modules ROHS	5	LCSC	$15.4650
LED1,LED2	17-21SURC/S530-...	LED0805-R-RD	17-21SURC/S530-A3/TR8 Red 624~632nm 0805 Light Emitting D...	20	JLCPCB	$0.4360

Please carefully check the packages of select

Similarly, download the Cpl file. After downloading both the files, just upload both the files onto this page. After that, select "Next." Now, here, it will show all the summit components that are to be soldered, and also, we can select which components will be soldered or not. Select the components according to your preference. After that, click on the "Next" button. Then, click "Save to Cart" to complete your order. After a week, my PCB arrived at my place in a new blue box from JLCPCB. Let me open the box. The packaging of the PCB in bubble wrap is very good.



Here it is—our home automation PCB. The quality of the PCB is good, and the summit components are soldered well. After soldering the rest of the components, the PCB will look like this—neat, clean, and well-arranged. To flash the code into the ESP32 chip, I will use an ESP32 Development Board. Now, make the connections according to this schematic. This is the code for our home automation project. This code is pretty simple, no need to explain.

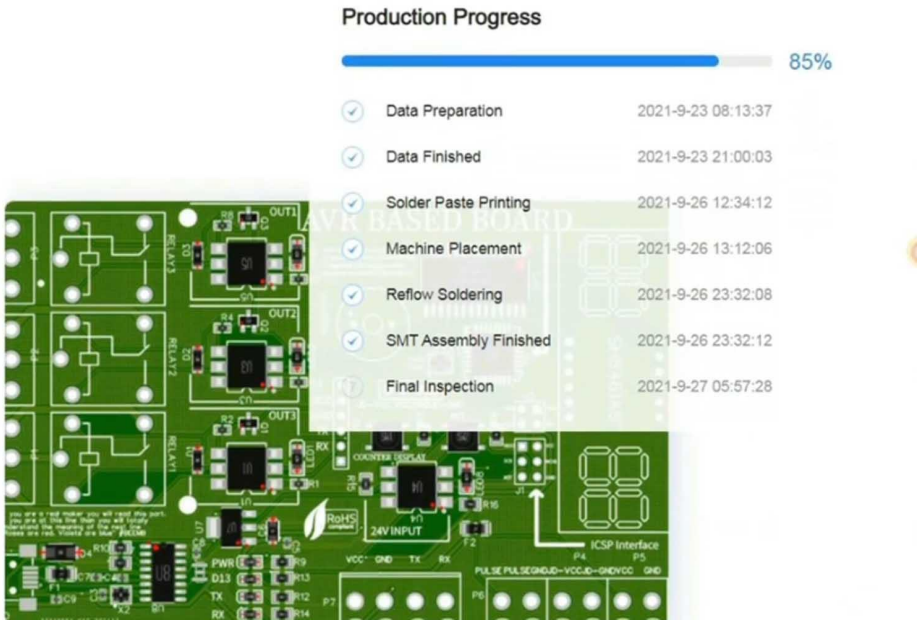


Just download this code and upload it after selecting the right board and COM port. After clicking the upload button onto the PCB, I will press and hold the boot button and press the reset button once to make this module go into the boot mode. As you can see, the code starts uploading. Now, connect all the bulbs and switches according to this circuit diagram. Now, everything is done. Let's see the project in action.

SMT SMART HOME-AUTOMATION PCB ESP32

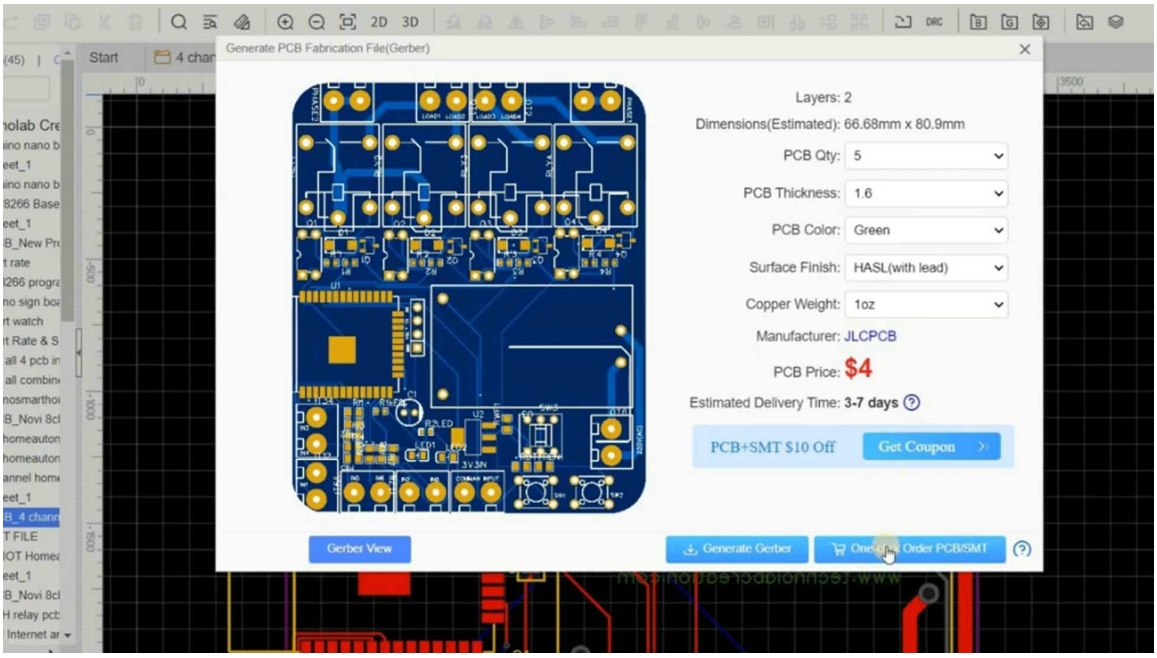
Hey, hello friends! Welcome to another project. In this project, we are going to make a very interesting home automation project. We are going to make lots of home automation projects with the Blink IoT cloud platform. Those projects were great. Now Blink comes with an update, Blink 2.0. Blink 2.0 has lots of amazing features that we missed in our previous projects. One of the features in Blink 2.0 is OTA (Over-The-Air). That means we can now update the code or Wi-Fi credentials over the air. Now, we don't need to hard code the credentials in the code every time we want to change. This feature is very useful. In this project, I will make a home automation project using Blink 2.0 and my 4-node ESP home automation PCB. If you want to purchase this PCB, check the description. All the necessary details are available in the description.

boards, allowing
duction. You can

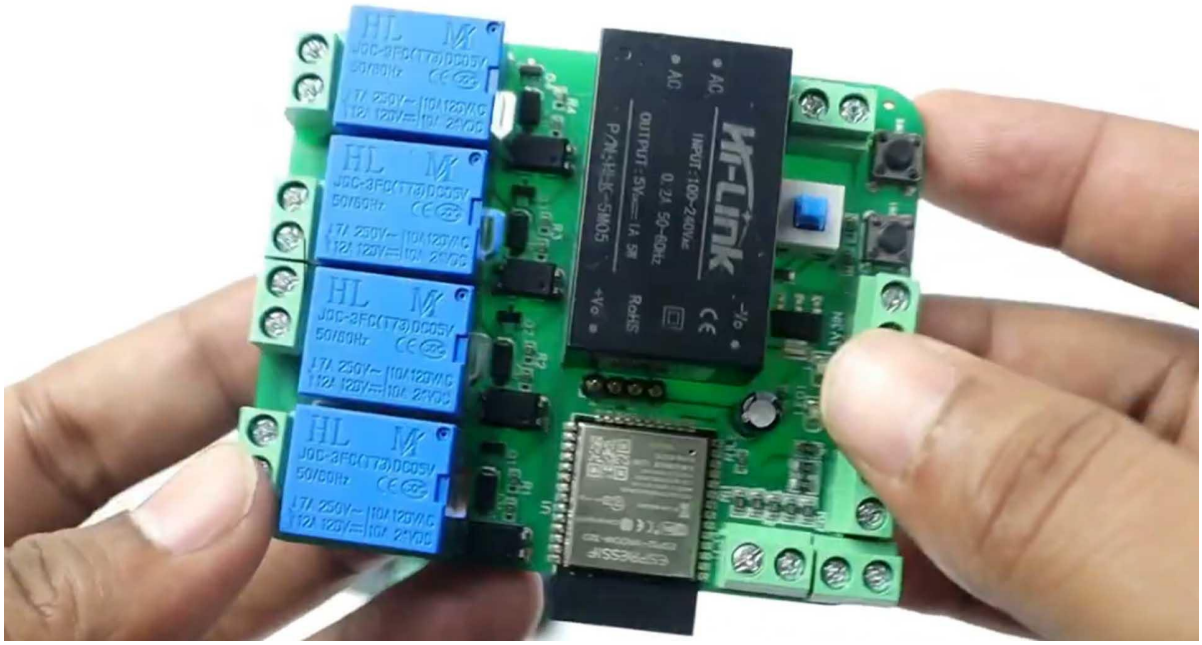


In this home automation project, we can control our home appliances via the Blink smartphone application from anywhere in the world.

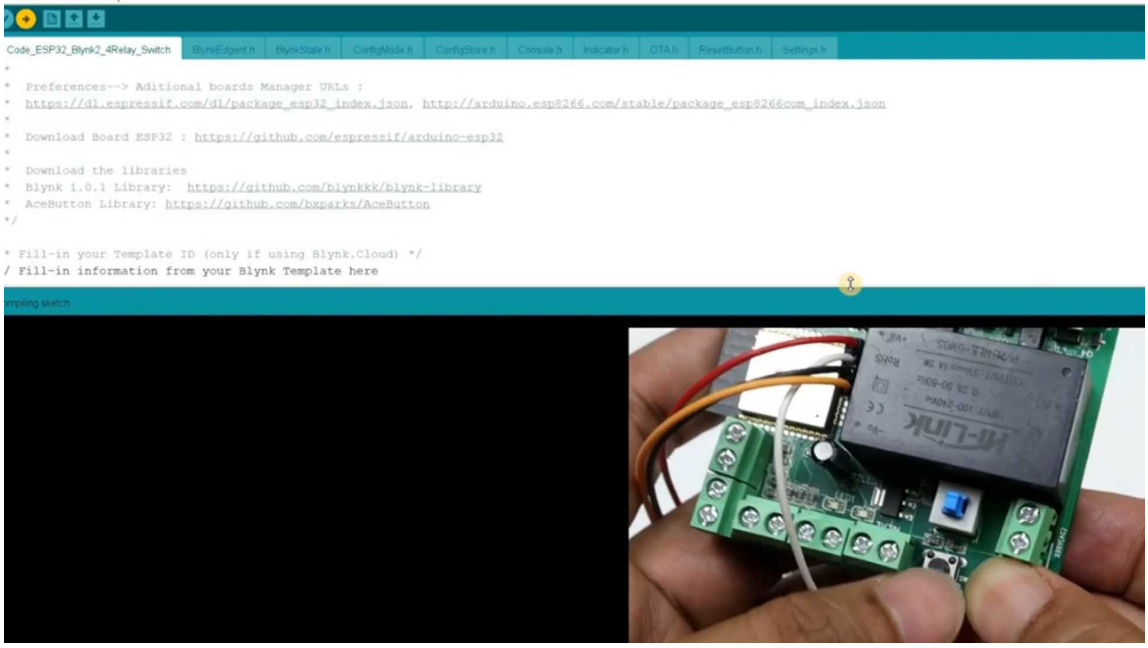
Additionally, we can control our home appliances via manual switch buttons, and we can also monitor the real-time status in the Blink app. During the project, I will explain the circuitry, code, and how to set up the Blink dashboard. So, I recommend you watch the project till the end. Now, let's get into this project. JLCPCB is a fast electronic manufacturing company. JLCPCB SMT services fulfill customers' money and time-saving needs. Customers enjoy low-cost, high-quality, and fast SMT services at a dollar-set-off fee. At the same time, they assemble electronic products from PCB design to PCB assembly products on the same online platform. JLCPCB provides a one-stop service from PCB design and PCB prototype to PCB assembly, and you can track their electronic manufacturing process in real-time. JLCPCB takes 24-hour summit build time and provides the fastest delivery. Get your PCB assembly product in one week from ordering, power sourcing, and assembly. And PCB assembly prototyping with thousands of components supported by JLCPCB and its reliable component partners like DigiKey and Mouser. Over 2000 in-stock components are available in the JLC PCB SMT parts library. This benefits customers to source components much faster and easier, bringing you a shorter PCB assembly production time. This is the schematic of today's home automation PCB. If you want, you can download this schematic from the description to design your own custom PCB. Now, convert this schematic into a PCB. After completing the design of your PCB, you can directly order the PCB from JLCPCB or just download the Gerber file from here. After that, go to the JLCPCB website. Then click on the "Quote Now" button under the summit assembly. After that, upload the Gerber file of your PCB. After that, select the number of PCBs and color masking of the PCB if you want. After that, select the summit assembly service, and here you have to select on which surface you want your components to be soldered—either top surface or bottom. After that, click on the confirm button.



Now, here you have to upload two more files. One is the Cpl (pick and place file), and another one is BOM (Bill of Material). You can download these files from your easyEDA account. Just open that PCB project on your easyEDA account and then click on "Fabrication," then "BOM." Now, click on "Export," and then "Export BOM" to download the BOM file. Similarly, download the Cpl file. After downloading both the files, just upload both the files onto this page. After that, select next. Now, here it will show all the summit components that are to be soldered, and also, we can select which components will be soldered or not. Select the components according to your preference. After that, click on the "Next" button. Then, click "Save to Cart" to complete your order. After a week, my PCB arrived at my place in a new blue box from JLCPCB. Let me open the box. The packaging of the PCB in bubble wrap is very good.



Here it is—our home automation PCB. The quality of the PCB is good, and the summit components are soldered well. After soldering the rest of the components, the PCB will look like this—neat, clean, and well-arranged. Open this page using the link given in the description. First of all, you have to sign up on this page using your email ID. Here, we have to verify the email ID. For this, go to Gmail and open the email sent by Blink. Then click on "Create Password."

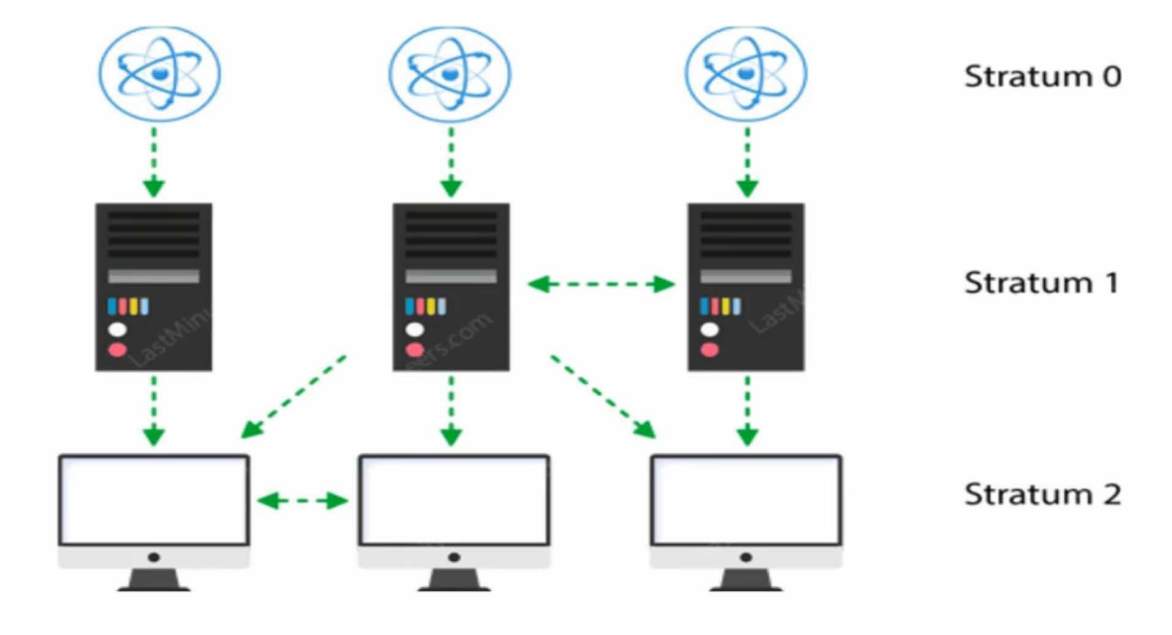


A new window will open, and here you have to create a password for

the Blink dashboard. Give any name for the dashboard admin, then click on "Done." Here, we have successfully created the account on Blink 2.0. These are some quick tutorials for setting up the Blink dashboard, but we don't need it, so close these windows. Now click here on the dotted icons, i.e., the template. Now click on "New Template." Give the name of your template on which your project is. I am giving it "4 Node Home Automation." Then select the hardware type. In my case, it is ESP32, and the connection type is Wi-Fi. After that, click on the "Done" button. Here, we have successfully created the template for our project. Now click on "Datastreams," then "New Data Stream," and select "Virtual Pin." Here, you have to give the name of the data stream. Give any conventional name you want. Now, select the pin on which you want to control your relay. I'm selecting virtual pin V1.

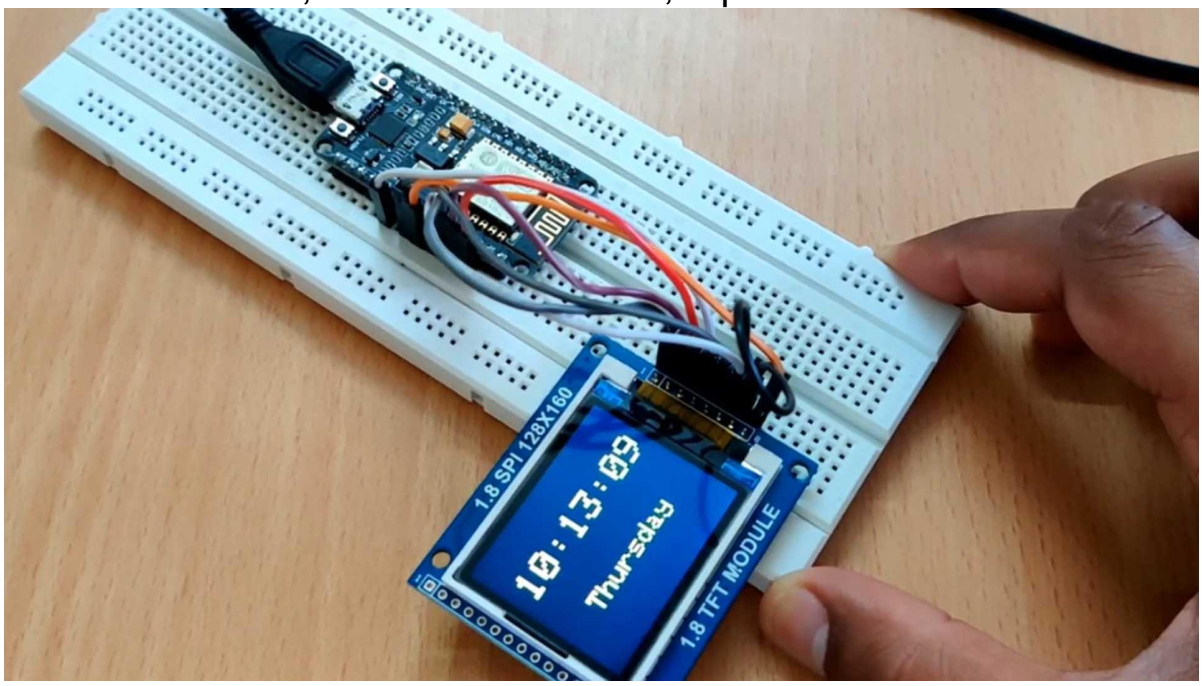
DIGITAL CLOCK USING NETWORK TIME PROTOCOL

This project is sponsored by JLCPCB. JLCPCB is a well-known PCB prototype company in China. It is specialized in quick PCB prototype and small batch production. You can now order a minimum of five PCBs for just $2. For more details, check the description. Hi, guys! Today, we will learn about the Network Time Protocol or NTP, which can provide time using the network, and we will make a clock using it. So, without wasting any more time, let's do this. So, before we go into all the stuff, let's understand about NTP.



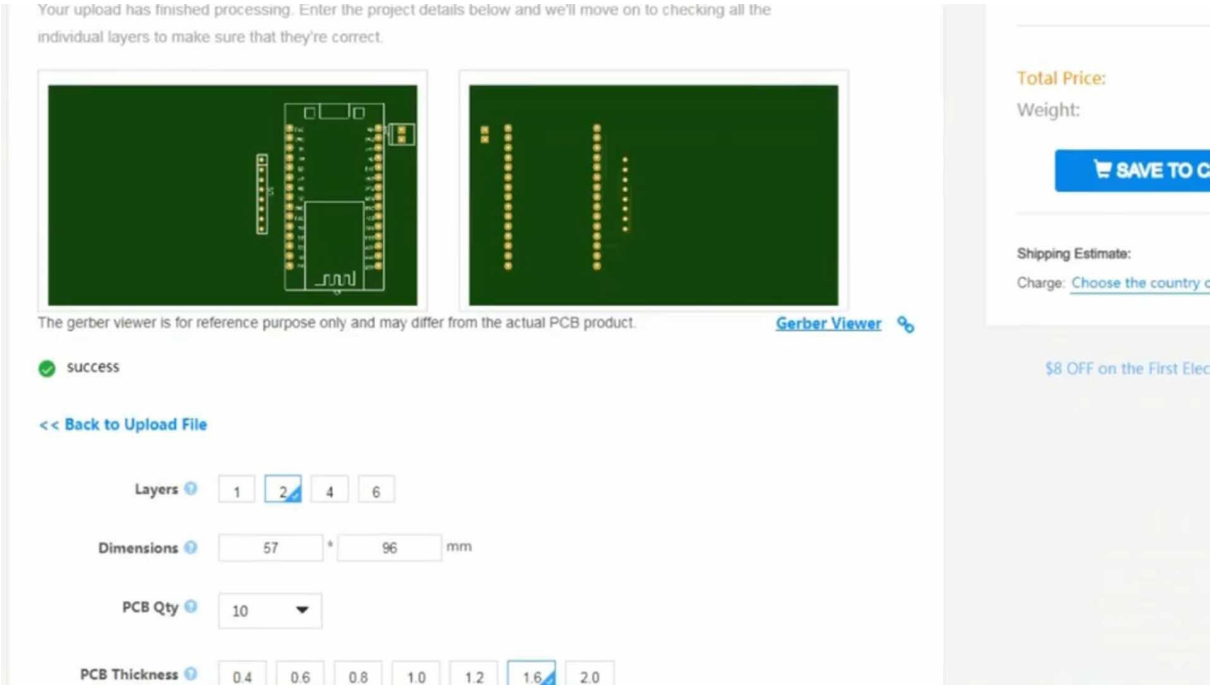
So basically, NTP is a standard Internet Protocol used to synchronize computer clocks, and using a cheap RTC module is OK, but they are not perfectly accurate. So, we will use a NodeMCU to get the time from the network, and an ESP8266 or NodeMCU just need a working Wi-Fi connection, that's it. Now open your Arduino IDE and make sure you have installed ESP8266 boards in your Arduino IDE. If it is not the case, then refer to the project given in the card section, and if you want to buy ESP8266, check out the

description given below. Then make sure you install the NTPclient library in your Arduino IDE, and if not, go ahead and install it. After this, you can try the examples provided within the library, or you can download my code from the description. I modified the example a little bit to get the time in proper format, and before uploading the code, make sure you put your Wi-Fi SSID and password in it. And one more thing you need to edit is the offset time. So, for my country, the offset is 19800 because my time zone is UTC+5:30. And 5:30 means 5 1/2, which is 5.55 * 60, and again multiplied by 60, which will give you 19800. So, you can calculate your offset time according to your time zone. Say your time zone is UTC+1, then your offset time will be 3600, which is 1 * 60 * 60, equals to 3600.

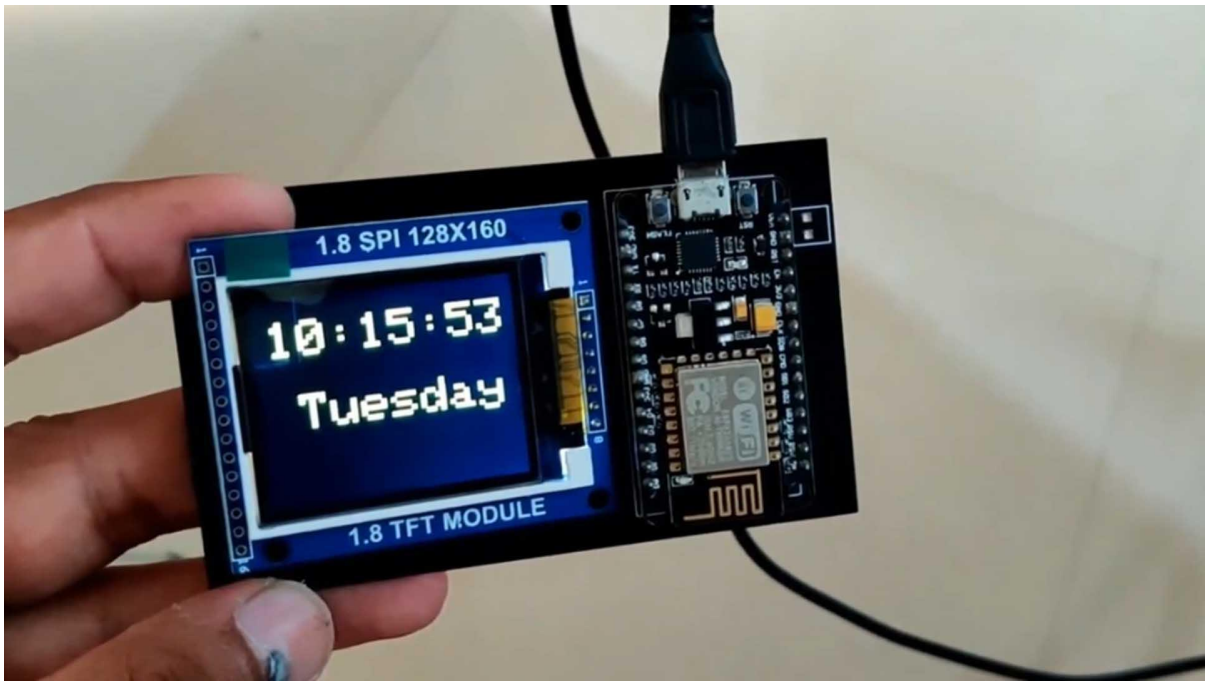


After editing the offset time, then hit upload to upload the code into the ESP8266. After uploading the code, open the serial monitor, and in the serial monitor, we can see we are getting our time on the serial monitor. To see the output properly on the display, I am going to use my 1.8-inch display. So, let's connect it all together. You can follow the schematic shown in the project. I've modified the previous code for this display. You can get both the code in the description as well as the display buying link. So, before we upload the code, make sure you have the ST7735 libraries in your Arduino IDE. If it is not the case, then install those libraries, then upload the code. Let's hit

the upload button and wait for a few seconds for the code to get uploaded. And after uploading the code, we can see our time and the day on our TFT display. To make it more professional, let's make a PCB of it.



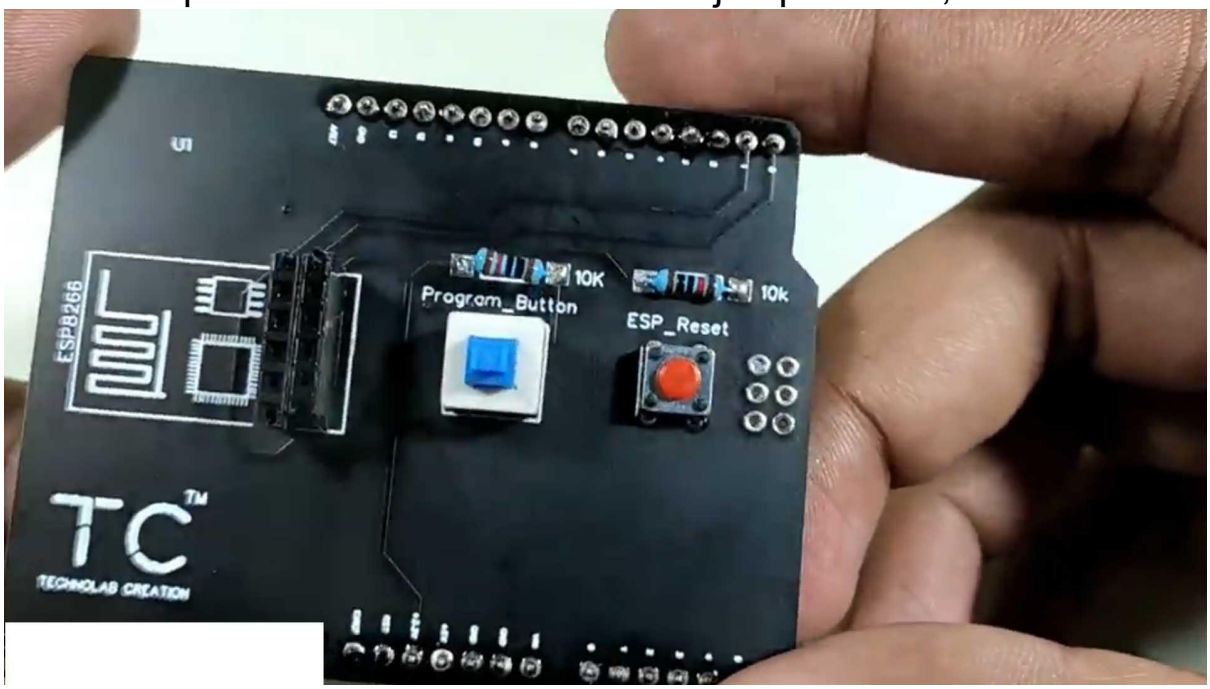
So, I went to the easyEDA website and selected the components and made a PCB of it, and then generated the Gerber files. And to save all this trouble for you guys, I pasted the link of Gerber files and schematics in the description. So, I uploaded these Gerber files to the JLCPCB website, and I got these PCBs in a week. So, let's open the box. And they sent this small scale with a magnifier on it as a gift. With it, let's take a look at our PCBs. So, let's cut it open and take it out. And here we can see our PCBs. Looking good, perfectly finished. You can also order your PCBs with the Gerber files provided in the description from JLCPCB website. Now, let's solder everything together and let's see how it works. So, I have soldered everything. It was really easy. Now, let's plug the cable and turn it on, and here we go. Our perfect desk clock is ready, and it is connected to my Wi-Fi, so we need not worry about RTC and its coin cell.



All we need is a proper case for this. Since I haven't made a case for this yet, I'm going to use it this way only for now, but later I'll make a case for that and use it like a proper desk clock. Well, you can make a case for yourself, but there's one more thing. To get this kind of font, you need to do a little more work. So, get the 7-segment font file provided in the description and paste that file in the "fonts" folder, which is inside the "Adafruit-GFX" folder, and then run the other code provided in the description for digital font, and you will get better fonts, as well as my fonts, showing on the screen. So, guys, that's all for today. I hope you like this project. If you do, then hit like, give me a share, and don't forget to subscribe and support me on Patreon to help me make more projects. And I'll see you guys next time. Till then, keep exploring.

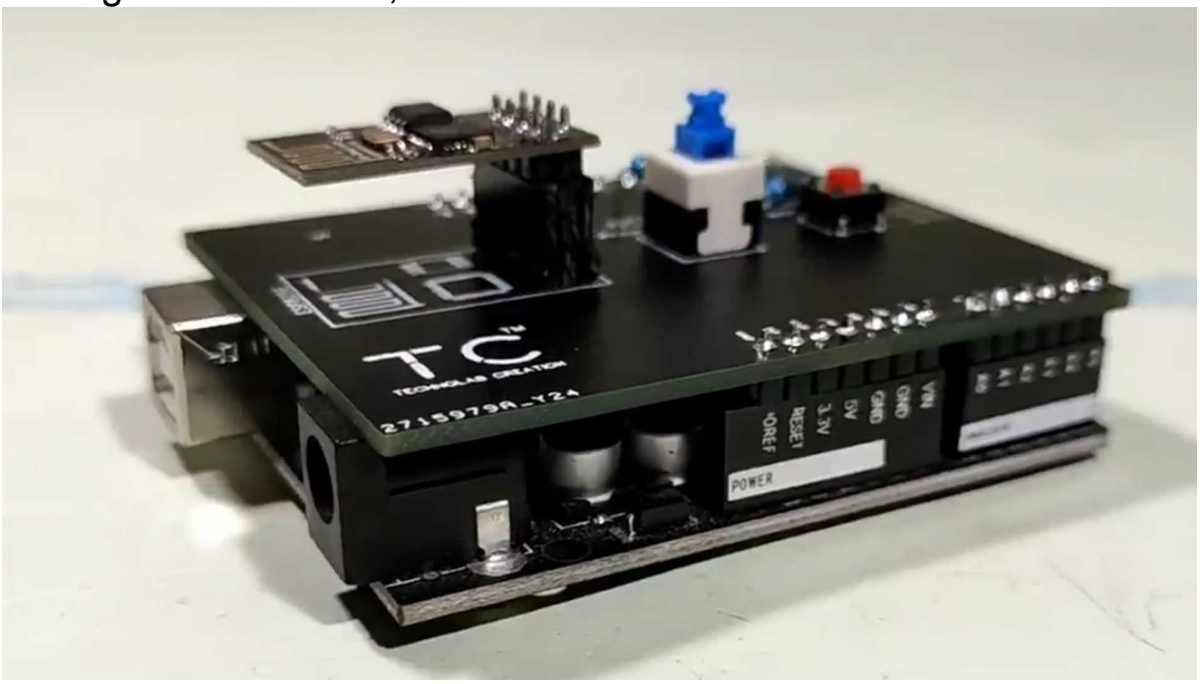
DIY PCB FOR ESP8266 WIFI MODULE

This project is sponsored by JLCPCB. JLCPCB is a well-known PCB prototype company in China. It specializes in quick PCB prototype and small batch production. You can order a minimum of 10 PCBs for just $2. Check the description for more information. Hey, hello friends! Welcome to another project. In this project, I will show you how to design and develop a programming PCB for the ESP8266 Wi-Fi module. If you remember my project on how to program ESP8266 using Arduino, I mentioned that the pins of the ESP8266 Wi-Fi module are not breadboard-friendly. There, I used jumper wires and a breadboard for programming the ESP8266 Wi-Fi module. Apart from the breadboard and jumper wires,



I still had to connect the level converter registers, post button, and reset the module separately. So, I have decided to make a PCB for the ESP8266 Wi-Fi module with all the required components, like a DPDT switch for selecting programming mode and normal mode, a

reset button, headers to insert the ESP8266 Wi-Fi module, and the level converter registers for the RX pin of the module. For designing my PCB, I am using EasyEDA. EasyEDA is very easy and simple for designing PCBs. After designing the PCB, I directly ordered from JLCPCB for manufacturing the PCB. After uploading the Gerber files, the software automatically detects the default settings. However, if you want, you can change these settings. Now save to cart to complete your order. After seven days, my PCB arrived at my place. "The world ain't all sunshine and rainbows. It will beat you to your knees and keep you there permanently if you let it. It's about how hard you can get hit and keep moving forward. That's how we did this. Done. You're better than that." After assembling all the components on the PCB, press the switch button to programming mode position and press the reset button once. Now the programming mode is activated. The ESP8266 Wi-Fi module can be programmed using the Arduino IDE, and in order to do that, we need to make a few changes to the Arduino IDE. First, go to File, then Preferences in the Arduino IDE, and in the Additional Board Manager URL section,

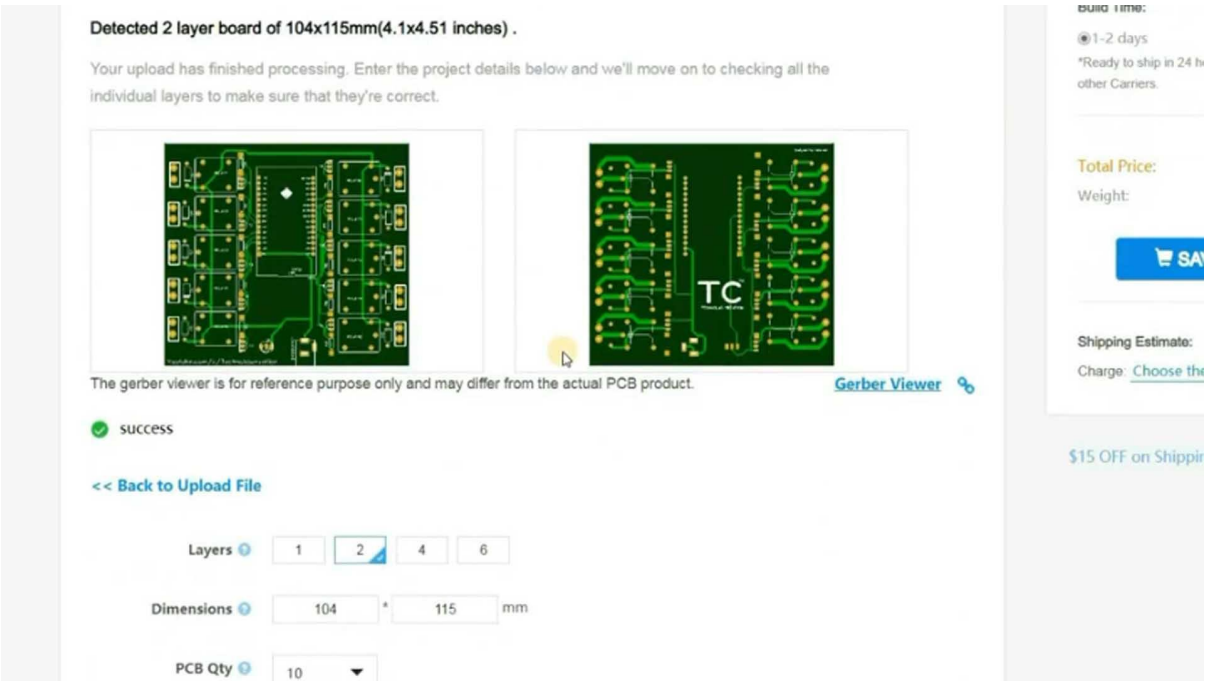


paste the URL copied from the description, and then click OK. Now go to Tools, Board, then Board Manager, and search for ESP8266. In the search field, select the ESP8266 ESP8266 Community, and then

click on install. I already installed it. Now our field is ready to flash the program. Let me show you an example by uploading a Blink sketch. Open the Arduino IDE. In the Board option, go to Tools, then Board. Look for Generic ESP8266. Select the Generic ESP8266 board. Select the appropriate port number in the IDE. Now open the Blink sketch and change the LED pin to two. Here, 2 means GPIO2 pin of the ESP8266 module. Hit the upload button, and the code will take a while to compile and upload. We can see the progress at the bottom of the IDE. This source program is successfully uploaded.

ESP32 BLUETOOTH CONTROLLED AUTOMATION SYSTEM USING ANDROID APP

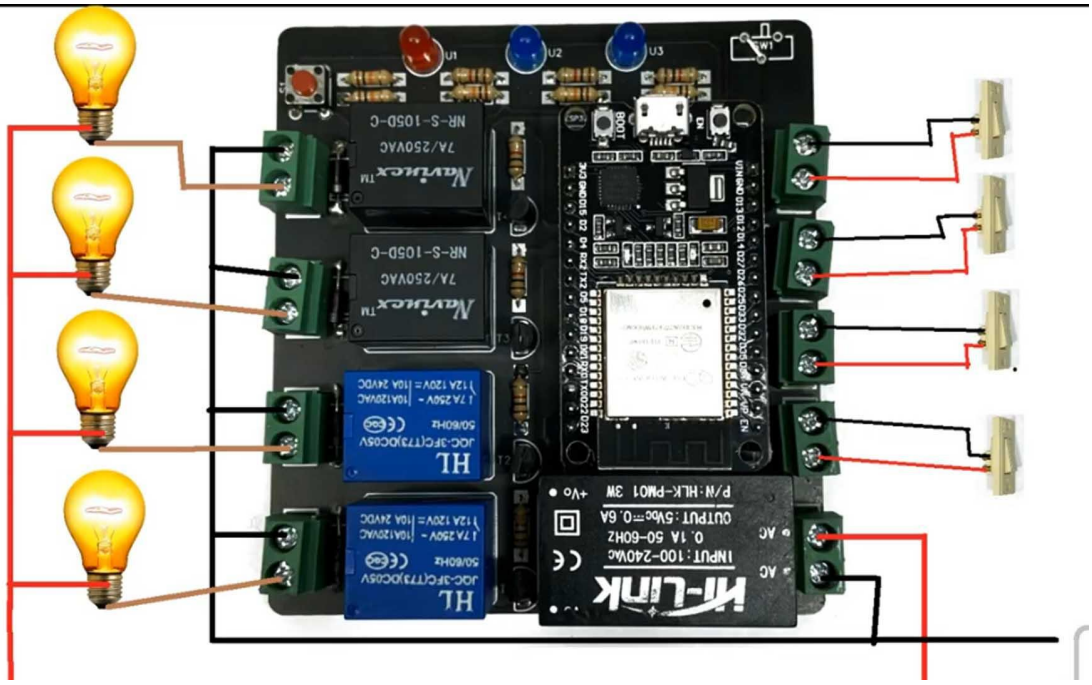
Hello everyone, welcome to another project. In this project, we are going to make an Android app-controlled home automation system. We all know ESP32 comes with Wi-Fi, Bluetooth Low Energy, and Bluetooth Classic. In this home automation project, we are going to use the ESP32's Bluetooth Classic feature to control our home appliances using this PCB and an Android app. With this PCB, we are able to control a total of 10 devices. So let's get started. This project is sponsored by JLCPCB. JLCPCB is a well-known PCB prototype company in China. It is specialized in quick PCB prototype and small batch production. You can now order a minimum of five PCBs for just $2. For more details, check the description. For this project, I am going to use this old PCB of mine. You can download the Gerber file of this PCB from the description. For designing my PCB, I am using EasyEDA. EasyEDA is very easy and simple for designing PCBs. After designing the PCB, I directly ordered from JLCPCB for the manufacturing of the PCB. After uploading the Gerber file, the software automatically detects the default settings. However, if you want, you can change these settings.



Now save to cart to complete your order. After seven days, my PCB arrived at my place. Connect all the bulbs in this manner. The code is very simple and doesn't need any modification. Just download it from the description and upload to your ESP32 board. One thing you could change if you want is the name of the Bluetooth device, and the rest of the things are OK. Hit the upload button after selecting the dashboard and COM port. Download this app from the given link in the description and install it on your Android phone. Now open the Bluetooth setting of your phone and pair your phone with ESP32. Now open the app again and tap on connect. Tap on "habt ESP32". If the app is connected to the ESP32 via Bluetooth, then the "Bluetooth is connected" message will be shown here.

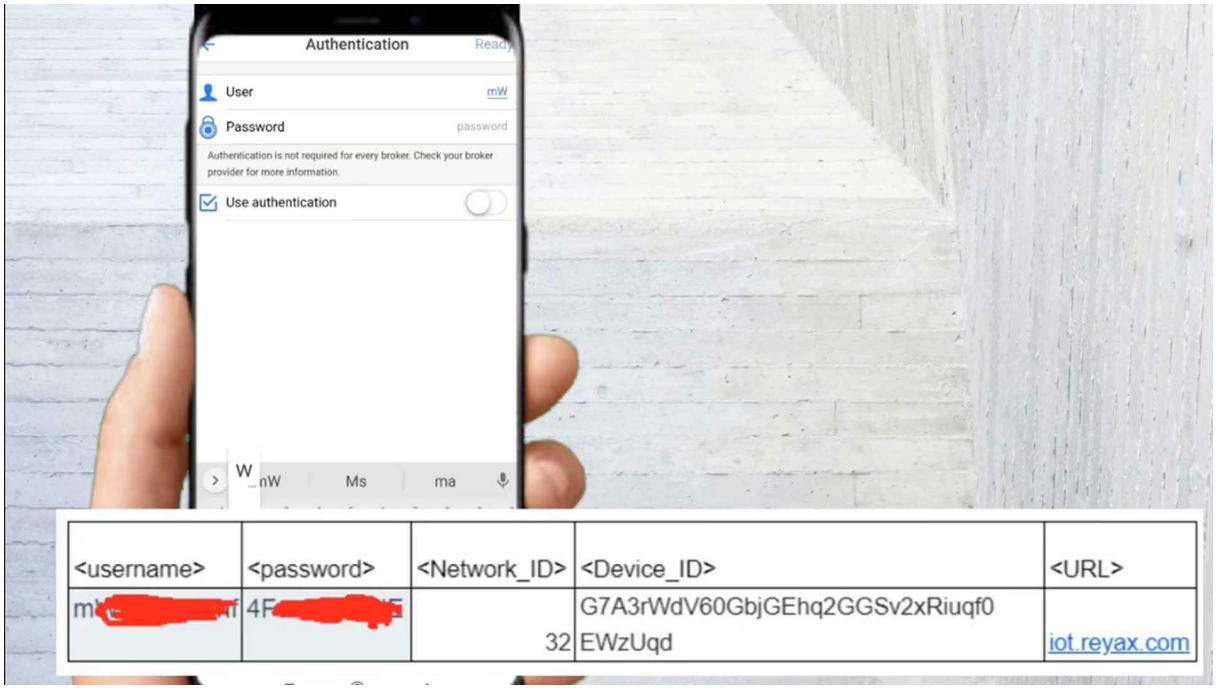
ESP32 INTERNET REAL TIME FEEDBACK USING REYAX MQTT CLOUD

Hello everyone, welcome to another project. In this project, we will learn how we can make our own home automation system using our own MQTT cloud broker. In this home automation project, we are able to control our home appliances by any smartphone and also control them with manual switches. We can monitor the real-time status on the smartphone as well. This project works from anywhere in the world because it's a cloud-based MQTT broker. So let's get started



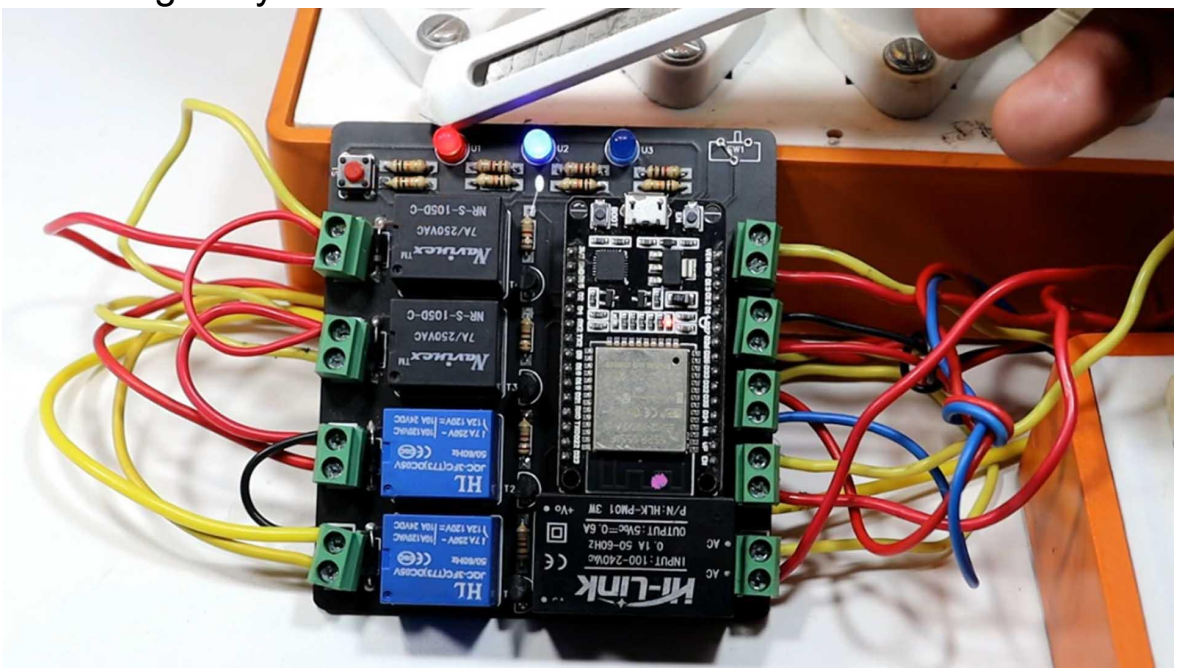
For making this home automation project, I am going to use the RexRYC 1001 MQTT IoT Cloud Platform, which you can easily purchase from Amazon.com. The link is available in the description. If, in any case, it's not available in your country or you're facing any problems while purchasing this IoT cloud platform, then just contact

the sales team of Rex. They will guide you. So for $15.00, we are getting the RexRYC1001 cloud server for five years, and the monthly message limit is 100K, which is really affordable in this price range. As soon as we make the purchase, we get our own credentials on our registered email ID. We will receive the username and password for our MQTT cloud platform, which you should keep confidential. This is the code for our today's home automation project. Download this code from the description and open it in Arduino IDE. Now before uploading the code, you need to make a few modifications in it. First of all, you have to add the PubSubClient library in your Arduino IDE. To add this library, go to Sketch, then Include Library, Manage Libraries, and search for PubSubClient. Scroll down, find the PubSubClient library, select the latest version, and click on Install.



Now close this window. In the code, enter your SSID and password of your router or hotspot. The MQTT server will be the same for all, so you don't have to change it. Then, enter the username and password you received in your registered email ID from Rex. That's it. After selecting the right board and COM port, hit the upload button to control the home appliances through the smartphone. We also need an MQTT client in our smartphone. For that, I am going to use the IoT On/Off app. This app is available for both Android and

iOS. You can easily download it from the App Store or Play Store. The links for both are available in the description. Now open this app. Tap on the menu button, then click on Settings, Configuration, and tap on MQTT Broker. Enter the MQTT server details like host, which is iot.rex.com. The port will be 1883, and turn off the websocket. Click save to add the broker to the list. Click on Authentication, and enter the username and password you received on your email address. Turn on authentication. If you've entered the correct details, the status should be connected. Now tap on the menu button, go to Dashboards, and tap on Garden. First, delete all the gadgets. Tap on Edit and delete all the gadgets one by one. Now tap on the plus button and select the switch. Give any display name you want, such as "Switch 1." Tap on Publish, turn on "Allow Publish," and tap on Topic String. Enter the subscriber name, which is "switch-one." After that, tap on Ready. Tap on Ready again. For true, enter 0, and for false, enter 1, as I'm using reverse logic. Tap on the Subscribe option, click on Topic Filter, and enter the publisher name, which is "switch-one-state." Provide the subscribe values, 0 for true and 1 for false. Click Ready. Repeat this process for the remaining relays.



Tap on Done. Now we are ready to control the home appliances. Let's see it in action. This home automation project has lots of cool features. Let me explain: As you can see, there are three onboard

LEDs. The first one is the network LED, which glows continuously when ESP32 is connected to the Internet; otherwise, it starts blinking, indicating that ESP32 is not connected to the Internet. The second one is the server LED. This LED glows when ESP32 is connected to the MQTT server; otherwise, it remains turned off. The third LED is the message LED, which blinks once whenever a new topic or message comes from the MQTT server, indicating that a new topic or message has arrived. Let me show you in action. I'm going to turn off the hotspot.