RANCANG BANGUN MODUL LATIH *AUTOMATIC BOTTLE FILLING AND CAPPING SYSTEM* BERBASIS *PLC* DENGAN MONITORING SCADA

TUGAS AKHIR

Radhifka Defandy Putra

2303321069

PROGRAM STUDI ELEKTRONIKA INDUSTRI

JURUSAN TEKNIK ELEKTRO

POLITEKNIK NEGERI JAKARTA

2026

PENGEMBANGAN SISTEM MONITORING DAN SUPERVISI SCADA TERINTEGRASI IOT PADA *AUTOMATIC BOTTLE FILLING AND CAPPING SYSTEM* BERBASIS PLC

TUGAS AKHIR

Diajukan guna melengkapi sebagian syarat dalam mencapai gelar Program Pendidikan Diploma III (D3)

Radhifka Defandy Putra
2303321069

PROGRAM STUDI ELEKTRONIKA INDUSTRI

JURUSAN TEKNIK ELEKTRO

POLITEKNIK NEGERI JAKARTA

2026

HALAMAN PERYATAAN ORISINALITAS

Tugas Akhir ini adalah hasil karya saya sendiri dan semua sumber baik yang dikutip maupun dirujuk telah saya nyatakan dengan benar.

Nama	**:**	**Radhifka Defandy Putra**
NIM	**:**	**2303321069**
Program Studi	**:**	**Elektronika Industri**
Tanggal	**:**	**Depok, 24/6/2026**
Tanda Tangan	**:**	

LEMBAR PENGESAHAN

TUGAS AKHIR

Tugas Akhir diajukan oleh	:	
Nama	:	Radhifka Defandy Putra
NIM	:	2303321069
Program Studi	:	Elektronika Industri
Judul Tugas Akhir	:	Rancang Bangun Modul Latih *Automatic Bottle Filling and Capping System* Berbasis PLC dengan Monitoring SCADA.
Sub-Judul Tugas Akhir	:	Pengembangan Sistem Monitoring dan Supervisi SCADA Terintegrasi IoT pada *Automatic Bottle Filling and Capping System* Berbasis PLC

Telah diuji oleh tim penguji dalam sidang Tugas Akhir pada hari senin
Tanggal 29-06-2026 dan dinyatakan **LULUS**.

Pembimbing I : Rizdam Firly Muzakki, S.Pd., M.T. ( )
NIP. 199311082024061001

Depok, 10 Juli 2026

Disahkan oleh :
Ketua Jurusan Teknik Elektro

Dr. Murie Dwiyaniti, S.T., M.T.
NIP. 197803312003122002

KATA PENGANTAR

Puji syukur saya panjatkan kepada Tuhan Yang Maha Esa, karena atas berkat dan rahmat-Nya, penulis dapat menyelesaikan Tugas Akhir ini. Penulisan Tugas Akhir ini dilakukan dalam rangka memenuhi salah satu syarat untuk mencapai gelar Diploma Tiga. Tugas Akhir yang penulis buat adalah Pengembangan Sistem Monitoring dan Supervisi SCADA Terintegrasi IoT pada *Automatic Bottle Filling and Capping System* Berbasis PLC. Penulis menyadari bahwa, tanpa bantuan dan bimbingan dari berbagai pihak, dari masa perkuliahan sampai pada penyusunan tugas akhir ini, sangatlah sulit bagi penulis untuk menyelesaikan tugas akhir ini. Oleh karena itu, penulis menguncapkan terima kasih kepada:

1. Ibu Dr. Murie Dwiyaniti, S.T., M.T. selaku Ketua Jurusan Teknik Elektro;
2. Bapak Ihsan Auditia Akhinov, S.T., M.T., selaku dosen Kepala Program Studi D3-Elektronika Industri
3. Bapak Rizdam Firly Muzakki, S.Pd., M.T., selaku dosen pembimbing yang telah memberikan arahan, dukungan, dan bantuan dalam penyelesaian Tugas Akhir.
4. Orang tua dan keluarga penulis yang telah memberikan dukungan moral, material, serta doa dalam menyusun dan menyelesaikan Tugas Akhir.
5. Rekan satu tim yang telah banyak membantu penulis dalam menyusun dan menyelesaikan Tugas Akhir.

Akhir kata, penulis berharap Tuhan Yang Maha Esa berkenan membalas segala kebaikan semua pihak yang telah membantu. Semoga Tugas Akhir ini membawa manfaat bagi pengembangan ilmu.

ABSTRAK

Tugas Akhir ini mengimplementasikan dan menguji sistem gateway berbasis mikrokontroler ESP32S untuk menjembatani komunikasi lokal Modbus RTU pada *Programmable Logic Controller* (PLC) Huikong FX3U – 28MR dengan protokol MQTT. Hasil pengujian menunjukkan bahwa *gateway* beroperasi secara andal; proses polling data Modbus pada baud rate 9600bps berjalan stabil dengan waktu rata-rata 84 ms, sementara konversi data ke format JSON hingga proses *publish* ke broker MQTT tereksekusi dalam 4,46ms. Pada pengujian komunikasi jaringan WiFi dengan RSSI sinyal -62dBm, sistem menghasilkan latensi rata-rata 192,54ms dengan rasio keberhasilan pengiriman data mencapai 100% untuk kebutuhan monitoring real-time. Integrasi perangkat keras dengan perangkat lunak Haiwell Cloud SCADA berhasil dilakukan sepenuhnya (100%), memungkinkan instruksi kontrol dari antarmuka SCADA dikirim secara presisi ke plant melalui perintah MQTT yang diteruskan menjadi instruksi tulis Modbus. Selain itu, mekanisme pemulihan perangkat lunak terbukti efektif dengan tingkat keberhasilan sebesar 100% saat terjadi simulasi kegagalan jaringan, sehingga sistem terhindar dari *blocking* permanen.

Kata kunci: SCADA, ESP32, IoT *Gateway*, PLC, Modbus RTU, MQTT.

ABSTRACT

This Final Assignment implements and tests an ESP32S microcontroller-based gateway system to bridge local Modbus RTU communication on a Huikong FX3U-28MR Programmable Logic Controller (PLC) with the MQTT protocol. The test results indicate that the gateway operates reliably; the Modbus data polling process at a 9600 bps baud rate runs stably with an average time of 84ms, while the data conversion to JSON format up to the publishing process to the MQTT broker is executed in 4.46ms. During Wi-Fi network communication testing with a signal attenuation of -62dBm, the system produced an average latency of 192.54ms with a 100% data transmission success rate for real-time monitoring needs. Hardware integration with the Haiwell Cloud SCADA software was fully accomplished (100%), enabling control instructions from the SCADA interface to be sent precisely to the plant via MQTT commands that are forwarded as Modbus write instructions. Furthermore, the software recovery mechanism proved effective with a 100% success rate during simulated network failures, thereby preventing the system from experiencing permanent blocking.

Keywords: SCADA, ESP32, IoT Gateway, PLC, Modbus RTU, MQTT.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan industri manufaktur sudah dari lama berjalan dengan kontrol dan indikator fisik, lalu berkembang ke HMI, SCADA dan yang sedang pesat sekarang adalah IoT(Aliyu Enemosah & Ogbonna George Ifeanyi, 2024).

Sistem IoT yang umumnya mengandalkan protokol TCP Rest API terasa kurang efisien karena memiliki sifat *stateless* yang mengharuskan perangkat berulang kali membuka dan menutup koneksi TCP(Esposito et al., 2023) sehingga dikembangkanlah WebSocket, namun kedua protokol tersebut memiliki kelemahan, dari yang Rest API yang boros karena membutuhkan *request* untuk mengecek setiap perubahan data, WebSocket yang memiliki header yang bisa dibilang boros untuk penggunaan IoT(Tsaqief & Sutopo, 2025) serta kekurangan mutlak dari kedua protokol tersebut adalah memerlukan *routing*, server dan logika kode untuk setiap projek(Amirkhanov et al., 2025).

MQTT menjawab kekurangan tersebut dengan sistem *publish* dan *subscribe* dengan penengah broker, server hanya dibutuhkan 1 yaitu server untuk hosting broker(Jesus et al., 2025). MQTT juga memiliki topik, setiap projek IoT yang dibuat dapat terhubung ke 1 broker dengan topik yang berbeda, setiap topic juga dapat dikunci dengan *username* dan *password* tertentu untuk privasi dan keamanan setiap projek.

Adapun masalah lain yaitu dari segi kontroler, masih banyak PLC yang tidak memiliki fitur komunikasi MQTT atau bahkan Ethernet secara default, tetapi masih memiliki cara komunikasi untuk area local seperti RS-485 dan RS-232 dengan protokol komunikasi Modbus(Nurahmat & Suryo, 2025).

Menjawab masalah tersebut, suatu hub atau *gateway* diperlukan agar perubahan data pada PLC dapat dikirim ke tempat lain melalui jaringan

internet secara realtime. *Gateway* tersebut harus mampu untuk melakukan banyak *request* ke Modbus Slave pada jalur komunikasi RS-485 lalu memuat data tersebut menjadi paket yang dapat dikirim secara efisien melalui protokol MQTT, dan juga melakukan sebaliknya yaitu menunggu paket data tiba dari MQTT dan mengubah nilai pada alamat modbus yang berkaitan.

Oleh karena itu, modul IoT sebagai Modbus MQTT *Gateway* ini dibuat untuk mengatasi masalah *polling task* yang umumnya terjadi pada implementasi MQTT pada ESP32 dengan arsitektur superloop pada sistem bare metal berbasis Arduino *Framework* yang akan menghabisakan kebanyakan siklus instruksi pada CPU hanya untuk melakukan check apakah ada paket MQTT yang datang(Espino et al., 2024). Masalah ini diselesaikan dengan cara menggunakan arsitektur *Event Driven* yang dimana *Task* MQTT tersebut akan dalam keadaan *sleep* hingga ada paket MQTT yang diterima dari topik yang di-*subscribe*(Bertoli et al., 2024).

Selain itu, modul ini juga mengatasi masalah pengembangan sistem IoT yang memakan waktu lama karena memerlukan pemrograman terlebih dahulu, yaitu dirancang Website yang berjalan lokal untuk konfigurasi WiFi, broker MQTT serta kredensialnya, topik MQTT untuk *publish* dan *subscribe*, Modbus, serta *maaping* pada alamat modbus dan variable tag SCADA agar pengguna tidak memerlukan pemrograman sama sekali dan dapat melakukan semua konfigurasi tersebut melalui antarmuka Website yang interaktif.

1.2 Perumusan Masalah

Berdasarkan latar belakang tersebut, dapat dirumuskan beberapa masalah yang akan dibahas dalam penelitian ini, yaitu:

1. Bagaimana mengimplementasikan sistem monitoring *SCADA* pada Modul Latih *Automatic Bottle Filling & Capping System* untuk menampilkan status proses secara *real-time* sebagai bagian dari sistem pengendalian industri?
2. Bagaimana mentransmisikan data dari Modbus yang ada di PLC ke MQTT

melalui jaringan internet secara efisien?

3. Optimisasi kode seperti apa yang akan dilakukan agar mikrokontroler dapat menanggulangi masalah ketika terjadi crash atau jaringan yang terputus saat tengah digunakan?

1.3 Tujuan

Berdasarkan rumusan masalah di atas, maka tujuan dari penelitian ini adalah sebagai berikut:

1. Membuat Antarmuka Web yang dapat digunakan untuk konfigurasi kredensial WiFi, MQTT, Konfigurasi dasar Modbus serta Konfigurasi Mapping alamat Modbus.
2. Membuat algoritma untuk melakukan pengecekan perubahan nilai pada alamat modbus tertentu dan mengelompokannya pada JSON supaya pengiriman data ke MQTT menjadi lebih efisien dan lebih hemat *bandwidth* internet.
3. Mengimplementasikan sistem monitoring SCADA untuk menampilkan status proses secara *real-time* sebagai bagian dari sistem pengendalian dan pengawasan operasi.
4. Melakukan optimisasi pada penggunaan memori yang terbatas pada ESP32, pengunaan heap supaya memori digunakan secara dinamis sesuai kebutuhan, dan melakukan *error handling* pada sistem yang krusial seperti koneksi WiFi dan MQTT.

1.4 Batasan Masalah

Dalam penyusunan penelitian ini, terdapat batasan masalah agar pembahasan lebih fokus dan terarah. Batasan tersebut yaitu :

1. Pengujian dilakukan pada lingkungan di kondisi yang sama yaitu dengan sumber WiFi yang sama dan jarak yang sama dengan halangan 1 tembok, jarak 2 meter dan RSSI -62dBm.

1.5 Luaran

Luaran yang diharapkan dari penelitian tugas akhir ini yaitu:

1. Laporan tugas akhir dan jurnal ilmiah yang memuat perancangan, implementasi, serta pengujian Modul Latih *Automatic Bottle Filling & Capping System* berbasis PLC dengan monitoring SCADA sebagai media pembelajaran otomasi industri.
2. Modul Latih *Automatic Bottle Filling & Capping System* berbasis PLC yang dilengkapi dengan sistem monitoring SCADA dan dapat digunakan sebagai sarana praktikum untuk memahami konsep pengendalian, otomasi, dan monitoring sistem industri.
3. *Jobsheet* (modul praktikum) Modul Latih sebagai panduan penggunaan alat dalam kegiatan pembelajaran.
4. Hasil analisis kinerja sistem yang meliputi lama pengambilan data dari PLC, Pengiriman data ke internet, akurasi sistem *SCADA* dalam menampilkan kondisi operasi secara *real-time*, Serta keberhasilan sistem untuk melakukan pemulihan setelah terjadinya kegagalan koneksi WiFi.

BAB V
PENUTUP

5.1 Kesimpulan

Adapun kesimpulan yang didapatkan dari hasil penelitian, perancangan, dan pengujian yang telah dilakukan terhadap sistem adalah sebagai berikut:

1. Mikrokontroler ESP32S telah berhasil diimplementasikan sebagai *gateway* yang andal untuk menjembatani komunikasi lokal Modbus RTU pada PLC dengan protokol internet MQTT. Proses *polling* data Modbus pada baud rate 9600 bps berjalan stabil dengan rata-rata waktu 84 ms, dan proses konversi data ke format JSON hingga *publish* ke broker MQTT dieksekusi dengan sangat cepat (rata-rata 4,46 ms) tanpa membebani kinerja PLC.
2. Pengujian komunikasi MQTT via jaringan WiFi menunjukkan performa yang sangat baik untuk kebutuhan *real-time monitoring*. Pada kondisi RSSI sinyal -62dBm (terhalang 1 tembok pada jarak 2 meter), latensi rata-rata yang dihasilkan adalah 192,54 ms dengan rasio keberhasilan pengiriman data mencapai 100%.
3. Integrasi antara perangkat keras dan perangkat lunak Haiwell Cloud SCADA terbukti berhasil 100%. Antarmuka SCADA mampu menjalankan instruksi kontrol (*Start, Stop, Emergency*, pengaturan aktuator, dan pompa) secara presisi ke plant (modul latih) melalui perintah *publish* MQTT yang diteruskan menjadi instruksi tulis (*write*) Modbus ke PLC.
4. Mekanisme *retry* dan perlindungan sistem (pengaman perangkat lunak) beroperasi sesuai dengan perancangan. Saat terjadi simulasi kegagalan jaringan WiFi, sistem mampu melakukan *auto-recovery* pada koneksi WiFi, MQTT, dan *polling* Modbus dengan tingkat keberhasilan pemulihan 100%, sehingga sistem terhindar dari *crash* atau *blocking* permanen.

5.2 Saran

1. Pengujian Modbus saat ini menggunakan baud rate standar 9600bps. Disarankan untuk melakukan eksperimen konfigurasi baud rate yang lebih tinggi (seperti 19200bps) guna memangkas waktu polling jika akan dilakukan penambahan jumlah data yang akan dikirimkan ke SCADA berbasis IoT.
2. Sistem SCADA saat ini berfokus pada monitoring dan kontrol real-time. Ke depannya, data dari broker MQTT dapat dihubungkan ke sistem backend berbasis database relasional atau time-series (seperti PostgreSQL dengan ekstensi TimescaleDB) agar riwayat pencatatan status alat dapat disimpan untuk keperluan analisis data historis yang lebih mendalam.

DAFTAR PUSTAKA

Al Sarfini, A. A., & Irawan, D. (2024). Sistem Kontrol Jarak Jauh Plc Menggunakan Esp32 Berbasis Iot. *JURNAL AMPLIFIER : JURNAL ILMIAH BIDANG TEKNIK ELEKTRO DAN KOMPUTER*, *14*(1). https://doi.org/10.33369/jamplifier.v14i1.33484

Aliyu Enemosah, & Ogbonna George Ifeanyi. (2024). SCADA in the Era of IoT: Automation, Cloud-driven security, and machine learning applications. *International Journal of Science and Research Archive*, *13*(1). https://doi.org/10.30574/ijsra.2024.13.1.1975

Amirkhanov, B., Amirkhanova, G., Kunelbayev, M., Adilzhanova, S., & Tokhtassyn, M. (2025). Evaluating HTTP, MQTT over TCP and MQTT over WEBSOCKET for digital twin applications: A comparative analysis on latency, stability, and integration. *International Journal of Innovative Research and Scientific Studies*, *8*(1), 679–694. https://doi.org/10.53894/ijirss.v8i1.4414

Arrahma, S. A., & Mukhaiyar, R. (2023). Pengujian Esp32-Cam Berbasis Mikrokontroler ESP32. *JTEIN: Jurnal Teknik Elektro Indonesia*, *4*(1). https://doi.org/10.24036/jtein.v4i1.347

Bertoli, G. de C., Fernandes, G. V. C., Monici, P. H. B., Guibo, C. H. de A., Dos Santos, A. L., & Pereira Junior, L. A. (2024). Design and implementation of intelligent packet filtering in IoT microcontroller-based devices. *Journal of Internet Services and Applications*, *15*(1). https://doi.org/10.5753/jisa.2024.3835

Duguma, A. L., & Bai, X. (2025). How the internet of things technology improves agricultural efficiency. *Artificial Intelligence Review*, *58*(2). https://doi.org/10.1007/s10462-024-11046-0

Espino, C. S., Vargas, F. G., Paiva-Peredo, E., & Segura, G. W. Z. (2024). Internet of things and radio frequency identification based embedded system to reduce shopping time in supermarkets. *Bulletin of Electrical Engineering and Informatics*, *13*(4). https://doi.org/10.11591/eei.v13i4.7343

Esposito, M., Belli, A., Palma, L., & Pierleoni, P. (2023). Design and Implementation of a Framework for Smart Home Automation Based on Cellular IoT, MQTT, and Serverless Functions. *Sensors*, *23*(9). https://doi.org/10.3390/s23094459

Gnana, G. S., Karthikeyan, A., Karthikeyan, K., Sanjeevikumar, P., Karapparambil Thomas, S., & Babu, A. (2024). Critical review Of SCADA And PLC in smart buildings and energy sector. In *Energy Reports* (Vol. 12). https://doi.org/10.1016/j.egyr.2024.07.041

Jesus, B., Lins, F., & Laranjeiro, N. (2025). An approach to assess robustness of MQTT-based IoT systems. *Internet of Things (The Netherlands)*, *31*. https://doi.org/10.1016/j.iot.2025.101590

Kachman, O., Malík, P., Baláž, M., Majer, L., & Gyepes, G. (2025). A Lightweight and Configurable Flash Filesystem for Low-Power Devices. *Journal of Low Power Electronics and Applications*, *15*(2). https://doi.org/10.3390/jlpea15020022

Kusuma, J. F., Rifa'i, M., & Saukani, I. (2024). Implementasi Protokol Komunikasi Modbus Untuk Mini Scada Pada Plant Pengisian Serbuk Temulawak. *Mutiara: Multidiciplinary Scientifict Journal*, *2*(5). https://doi.org/10.57185/mutiara.v2i5.182

Nadhiroh, N., Wardhany, A. K., Setiana, H., Renaldy, R., Putri, A. A., & Handayani, M. D. (2024). Penyiram Tanaman Hidroponik Otomatis Berbasis IoT Dengan PLC Outseal Dan ESP32. *ELECTRICES*, *6*(1). https://doi.org/10.32722/ees.v6i1.6361

Nurahmat, H., & Suryo, Y. A. (2025). Implementation of Outseal PLC with RS-485 as a Multi Node in Chicken Farming Using Raspberry Pi and Node RED. *G-Tech: Jurnal Teknologi Terapan*, *9*(4). https://doi.org/10.70609/g-tech.v9i4.8155

Priyatna, I. A., Darmawan, B., & Paniran. (2024). Rancang Bangun Sistem Monitoring Parkir Mobil Indoor Dengan Wireless Sensor Network Menggunakan Nodemcu Esp8266 Berbasis Internet Of Things.

DIELEKTRIKA, *11*(2). https://doi.org/10.29303/dielektrika.v11i2.366

Rafique, W. (2025). ML-RASPF: A Machine Learning-Based Rate-Adaptive Framework for Dynamic Resource Allocation in Smart Healthcare IoT. *Algorithms*, *18*(6). https://doi.org/10.3390/a18060325

Tsaqief, M. F., & Sutopo, J. (2025). Comparative Performance Analysis Between the MQTT and WebSocket Protocols. *Bit-Tech*, *8*(2). https://doi.org/10.32877/bt.v8i2.3223

Xu, J., Fan, L., Du, J., Wang, X., Qin, L., Ding, X., Wang, Z., & Tao, Y. (2024). Research on the Application of Littlefs and FATFS File Systems in Smart IoT Electricity Meters. *2024 IEEE 4th International Conference on Power, Electronics and Computer Applications, ICPECA 2024*. https://doi.org/10.1109/ICPECA60615.2024.10471164

Yanyachi, P. R., Mendoza-Chok, J., Espinoza-Garcia, B., Cutipa Luque, J. C., & Yanyachi Aco Cardenas, D. (2025). OpenNavSense platform: A low-cost, open-source inertial navigation system for the evaluation of estimation algorithms. *HardwareX*, *21*. https://doi.org/10.1016/j.ohx.2024.e00621

LAMPIRAN

Lampiran 1. Daftar Riwayat Hidup

Radhifka Defandy Putra

Lahir di Bekasi pada tanggal 15 Maret 2005. Lulus dari SDIT Assalam Terpadu pada tahun 2017, SMPN 3 Tambun Utara pada tahun 2020, dan SMKS Karya Guna Bhakti 1 pada tahun 2023. Kemudian melanjutkan kuliah dengan Program Studi Teknik Elektronika Industri, Teknik Elektro di Politeknik Negeri Jakarta pada tahun 2023

Lampiran 2. Datasheet ESP32

Table 1 provides the specifications of ESP-32S.

Table 1 ESP-32S Specifications

Categories	Items	Values
WiFi	Standards	
	Protocles	802.11 b/g/n/d/e/i/k/r (802.11n up to 150 Mbps)
	Frequency Range	2.4GHz-2.5GHz (2400M-2483.5M)
Bluetooth	Protocols	Bluetooth v4.2 BR/EDR and BLE specification
	Radio	NZIF receiver with -98 dBm sensitivity
		Class-1, class-2 and class-3 transmitter
		AFH
	Audio	CVSD and SBC
Hardware	Module interface	SD card, UART, SPI, SDIO, I2C, LED PWM, Motor PWM, I2S, I2C, IR
		GPIO, capacitive touch sensor, ADC, DAC, LNA pre-amplier
	On-chip sensor	3.0~3.6V
	On-board clock	Average value: 80mA
	Operating voltage	-40°~125°
	Operating current	Normal temperature
	Operating temperature range	14.3mm*24.8mm*3mm
	Ambient temperature range	N/A
	Package size	
Software	Wi-Fi mode	Station/softAP/SoftAP+station/P2P
	Security	WPA/WPA2/WPA2-Enterprise/WPS
	Encryption	AES/RSA/ECC/SHA
	Firmware Upgrade	UART Download / OTA (via network) / download and write firmware via host

Lampiran 3. Datasheet MAX485

maxim integrated

MAX481/MAX483/MAX485/ MAX487–MAX491/MAX1487

Low-Power, Slew-Rate-Limited RS-485/RS-422 Transceivers

General Description

The MAX481, MAX483, MAX485, MAX487–MAX491, and MAX1487 are low-power transceivers for RS-485 and RS-422 communication. Each part contains one driver and one receiver. The MAX483, MAX487, MAX488, and MAX489 feature reduced slew-rate drivers that minimize EMI and reduce reflections caused by improperly terminated cables, thus allowing error-free data transmission up to 250kbps. The driver slew rates of the MAX481, MAX485, MAX490, MAX491, and MAX1487 are not limited, allowing them to transmit up to 2.5Mbps.

These transceivers draw between 120µA and 500µA of supply current when unloaded or fully loaded with disabled drivers. Additionally, the MAX481, MAX483, and MAX487 have a low-current shutdown mode in which they consume only 0.1µA. All parts operate from a single 5V supply.

Drivers are short-circuit current limited and are protected against excessive power dissipation by thermal shutdown circuitry that places the driver outputs into a high-impedance state. The receiver input has a fail-safe feature that guarantees a logic-high output if the input is open circuit.

The MAX487 and MAX1487 feature quarter-unit-load receiver input impedance, allowing up to 128 MAX487/MAX1487 transceivers on the bus. Full-duplex communications are obtained using the MAX488–MAX491, while the MAX481, MAX483, MAX485, MAX487, and MAX1487 are designed for half-duplex applications.

Applications

Low-Power RS-485 Transceivers
Low-Power RS-422 Transceivers
Level Translators
Transceivers for EMI-Sensitive Applications
Industrial-Control Local Area Networks

Next Generation Device Features

- **For Fault-Tolerant Applications**
  MAX3430: ±80V Fault-Protected, Fail-Safe, 1/4 Unit Load, +3.3V, RS-485 Transceiver
  MAX3440E–MAX3444E: ±15kV ESD-Protected, ±60V Fault-Protected, 10Mbps, Fail-Safe, RS-485/J1708 Transceivers
- **For Space-Constrained Applications**
  MAX3460–MAX3464: +5V, Fail-Safe, 20Mbps, Profibus RS-485/RS-422 Transceivers
  MAX3362: +3.3V, High-Speed, RS-485/RS-422 Transceiver in a SOT23 Package
  MAX3280E–MAX3284E: ±15kV ESD-Protected, 52Mbps, +3V to +5.5V, SOT23, RS-485/RS-422, True Fail-Safe Receivers
  MAX3293/MAX3294/MAX3295: 20Mbps, +3.3V, SOT23, RS-485/RS-422 Transmitters
- **For Multiple Transceiver Applications**
  MAX3030E–MAX3033E: ±15kV ESD-Protected, +3.3V, Quad RS-422 Transmitters
- **For Fail-Safe Applications**
  MAX3080–MAX3089: Fail-Safe, High-Speed (10Mbps), Slew-Rate-Limited RS-485/RS-422 Transceivers
- **For Low-Voltage Applications**
  MAX3483E/MAX3485E/MAX3486E/MAX3488E/ MAX3490E/MAX3491E: +3.3V Powered, ±15kV ESD-Protected, 12Mbps, Slew-Rate-Limited, True RS-485/RS-422 Transceivers

Ordering Information appears at end of data sheet.

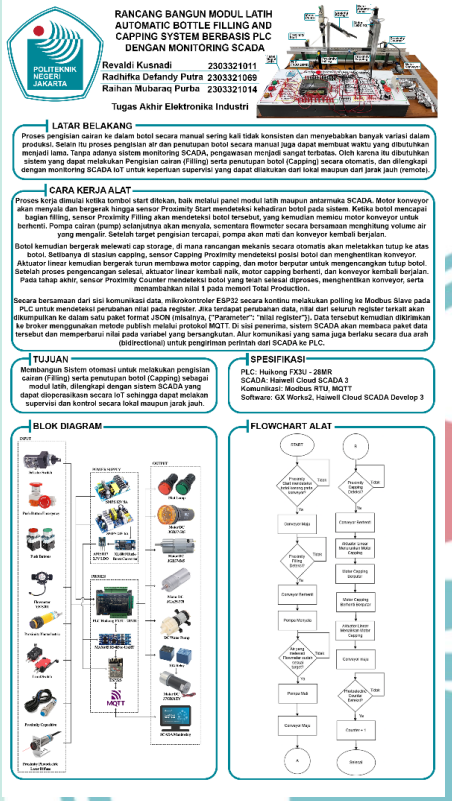
Selection Table

PART NUMBER	HALF/FULL DUPLEX	DATA RATE (Mbps)	SLEW-RATE LIMITED	LOW-POWER SHUTDOWN	RECEIVER/ DRIVER ENABLE	QUIESCENT CURRENT (µA)	NUMBER OF RECEIVERS ON BUS	PIN COUNT
MAX481	Half	2.5	No	Yes	Yes	300	32	8
MAX483	Half	0.25	Yes	Yes	Yes	120	32	8
MAX485	Half	2.5	No	No	Yes	300	32	8
MAX487	Half	0.25	Yes	Yes	Yes	120	128	8
MAX488	Full	0.25	Yes	No	No	120	32	8
MAX489	Full	0.25	Yes	No	Yes	120	32	14
MAX490	Full	2.5	No	No	No	300	32	8
MAX491	Full	2.5	No	No	Yes	300	32	14
MAX1487	Half	2.5	No	No	Yes	230	128	8

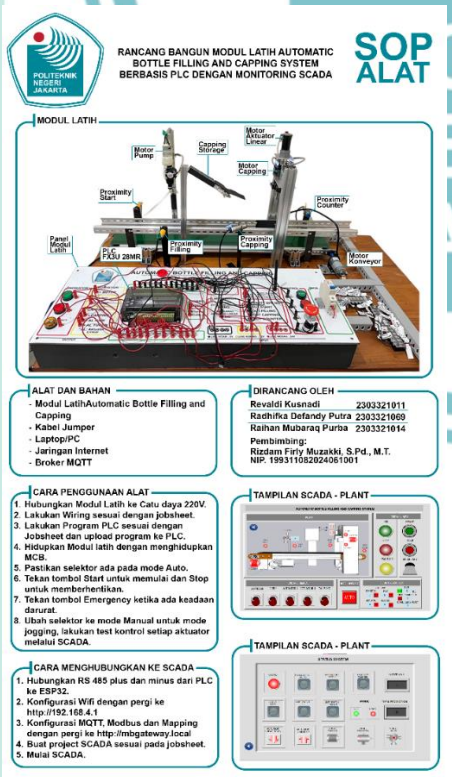
For pricing, delivery, and ordering information, please contact Maxim Direct at 1-888-629-4642, or visit Maxim Integrated's website at www.maximintegrated.com.

19-0122; Rev 10; 9/14

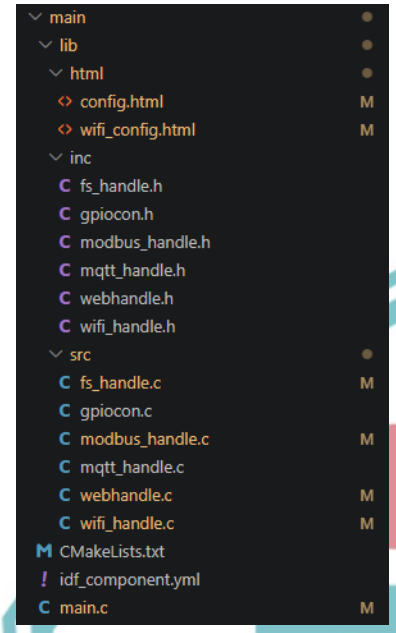
Lampiran 4. Poster Alat



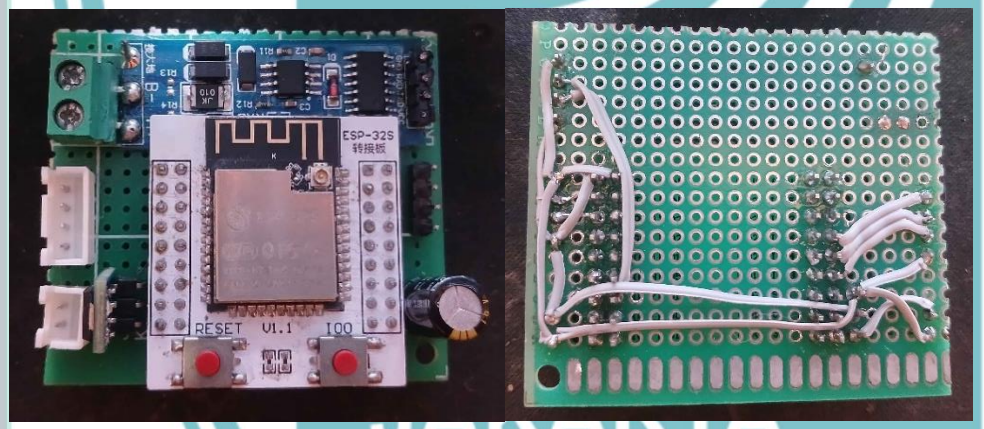
Lampiran 5. SOP Alat



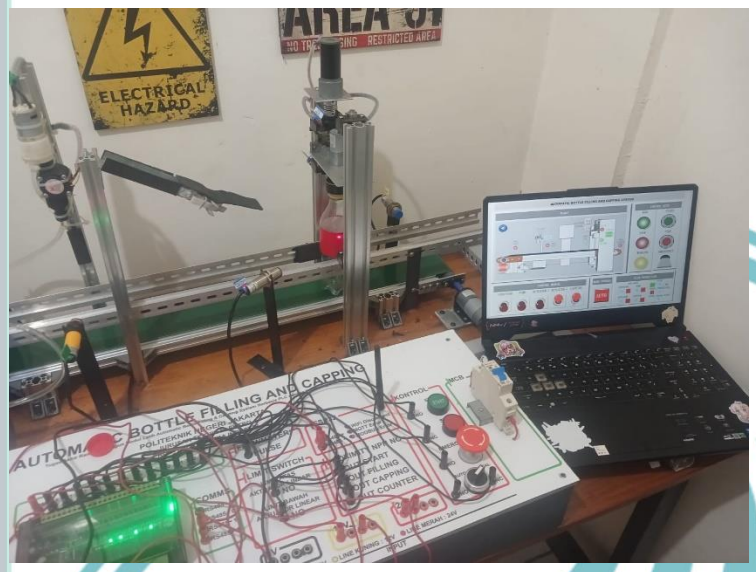
Lampiran 6. Struktur Program



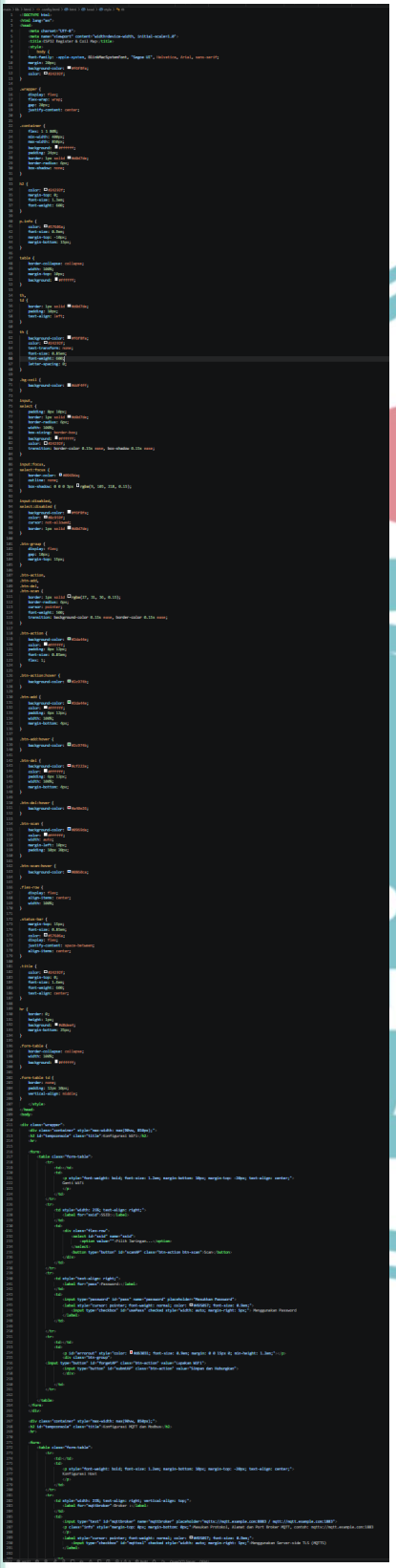
Lampiran 7. Foto Alat

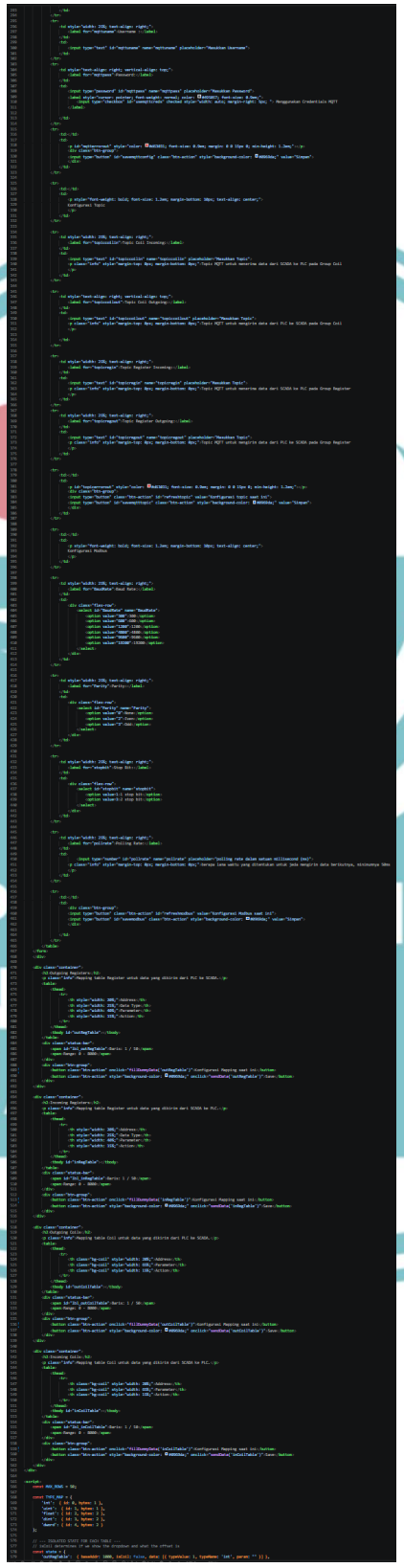


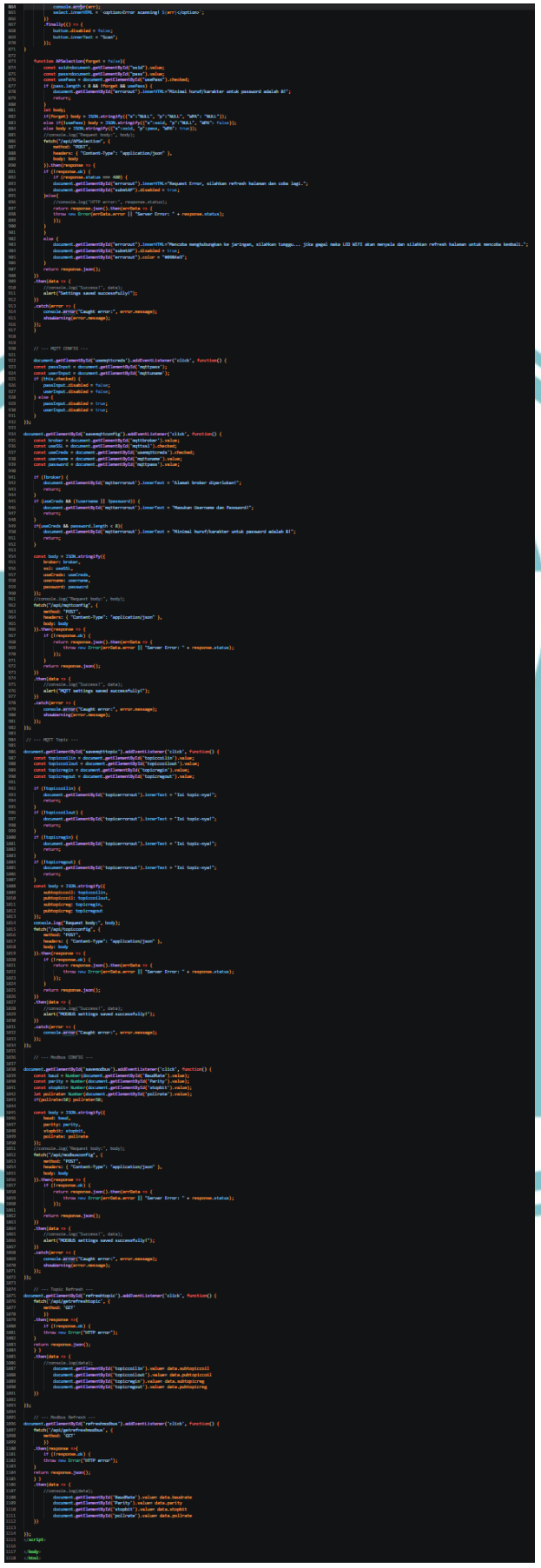
Lampiran 8. Pengujian Alat



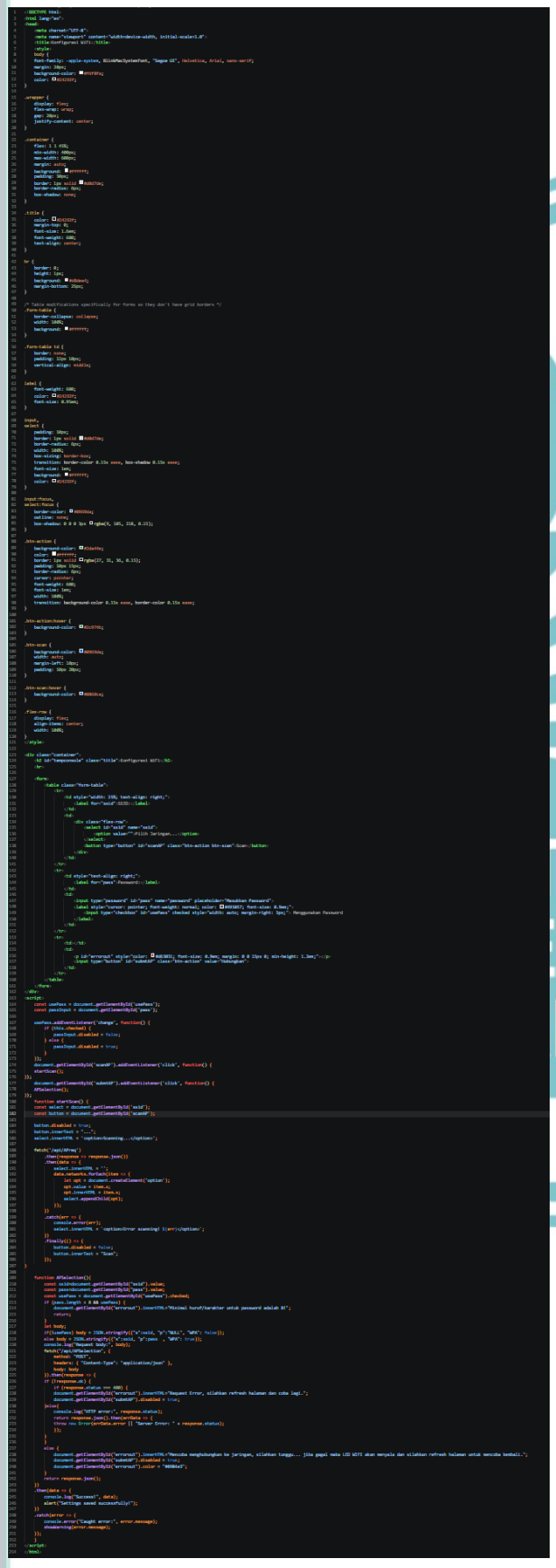
Lampiran 9. Program Modbus MQTT Gateway – config.html







Lampiran 10. Program Modbus MQTT Gateway – wifi_config.html



Lampiran 11. Program Modbus MQTT Gateway – main.c

```c
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_log.h"
#include "nvs_flash.h"
#include "lwip/err.h"
#include "lwip/sys.h"
#include "webhandle.h"
#include "wifi_handle.h"
#include "gpiocon.h"
#include "mqtt_handle.h"
#include "modbus_handle.h"
#include "esp_heap_caps.h"
#include "fs_handle.h"

//static const char *TAG = "MEM";
//static const char *TAG= "Main";

void app_main(void){
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
      ESP_ERROR_CHECK(nvs_flash_erase());
      ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(ret);
    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());
    esp_netif_create_default_wifi_sta();

    esp_log_level_set("*", ESP_LOG_WARN);
    wifi_event_init();
    wifi_init();
    init_littlefs();
    server_init();
    wificonfig_serveron();
    gpioinit();
    init_modbus_config();
    xTaskCreate(modbus_polling_task, "modbus_task", 4096, NULL, 5, NULL);
}
```

Lampiran 12. Program Modbus MQTT Gateway – fs_handle.c dan fs_handle.h

```c
#include "esp_littlefs.h"
#include <stdlib.h>
#include "fs_handle.h"
#include "esp_log.h"
#include "nvs.h"

static const char *TAG = "FS_Handle";

void init_littlefs(void){
    esp_vfs_littlefs_conf_t conf = {
        .base_path = "/littlefs",
        .partition_label = "storage",
        .format_if_mount_failed = true,
    };

    esp_err_t ret = esp_vfs_littlefs_register(&conf);
    ESP_ERROR_CHECK(ret);
}

void savefile(const char* filename, const void* data, size_t size) {
    char path[64];
    snprintf(path,sizeof(path), "/littlefs/%s", filename);
    FILE *f = fopen(path, "wb");
    if (!f) {
        ESP_LOGE(TAG, "Failed to open file for writing");
        return;
    }
    if (!data) {
    ESP_LOGE(TAG, "NULL data");
    return;
    }
    fwrite(data, size, 1, f);
    fclose(f);
}

void readfile(const char* filename, void* data, size_t size) {
    char path[64];

    int len = snprintf(path, sizeof(path), "/littlefs/%s", filename);
    if (len >= sizeof(path)) {
        ESP_LOGE(TAG, "Path too long");
        return;
    }

    FILE *f = fopen(path, "rb");
    if (!f) {
        ESP_LOGE(TAG, "Failed to open file for reading");
        return;
    }

    if (!data || size == 0) {
        ESP_LOGE(TAG, "Invalid buffer");
        fclose(f);
        return;
    }

    size_t read = fread(data, 1, size, f);

    if (read != size) {
        ESP_LOGW(TAG, "Read size mismatch: %d/%d", read, size);
    }
    ESP_LOGI(TAG, "Actually read: %d bytes", read);
    fclose(f);
}

void savenvs(const char* key, void* data, size_t size) {
    nvs_handle_t nvs;
    esp_err_t err = nvs_open("storage", NVS_READWRITE, &nvs);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "NVS open fail");
        return;
    }
    err = nvs_set_blob(nvs, key, data, size);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "NVS WR fail");
    }
    nvs_close(nvs);
}

void readnvs(const char* key, void* value, size_t size){
    nvs_handle_t nvs;
    esp_err_t err = nvs_open("storage", NVS_READWRITE, &nvs);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "NVS open fail");
        return;
    }
    size_t required_size = 0;
    err = nvs_get_blob(nvs, key, NULL, &required_size);
    if (err == ESP_ERR_NVS_NOT_FOUND) {
        ESP_LOGW(TAG, "'%s' not in NVS", key);
        nvs_close(nvs);
        return;
    } else if (err != ESP_OK) {
        ESP_LOGE(TAG, "NVS RD fail");
        nvs_close(nvs);
        return;
    }
    if (required_size > size) {
        ESP_LOGE(TAG, "buffer too small for key '%s'", key);
        nvs_close(nvs);
        return;
    }
    err = nvs_get_blob(nvs, key, value, &required_size);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "NVS RD fail");
    }
    nvs_close(nvs);
}

void removenvs(const char* key){
    nvs_handle_t nvs;
    esp_err_t err = nvs_open("storage", NVS_READWRITE, &nvs);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "NVS open fail");
        return;
    }
    err = nvs_erase_key(nvs, key);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "NVS erase fail");
    }
    nvs_close(nvs);
}
```

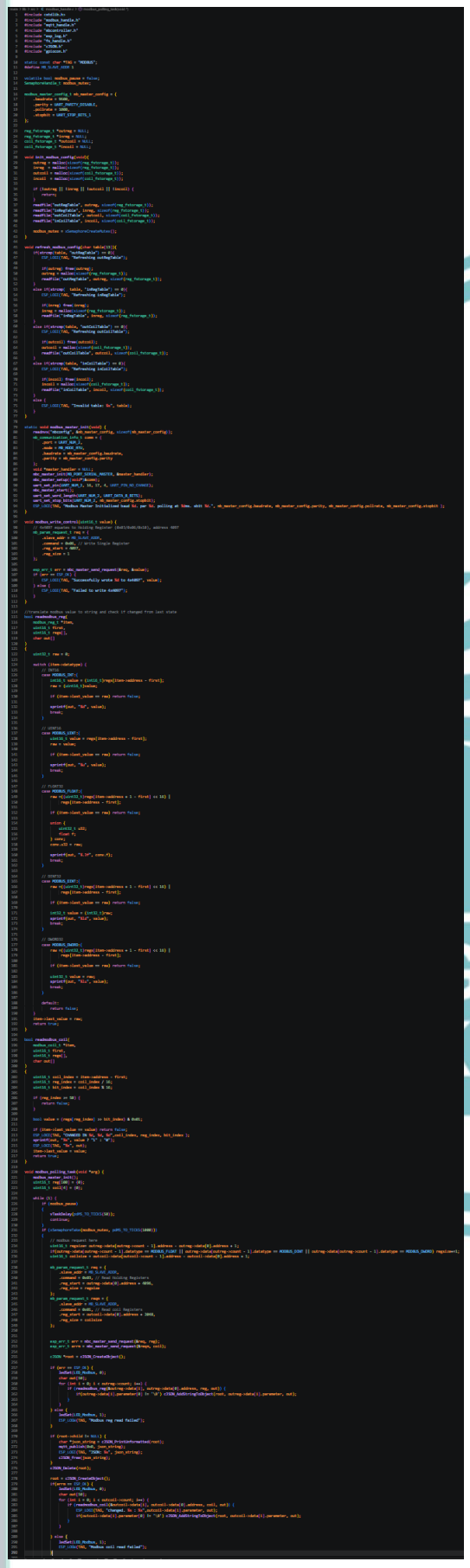
```c
#ifndef FS_HANDLE_H
#define FS_HANDLE_H

void init_littlefs(void);
void savefile(const char* filename, const void* data, size_t size);
void readfile(const char* filename, void* data, size_t size);
void savenvs(const char* key, void* data, size_t size);
void readnvs(const char* key, void* data, size_t size);
void removenvs(const char* key);

#endif
```

Lampiran 13. Program Modbus MQTT Gateway – gpiocon.c dan gpiocon.h

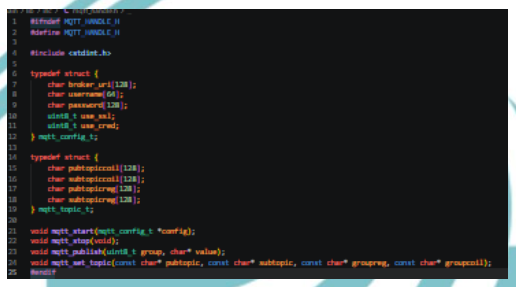
```c
#include "driver/gpio.h"
#include "esp_timer.h"
#include "gpiocon.h"

typedef struct {
    int gpio;
    int state;
    int blinking;
    esp_timer_handle_t timer;
} blink_led_t;

#define MAX_BLINK_LED 3

static blink_led_t leds[MAX_BLINK_LED];
static blink_led_t* get_led(int gpio)
{
    for (int i = 0; i < MAX_BLINK_LED; i++) {
        if (leds[i].gpio == gpio)
            return &leds[i];
    }

    for (int i = 0; i < MAX_BLINK_LED; i++) {
        if (leds[i].gpio == 0) {
            leds[i].gpio = gpio;
            return &leds[i];
        }
    }

    return NULL;
}

static void blink_cb(void *arg)
{
    blink_led_t *led = (blink_led_t*)arg;
    led->state = !led->state;
    gpio_set_level(led->gpio, led->state);
}

void gpioinit(void){
    gpio_config_t LED_Wifi_conf = {
    .pin_bit_mask = (1ULL << LED_Wifi),
    .mode = GPIO_MODE_OUTPUT,
    .pull_up_en = GPIO_PULLUP_DISABLE,
    .pull_down_en = GPIO_PULLDOWN_DISABLE,
    .intr_type = GPIO_INTR_DISABLE
};
gpio_config_t LED_Modbus_conf = {
    .pin_bit_mask = (1ULL << LED_Modbus),
    .mode = GPIO_MODE_OUTPUT,
    .pull_up_en = GPIO_PULLUP_DISABLE,
    .pull_down_en = GPIO_PULLDOWN_DISABLE,
    .intr_type = GPIO_INTR_DISABLE
};
gpio_config_t LED_MQTT_conf = {
    .pin_bit_mask = (1ULL << LED_MQTT),
    .mode = GPIO_MODE_OUTPUT,
    .pull_up_en = GPIO_PULLUP_DISABLE,
    .pull_down_en = GPIO_PULLDOWN_DISABLE,
    .intr_type = GPIO_INTR_DISABLE
};

    gpio_config(&LED_Wifi_conf);
    gpio_config(&LED_Modbus_conf);
    gpio_config(&LED_MQTT_conf);
}

void ledBlink(int gpio, int interval_ms)
{
    blink_led_t *led = get_led(gpio);
    if (!led) return;

    if (led->blinking) return;

    const esp_timer_create_args_t args = {
        .callback = blink_cb,
        .arg = led,
        .name = "blink"
    };

    esp_timer_create(&args, &led->timer);
    esp_timer_start_periodic(led->timer, interval_ms * 1000);

    led->blinking = 1;
}

void ledSet(int gpio, int level)
{
    blink_led_t *led = get_led(gpio);
    if (!led) return;

    if (led->blinking) {
        esp_timer_stop(led->timer);
        esp_timer_delete(led->timer);
        led->blinking = 0;
    }

    led->state = level;
    gpio_set_level(gpio, level);
}
```

```c
#ifndef GPIOCON_H
#define GPIOCON_H

//merah wifi
//kuning mqtt
//putih modbus

typedef enum{
LED_Wifi= 13,
LED_Modbus= 14,
LED_MQTT= 26
}led_pin_t;

void ledBlink(int gpio, int interval_ms);
void ledSet(int gpio, int level);
void gpioinit(void);

#endif
```

Lampiran 14. Program Modbus MQTT Gateway – modbus_handle.c dan modbus_handle.h

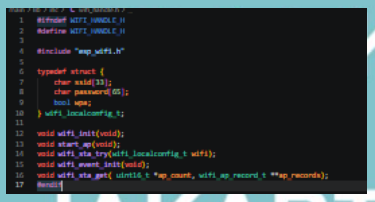


Lampiran 15. Program Modbus MQTT Gateway – mqtt_handle.c dan mqtt_handle.h



```c
#ifndef MQTT_HANDLE_H
#define MQTT_HANDLE_H

#include <stdint.h>

typedef struct {
    char broker_uri[128];
    char username[64];
    char password[128];
    uint8_t use_ssl;
    uint8_t use_cred;
} mqtt_config_t;

typedef struct {
    char pubtopicctrl[128];
    char subtopicctrl[128];
    char pubtopicrep[128];
    char subtopicrep[128];
} mqtt_topic_t;

void mqtt_start(mqtt_config_t *config);
void mqtt_stop(void);
void mqtt_publish(uint8_t group, char* value);
void mqtt_set_topic(const char* pubtopic, const char* subtopic, const char* groupreg, const char* groupctrl);
#endif
```

Lampiran 16. Program Modbus MQTT Gateway – wifi_handle.h



```c
#ifndef WIFI_HANDLE_H
#define WIFI_HANDLE_H

#include "esp_wifi.h"

typedef struct {
    char ssid[33];
    char password[65];
    bool wpa;
} WiFi_localconfig_t;

void wifi_init(void);
void start_ap(void);
void wifi_sta_try(wifi_localconfig_t wifi);
void wifi_event_init(void);
void wifi_sta_get( uint16_t *ap_count, wifi_ap_record_t **ap_records);
#endif
```

Lampiran 17. Program Modbus MQTT Gateway – webhandle.c dan webhandle.h