IoT-Based Industrial Debug Message Display Using AWS, ESP8266 And C#

Sudip Chakraborty [1] & P. S. Aithal [2]
[1] D.Sc. Researcher, Institute of Computer Science and Information sciences, Srinivas University, Mangalore-575 001, India,
OrcidID: 0000-0002-1088-663X; E-mail: sudip.pdf@srinivasuniversity.edu.in
[2] ViceChancellor, Srinivas University, Mangalore, India,
OrcidID: 0000-0002-4691-8736; E-Mail: psaithal@gmail.com

Area/Section: Computer Science.
Type of the Paper: Experimental Research.
Type of Review: Peer Reviewed as per |C|O|P|E| guidance.
Indexed in: OpenAIRE.
DOI: https://doi.org/10.5281/zenodo.8250418
Google Scholar Citation: IJMTS

How to Cite this Paper:
Chakraborty, S., & Aithal, P. S. (2023). IoT-Based Industrial Debug Message Display Using AWS, ESP8266 And C#. *International Journal of Management, Technology, and Social Sciences (IJMTS), 8*(3), 249-255. DOI: https://doi.org/10.5281/zenodo.8250418

International Journal of Management, Technology, and Social Sciences (IJMTS)
A Refereed International Journal of Srinivas University, India.

CrossRef DOI: https://doi.org/10.47992/IJMTS.2581.6012.0300

Received on: 16/07/2023
Published on: 16/08/2023

IoT-Based Industrial Debug Message Display Using AWS, ESP8266 And C#

Sudip Chakraborty [1] & P. S. Aithal [2]
[1] D.Sc. Researcher, Institute of Computer Science and Information sciences, Srinivas University, Mangalore-575 001, India,
Orcid ID: 0000-0002-1088-663X; E-mail: sudip.pdf@srinivasuniversity.edu.in
[2] Vice Chancellor, Srinivas University, Mangalore, India,
Orcid ID: 0000-0002-4691-8736; E-Mail: psaithal@gmail.com

ABSTRACT

Purpose: *In the industrial automation field, debugging is an essential part. Generally, most of the debugging we do in the product development phase and a little bit at the service time. The typical procedure to debug any electronic device is to display a "debug message" inside the terminal window. For this purpose, we commonly use various converters which convert from USB to other ports like RS232, RS485, etc. However, sometimes, we cannot connect our debug cable directly to the working devices. That would be better if the debug message is displayed on the working system terminal without any wire. Here we demonstrate a procedure to display the debug message without any wire. Using the IoT, we do that efficiently. The Code is available on GitHub. Interested researchers can download and continue further research on it.*

Design/Methodology/Approach: *First, We need to create the IoT profile inside the AWS IoT cloud. Using the AWS IoT credential, We update the firmware of the Wifi module. Then it is installed inside the system, which we are debugging. The wifi module's Transmit and receive pin will be connected to Device's serial PIN (TX-RX, RX-TX). When the Device sends any debug message through the serial port, it is received by the Wifi module and will update the AWS cloud shadow register. We build an AWS IoT MQTT client in the C# (visual studio. NET). The IoT client fetches the updated data and notifies the UI main thread. One listbox is connected to it. Finally, the UI thread pushes the updated data into the Listbox's first row.*

Findings/Result: *Sometimes, our Device needs to debug without any wire because there is no scope to connect it with the cables. So this procedure can help to debug wirelessly. This procedure has several advantages. Using this debug display eliminates to gets hazards from debugging systems. The debug message can be visible from a remote place. It is also possible to observe the message from multiple remote locations Opening multiple IoT Clients, and subscribing to the same IoT Topic.*

Originality/Value: *The described procedure is a different way to display the debug message. Some advantages are available over traditional procedures. Using this procedure, the researcher can transmit real-time data. It can provide value to the researcher's work. Using this procedure, the researcher can transmit their sensor data to a remote place. Furthermore, it can save to the remote server for future use.*

Paper Type: *Experimental-based Research.*

Keywords: wireless display, wireless debug display, IoT debugger

1. INTRODUCTION :

Debug is an essential journey. More or less, we do it for every product developer. When we develop the product in the lab environment, all well-equipped measurement tools, like a programmer, debugger, oscilloscope, function generator, etc., are ready to use. Once the product is developed, it is installed in the actual environment, mainly in the industry. Most of the time, the embedded system is a kind of encapsulated box, and it is fitted inside the panel. Everything is fine till the Device is working properly. Let us say some problem occurred and needs to debug. Then we need to go to the factory, open the panel, connect the debugger, and find out the issue connecting with the working laptop or system. Here

are some issues incurred. First, we must stay there for the remote site until the system reaches full-fledged working condition. Second, the working system must connect with devices inside the panel. The risk incurred if high voltage injection. These issues we can solve using IoT.

IoT is now an emerging technology. Most IT giant company has their IoT domain. Among them, AWS is primarily popular, has robust security, and is cheap. Using this technology, we can make our embedded debug more friendly. From our office or product development laboratory, we can debug the Device where the Device is installed. If the system does not suffer hardware damage, firmware-related bug fixing is possible remotely. We can upgrade firmware and monitor the Device's debug message through this procedure.

We need to plan according to that if we use remote debugging. So we need to install the ESP8266 Wifi module inside the Device. For that, one serial port of the microcontroller is needed. The Wifi module is connected to a nearby router. When the Device was powering up, it communicated with the AWS server and established the connection. When the controller sends some data, the ESP receives it and sends it to the AWS cloud. Conversely, the C# IoT client receives the data and updates the display.

2. RELATED WORKS :

Froiz-Míguez et al. (2018) focused on designing and implementing an IoT home automation system for fog computing applications using MQTT and ZigBee-WiFi sensor nodes [1]. Burd et al. (2018) presented a study on courses, content, and tools for IoT in computer science education [2]. Al Mubarak (2022) conducted a master's thesis on designing and implementing an industrial IoT-based SCADA system for petroleum pipeline systems [3]. Uddin et al. (2021) conducted an empirical study on IoT topics discussed by developers on Stack Overflow [4]. ŞİMŞEK and ATILGAN (2023) investigated attacks on the availability of IoT middleware protocols, focusing on a case study of the MQTT protocol [5]. Baronia introduced a mobile app, "Rescuer, " incorporating voice recognition, location SMS reciprocation, and push-aid systems [6]. Widerberg Palmfeldt and Mattsson (2023) compared existing penetration testing frameworks for IoT security and proposed a generic framework [7]. Balau (2020) explored using Azure IoT and Edge technologies [8]. Στούλος (2019) proposed an architectural proposal for an automated watering system in agriculture using wireless sensor networks [9]. Borba (2018) proposed a discovery service for the MQTT protocol in an IoT-based motor-generator monitoring system [10]. In summary, the reviewed literature covers various topics related to IoT applications, protocols, security, education, and implementation in multiple domains. The studies provide valuable insights into IoT systems' practical aspects, challenges, and potential solutions.

3. OBJECTIVES :

The research aims to provide information for the researcher to display wirelessly using the latest emerging IoT technology for embedded system development. Sometimes our researcher needs to see the debug message from the remote instrument because the Device is already installed in that factory or industry and is not easily reachable. To debug any system, observing the debug message is very important. Without observing debug messages, debugging any system takes a long time. It too critical to assume exactly where the programming pointer is and what is the register content. So these procedures can provide the solution.

4. APPROACH AND METHODOLOGY :

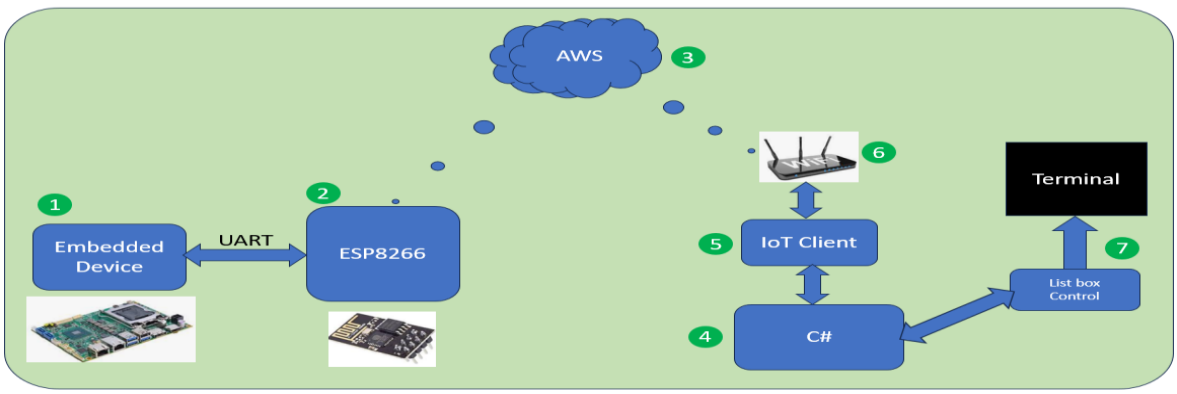

Fig. 1: Block Diagram of the project [Source: Author's]

In Figure 1, we describe the block diagram of the project. We have every block marked so that the description can be easily understandable.

1) **Embedded Device**: It is a debuggable embedded Device. Most of the microcontroller has several UART. For our wifi module, we need one serial port. The Microcontroller RX pin will be connected to the wifi modules' TX pin, and the microcontroller's TX pin will be connected to the RX pin of the modules. We must select different ESP modules if a serial port is unavailable inside the microcontroller. There is a couple of ESP module versions available. We can select according to our available hardware.
2) **Wifi module**: the ESP8266 module is cheaper than another available module we can easily install inside our Device. We use ESP8266 ESP-01, the basic model with minimum IOs.
3) **AWS Cloud**: AWS cloud acts as a bridge between the embedded Device we are debugging and the remote display we want to display the debug message. Before we use the AWS cloud, we need to configure it for IoT operation. For that, we need to create an AWS IoT account, and then we need to create an AWS device. Several documents are available online on how to create an AWS account. We have listed a couple of our papers describing the creation of an IoT inside the AWS cloud server.
4) **C# Application**: The c # application displays the received content from that AWS cloud. When the application started, MQTT clients were created. It is responsible for fetching updates from the AWS IoT shadow register.
5) **IoT Client**: It is a C# plug ins we add with our C# application. It is responsible for receiving the data from the AWS IoT cloud. We must provide AWS IoT credentials to communicate with the AWS server. It is available when we create an AWS IoT profile, and Shadow register.
6) **Wifi Router**: The wifi router provided the Internet connection to the Wifi module ESP8266 module. In our ESP module firmware, we add the Wifi router credential. When our ESP module is powered up, it tries to connect with Wifi using hardcoded credentials. Once connected, it tries to connect with the AWS cloud IoT Server. If the connection is successful, the ESP module can update the IoT shadow register to the cloud.
7) **Terminal**: we add a list box in our C# application. It displays the received content from the AWS cloud.

5. EXPERIMENT :

Now we do some experiments. Before we start practical, we need to learn how to create IoT inside the AWS cloud. We included some of our research papers under the recommend section for reference. Once we are confident, then we need to follow the below instruction.

1) Create IoT credentials inside the AWS server.
2) Download and install Microsoft visual studio from their respective website.
3) Download and install Arduino IDE.
4) Download the Code from the GitHub link: https://github.com/sudipchakraborty/IoT-Based-Indusrtrial-Debug-Message-Display-Using-AWS-ESP8266-And-C_Sharp.git.
5) The IoT credentials save inside the "**..\IoT_Debugger\IoT_Debugger\bin\Debug**" folder. We need to add two files. One is "**AmazonRootCA1.pem**", and the other is the "**device_certificate.cert.pfx"** file.
6) Under the IoT_Debugger folder, open "**IoT_Debugger.sln**". Install a couple of dependency packages using the Nuget package manager.
7) Build and run the project. One window with a black screen will open like in Figure 2
8) Buy one ESP8266 ESP-01 module. It is available online.

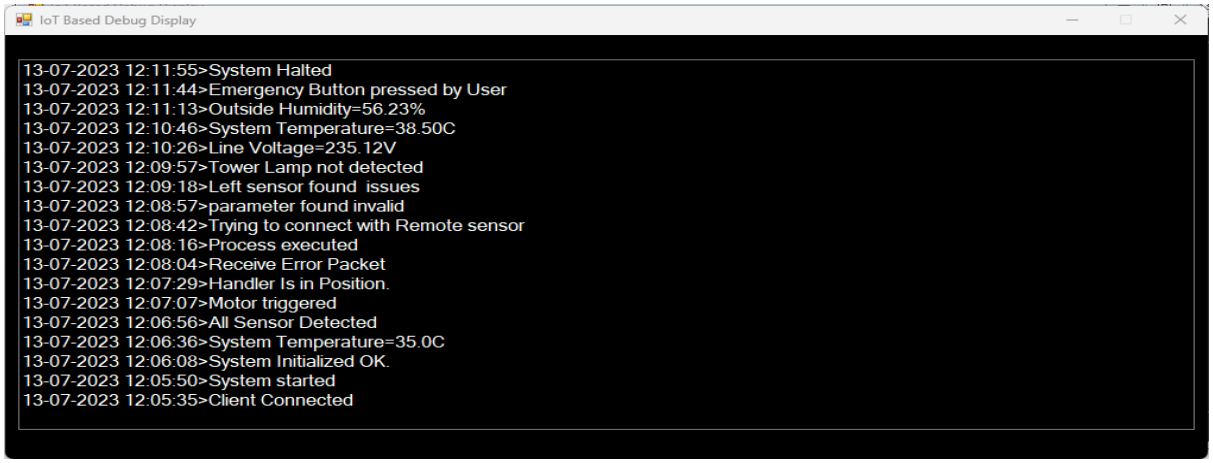

Fig. 2: Application Terminal Window [Source: Author's]

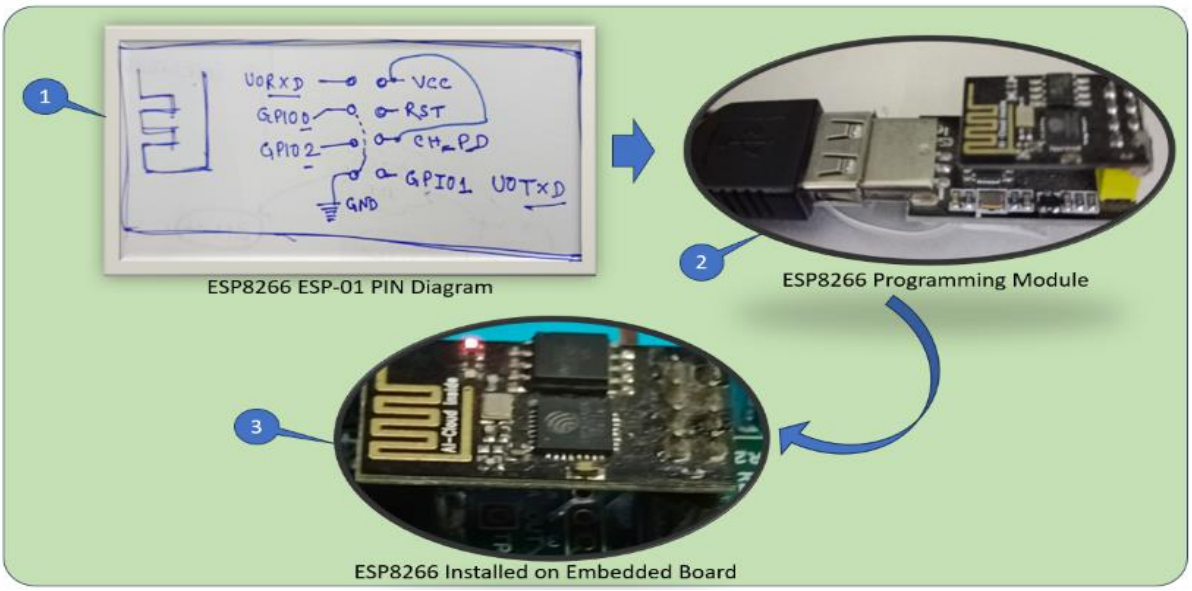

Fig. 3: ESP8266 ESP-01 Module [Source: Author's]

9) Buy one programmer for the ESP-01 module depicted in Figure 3, marking in 2. It is available online at https://www.amazon.in/Techtonics-ESP-01-ESP8266-Adapter-Programmer/dp/B08GG78ZH5/ref=sr_1_2?keywords=esp01+programmer&qid=1689495285&sprefix=esp01+%2Caps%2C451&sr=8-2.
10) We need to connect the Wifi module to the microcontroller according to Figure 3, marked 1.
11) Now under the Arduino_ESP8266 folder, Open the ESP8266_IoT.ino file. Add the AWS credential inside the "Secrets. h" file.
12) Build and connect the Wifi module using the programmer to upload the Code.
13) Select the proper COM PORT and upload the firmware.
14) In the Arduino terminal, type some string and press enter. The typed character is displayed inside the C# application terminal, like Figure 2.
15) Open the AWS IoT and observe that the Topic updates when we send the data, like in Figure 4.
16) If it does not happen as expected, be calm and debug according to the data flow.
17) Finally, Install the wifi module inside the Device like in Figure 3 marked 3.

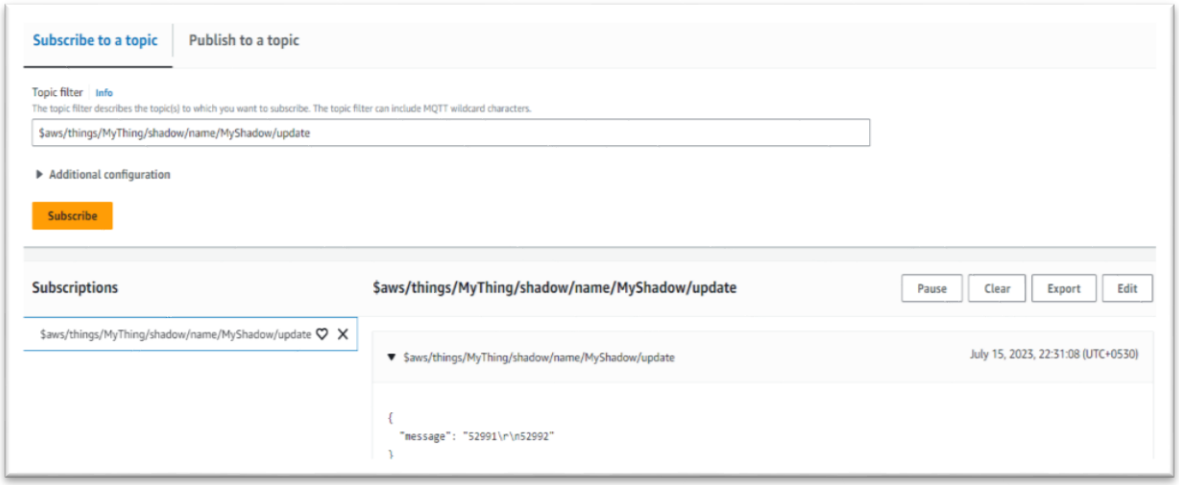

Fig. 4: AWS Shadow register Update [Source: Author's]

6. RECOMMENDATIONS :

- IoT device creation Inside the AWS: https://www.srinivaspublication.com/journal/index.php/ijcsbe/article/view/2283/875 [11]
- Create physical IoT Device: https://www.srinivaspublication.com/journal/index.php/ijmts/article/view/2321/881 [12]
- To Create Multiple IoT device Controller: https://www.srinivaspublication.com/journal/index.php/ijaeml/article/view/2359/896 [13]
- To Create desktop IoT Soft switchboard: https://supublication.com/index.php/ijaeml/article/view/25/22 [14]
- To create Alexa Skill in the AWS cloud server: https://supublication.com/index.php/ijcsbe/article/view/35/31 [15]
- To create a Lambda function inside the AWS server: https://supublication.com/index.php/ijmts/article/view/45/40 [16]
- complete Code we provided is we kept easy to understand better.
- Experience researchers can add more functionality.
- We did not add the exception handling due to the complexity of the Code. The researcher must add it before going for production [17].

7. CONCLUSION :

Sometimes, we cannot reach the Device to observe the debug message in the product's initial phase or product service phase. Here, Through the step-by-step procedure, we created a debug message display without wire using the Wifi and AWS IoT. Following this procedure, we can do various activities with wirelessly connected remote Devices, which is sometimes essential for the researcher. It is also helpful for remote debug message display, updating the firmware, and updating various device parameter configurations remotely.

REFERENCES :

[1] Froiz-Míguez, I., Fernández-Caramés, T. M., Fraga-Lamas, P., & Castedo, L. (2018). Design, implement, and practically evaluate an IoT home automation system based on MQTT and ZigBee-WiFi sensor nodes for fog computing applications. Sensors, 18(8), 2660. Google Scholar↗

[2] Burd, B., Barker, L., Divitini, M., Perez, F. A. F., Russell, I., Siever, B., & Tudor, L. (2018, January). Courses, content, and tools for Internet of things in computer science education. In Proceedings of the 2017 ITiCSE conference on working group reports (pp. 125-139). Google Scholar↗

[3] Al Mubarak, O. M. M. (2022). Design and implementation of an industrial Internet of Things based Scada system: Case study the petroleum pipeline systems (Master's thesis, İstanbul Gelişim Üniversitesi Lisansüstü Eğitim Enstitüsü). Google Scholar↗

[4] Uddin, G., Sabir, F., Guéhéneuc, Y. G., Alam, O., & Khomh, F. (2021). An empirical study of IoT Topics in IoT developer discussions on stack overflow. *Empirical Software Engineering*, *26*(1), 1-45. Google Scholar↗

[5] ŞİMŞEK, M. M., & ATILGAN, E. Attacks on Availability of IoT Middleware Protocols: A Case Study on MQTT. *Eskişehir Türk Dünyası Uygulama ve Araştırma Merkezi Bilişim Dergisi, 4*(2), 16-27. Google Scholar↗

[6] Baronia, R. (2017). Rescuer: Emergency Mobile App with Voice Recognition, Volume Key Pattern, Location SMS Reciprocation, & Push-Aid Systems. 01-35. Google Scholar↗

[7] Widerberg Palmfeldt, A., & Mattsson, W. (2023). Testing IoT Security: A comparison of existing Penetration testing frameworks and proposing a generic framework, 01-61. Google Scholar↗

[8] Balau, R. A. F. F. (2020). Exploring Azure: Internet of Things and Edge (Doctoral dissertation, Universidade do Porto (Portugal)). Google Scholar↗

[9] Στούλος, A. (2019). Internet of things in agriculture. An architectural proposal for an automated watering system based on a wireless sensors' network, 01-77. Google Scholar↗

[10] Borba, B. D. (2018). Serviço de descoberta para o protocolo MQTT em um sistema de monitoramento de grupos motor-gerador baseado em internet das coisas (Doctoral dissertation). 01-57. Google Scholar↗

[11] Chakraborty, S., & Aithal, P. S. (2023). Let Us Create an IoT Inside the AWS Cloud. *International Journal of Case Studies in Business, IT, and Education (IJCSBE), 7*(1), 211-219. Google Scholar↗

[12] Chakraborty, S., & Aithal, P. S. (2023). Let Us Create a Physical IoT Device Using AWS and ESP Module. *International Journal of Management, Technology, and Social Sciences (IJMTS), 8*(1), 224-233. DOI: https://doi.org/10.5281/zenodo.7779097

[13] Chakraborty, S. & Aithal, P. S. (2023). Let Us Create Multiple IoT Device Controller Using AWS, ESP32 And C#. *International Journal of Applied Engineering and Management Letters (IJAEML), 7*(2), 27-34. DOI: https://doi.org/10.5281/zenodo.7857660

[14] Chakraborty, S. & Aithal, P. S. (2023). Let Us Create an Alexa Skill for Our IoT Device Inside the AWS Cloud. *International Journal of Case Studies in Business, IT, and Education (IJCSBE), 7*(2), 214-225. ISSN: 2581-6942. DOI: https://doi.org/10.5281/zenodo.7940237

[15] Chakraborty, S. & Aithal, P. S. (2023). Let Us Create an Alexa Skill for Our IoT Device Inside the AWS Cloud. *International Journal of Case Studies in Business, IT, and Education (IJCSBE), 7*(2), 214-225. DOI: https://doi.org/10.5281/zenodo.7940237.

[16] Chakraborty,S.,& Aithal, P. S. (2023). Let Us Create A Lambda Function for Our IoT Device In The AWS Cloud Using C#. *International Journal of Management, Technology, and Social Sciences (IJMTS), 8*(2), 145-155. DOI: https://doi.org/10.5281/zenodo.7995727.

[17] Chakraborty, S., & Aithal, P. S. (2023). Let Us Create Our Desktop IoT Soft-Switchboard Using AWS, ESP32 and C#. *International Journal of Case Studies in Business, IT, and Education (IJCSBE), 7*(3), 185-193. DOI: https://doi.org/10.5281/zenodo.8234036

****