UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
CURSO DE GRADUAÇÃO EM ENGENHARIA
DE CONTROLE E AUTOMAÇÃO

FILIPE GOMES DE SOUZA

EXPLORANDO A APLICAÇÃO DA NORMA IEC 61499 NA AUTOMAÇÃO RESIDENCIAL: Um estudo de caso em sistemas de controle distribuídos.

Recife
2025

FILIPE GOMES DE SOUZA

EXPLORANDO A APLICAÇÃO DA NORMA IEC 61499 NA AUTOMAÇÃO RESIDENCIAL: Um estudo de caso em sistemas de controle distribuídos.

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Elétrica da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação.

Orientador(a): Prof. Dr. Herbert Albérico de Sá Leitão

Recife
2025

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Souza, Filipe Gomes de.
Explorando a aplicação da norma IEC 61499 na automação residencial: Um estudo de caso em sistemas de controle distribuídos. / Filipe Gomes de Souza. - Recife, 2025.
100 p : il.

Orientador(a): Herbert Albérico de Sá Leitão
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, Engenharia de Controle e Automação - Bacharelado, 2025.
Inclui referências, apêndices.

1. Automação residencial. 2. IEC 61499. 3. ESP32. 4. Forte 4Diac. 5. MQTT. I. Sá Leitão, Herbert Albérico de . (Orientação). II. Título.

620 CDD (22.ed.)

FILIPE GOMES DE SOUZA

EXPLORANDO A APLICAÇÃO DA NORMA IEC 61499 NA AUTOMAÇÃO RESIDENCIAL: Um estudo de caso em sistemas de controle distribuídos.

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Elétrica da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação.

Aprovado em: 14/04/2025

BANCA EXAMINADORA

_____

Prof. Dr. Herbert Albérico de Sá Leitão (Orientador)
Universidade Federal de Pernambuco

_____

Prof. Dr. Davidson da Costa Marques (Examinador Interno)
Universidade Federal de Pernambuco

_____

Eng. M. Sc. Renato Andrade Freitas (Examinador Interno)
Universidade Federal de Pernambuco

Dedico este trabalho àqueles que, como eu, compreendem o verdadeiro valor do esforço e têm uma apreciação genuína pelo que é inovador. Que este trabalho inspire a busca constante de novas fronteiras.

AGRADECIMENTOS

Gostaria de expressar minha gratidão a Deus pela realização deste trabalho. Agradeço aos meus pais, Sérgio e Janilsa, minha irmã Priscila e à minha esposa, Maria Eduarda, pelo apoio constante e incentivo em minha formação. O apoio de vocês foi fundamental para alcançar este objetivo.

Agradeço também aos meus amigos de curso, especialmente a Vitor Rodrigo, Luiz Augusto, Denis Tiburtino e Arthur Medalha, pelo companheirismo e pelos momentos compartilhados durante essa trajetória acadêmica. Também sou grato aos companheiros de trabalho, em especial a Igor, Davi e Vitor pelo apoio e colaboração ao longo desse período.

Por fim, agradeço ao meu orientador, o Prof. Dr. Herbert Albérico, pela orientação e paciência durante o desenvolvimento deste trabalho. Seus conselhos foram essenciais para o seu aprimoramento.

RESUMO

O desenvolvimento tecnológico tem sido um impulsionador para a evolução das residências inteligentes, impulsionando uma demanda projetada de crescimento consistente nos próximos anos. No entanto, a automação residencial enfrenta desafios com a falta de padronização que dificulta a interoperabilidade entre tecnologias proprietárias. Diante disso, a norma internacional IEC 61499 destaca-se por oferecer uma arquitetura que simplifica a integração entre dispositivos de diferentes fabricantes e promove uma estrutura modular e flexível, facilitando a padronização. Este estudo analisa como aplicar a norma IEC 61499 integrando o Eclipse Forte 4Diac ao SoC (*System on a chip*) ESP32-S3 da ESPRESSIF. O projeto visa analisar as potencialidades dessa integração por meio da implementação de blocos de interface de serviço para leitura e escrita em entradas e saídas digitais e analógicas, além de blocos de comunicação com o protocolo MQTT (*Message Queuing Telemetry Transport*). A pesquisa busca avaliar a viabilidade, eficiência e benefícios do uso da IEC 61499 em dispositivos microcontroladores de baixo custo voltados para aplicações em casas inteligentes e demonstrar a relevância da norma para atender demandas do setor como segurança e maior compatibilidade.

Palavras-chave: Automação residencial; IEC 61499; ESP32; Forte 4Diac, MQTT.

ABSTRACT

Technological development has been a driving force behind the evolution of smart homes, fueling a projected demand for consistent growth in the coming years. However, home automation faces challenges due to the lack of standardization, which hinders interoperability among proprietary technologies. In this context, the international standard IEC 61499 stands out by offering an architecture that simplifies the integration of devices from different manufacturers and promotes a modular and flexible structure, facilitating standardization. This study analyzes how to apply the IEC 61499 standard by integrating Eclipse Forte 4DIAC with the ESPRESSIF ESP32-S3 System on Chip (SoC). The project aims to explore the potential of this integration through the implementation of service interface function blocks for reading and writing digital and analog inputs and outputs, as well as communication blocks using the MQTT (Message Queuing Telemetry Transport) protocol. The research seeks to assess the feasibility, efficiency, and benefits of using IEC 61499 in low-cost microcontroller-based devices for smart home applications, and to demonstrate the relevance of the standard in addressing sector demands such as security and increased compatibility.

Keywords: Home Automation; IEC 61499; ESP32; Forte 4DIAC, MQTT.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

4Diac	*Framework for Distributed Industrial Automation & Control*
ADC	*Analog-Digital Converter*
API	*Application Programming Interface*
BFB	*Basic Function Block*
BLE	*Bluetooth Low Energy*
CFB	*Composite Function Block*
DAC	Digital-Analog Converter
DRAM	*Dynamic Random Access Memory*
ECC	*Execution Control Chart*
EFB	*Event Function Block*
ESP	*Espressif System on Chip*
ESP-IDF	Espressif IoT *Development Framework*
FAL	*Forte Abstraction Layer*
FB	*Function Blocks*
GPIO	*General Purpose Input Output*
I2C	*Inter-Integrated Circuit*
IANA	*Internet Assigned Numbers Authority*
IDE	*Integrated Development Environment*
IEC	*International Electrotechnical Commission*
IoT	*Internet of Things*
LED	*Light Emitting Diode*
LwIP	*Lightweight Internet Protocol*
LTS	*Long-term support*
MQTT	*Message Queuing Telemetry Transport*
OPC UA	*Open Platform Communications Unified Architecture*
PSRAM	*Pseudo-Static Random Access Memory*
PWM	*Pulse Width Modulation*
RAM	*Random Access Memory*
FreeRTOS	*Free Real-Time Operating System*
RTE	*Runtime Environment*
SDK	*Software Development Kit*
SIFB	*Service Interface Function Block*
SoC	*System on a chip*

SPI	*Serial Peripheral Interface*
SSL	*Secure Sockets Layer*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
TLS	*Transport Layer Security*
TRIAC	*Triode for Alternating Current*
UART	*Universal Asynchronous Receiver Transmitter*

SUMÁRIO

1 INTRODUÇÃO

A automação residencial, também conhecida como domótica, vem se tornando mais acessível devido a diversos fatores, como o aumento dos investimentos em inovação tecnológica, a redução nos custos de produção de dispositivos eletrônicos e a possibilidade de utilização de uma infraestrutura digital mais acessível, que agora inclui redes de comunicação mais eficientes e com conectividade otimizada.

De acordo com Muratori (2017), a automação residencial pode ser organizada em níveis de interação, que variam principalmente com a quantidade de intervenção humana exigida para sua operação, classificando a residência inteligente, como um modelo mais avançado de automação residencial, capaz de gerenciar subsistemas de forma centralizada e adaptativa, com base em dados e preferências dos usuários.

Ainda conforme Marikyan (2019), uma casa inteligente é caracterizada pelo uso de dispositivos e sensores integrados em um sistema que permite o gerenciamento, monitoramento e o fornecimento de serviços de resposta. A implementação de tal sistema proporciona múltiplas vantagens, entre elas mais segurança e proteção aos usuários, economia eficiente de energia, mais conforto e valorização do imóvel.

Pesquisas recentes mostram projeções expressivas para o crescimento da automação no cenário doméstico. A *Fortune Business Insights* (2025), consultoria global de análise de mercado, destaca em seu relatório que a América do Sul, especialmente o Brasil, deve apresentar uma taxa de crescimento anual composta de 20,2% no setor de casas inteligentes entre os anos de 2023 e 2030.

Ainda segundo a *Fortune Business Insights* (2025) observa-se uma tendência crescente de incorporação de tecnologias como a IoT (*Internet of Things*), inteligência artificial e aprendizado de máquina em dispositivos residenciais. Esse cenário reforça a necessidade de adoção de arquiteturas flexíveis e eficientes.

Esse avanço no setor também traz desafios significativos, especialmente no que se refere à ausência de padrões consistentes de programação, que dificulta a interoperabilidade entre tecnologias proprietárias e torna complexa a configuração e manutenção dos sistemas. Por isso, a adoção de uma arquitetura de programação padronizada e descentralizada mostra-se fundamental para aumentar a flexibilidade dos sistemas e trazer mais confiabilidade. A IEC 61499, inicialmente destacada na

automação industrial, mostra-se uma alternativa viável para solucionar esses desafios atuais da automação residencial.

De acordo com Damaso (2011), a IEC 61499 surgiu com características que permitem a introdução de uma nova geração de processos de controle distribuído, tornando-se ideal para o desenvolvimento de projetos em diversas áreas de controle e supervisão, incluindo o setor de automação residencial. Sendo assim, este trabalho aplica a IEC 61499 na automação residencial por meio da integração da tecnologia *open-source* 4Diac, que inclui o 4Diac-IDE e o *runtime* Forte, com o microcontrolador ESP32-S3 da ESPRESSIF.

A escolha do 4Diac deve-se à sua ampla adoção em sistemas de automação baseados no padrão IEC 61499. Por sua vez, o ESP32-S3 foi selecionado principalmente por oferecer suporte à plataforma nativa de desenvolvimento ESP-IDF, a qual proporciona um ambiente robusto para a compilação cruzada e execução eficiente do *runtime* Forte, utilizado pelo 4Diac. Outros fatores determinantes incluíram o bom custo-benefício, a arquitetura de 32 bits — requisito mínimo para aplicações com o 4Diac —, e conectividade Wi-Fi estável.

A integração será realizada por meio da compilação do código-fonte do Forte *runtime*, utilizando como base o FreeRTOS (*Free Real-Time Operating System*) e o LwIP (*Lightweight Internet Protocol*), com o objetivo de integrar o sistema ao ESP32-S3 como um componente do ESP-IDF (*Espressif IoT Development Framework*). A última etapa da integração será o desenvolvimento de blocos de serviços para comunicação com as entradas e saídas digitais e analógicas do ESP32-S3 e um bloco de comunicação com a rede, utilizando o protocolo MQTT (*Message Queuing Telemetry Transport*).

Por fim, será realizado um estudo de caso, no qual os blocos de serviço desenvolvidos ao longo do projeto, serão aplicados a um ambiente de testes doméstico, explorando a comunicação com o hardware e com a internet. Ele possibilitará validar a funcionalidade da integração entre a ferramenta Eclipse 4Diac, o microcontrolador e os dispositivos residenciais, demonstrando a viabilidade da aplicação da norma IEC 61499 em automação residencial.

1.1 Objetivos

1.1.1 Geral

Desenvolver uma aplicação da norma IEC 61499 para sistemas de automação residencial, integrando a ferramenta Eclipse 4Diac com o microcontrolador ESP32-S3 e validando a viabilidade, eficiência e benefícios do uso de uma arquitetura padronizada e distribuída na automação de casas inteligentes.

1.1.2 Específicos

- Desenvolver blocos de serviço baseados na norma IEC 61499 para a comunicação com entradas e saídas digitais e analógicas do ESP32-S3.
- Desenvolver blocos de serviço de comunicação com o protocolo MQTT baseado na norma IEC 61499 para conectar dispositivos à internet e permitir o gerenciamento remoto do sistema.
- Desenvolver e executar um estudo de caso em um ambiente doméstico para testar a integração entre os blocos de serviço, o microcontrolador ESP32-S3 e a ferramenta Eclipse 4Diac.

1.2 Organização do Trabalho

O capítulo 2 apresenta a fundamentação teórica que engloba o desenvolvimento deste trabalho, abordando a norma IEC 61499, o ESP32-S3, o protocolo de comunicação MQTT e a ferramenta Eclipse 4Diac, destacando os conceitos principais.

No capítulo 3, são descritas as etapas de desenvolvimento do projeto, incluindo a preparação do ambiente de desenvolvimento, a compilação do runtime 4Diac Forte no ESP32-S3 e a criação dos blocos de serviço para controle e comunicação do sistema.

O capítulo 4 apresenta o estudo de caso realizado para validar a integração entre o ESP32-S3, o Eclipse 4Diac e os dispositivos de automação residencial, demonstrando os resultados obtidos.

Por fim, o capítulo 5 traz as considerações finais do trabalho, comparando os resultados alcançados com as expectativas iniciais e apontando possíveis melhorias e direções para trabalhos futuros.

2 METODOLOGIA

Este trabalho caracteriza-se como uma pesquisa aplicada de natureza experimental, voltada ao desenvolvimento e validação de uma solução prática de automação residencial baseada na norma IEC 61499. A metodologia adotada foi organizada em etapas sequenciais, com o objetivo de garantir a integração adequada entre a ferramenta Eclipse 4Diac, o microcontrolador ESP32-S3 e os dispositivos físicos do ambiente de teste. A seguir, são descritas as etapas que compõem o processo de desenvolvimento.

2.1 Preparação do Ambiente de Desenvolvimento

A primeira etapa consistiu na preparação do ambiente de desenvolvimento para compilação e execução do Forte *runtime* da ferramenta Eclipse 4Diac. Para isso, utilizou-se o ESP-IDF, que oferece suporte ao sistema operacional FreeRTOS e à pilha de rede LwIP. Esses elementos garantem a compatibilidade do ESP32-S3 com o Forte, possibilitando o desenvolvimento das aplicações desejadas.

2.2 Compilação e Execução do Forte Runtime no ESP32-S3

Após a configuração do ambiente, procedeu-se à compilação cruzada do código-fonte do Forte runtime e posteriormente sua integração ao ESP32-S3 como um componente do ESP-IDF. Essa etapa exigiu a adaptação de configurações específicas do ESP32-S3, como uso e configuração do FreeRTOS, alocação de

memória, conectividade Wi-Fi e suporte a periféricos, de modo a assegurar a correta execução do runtime no microcontrolador.

2.3 Desenvolvimento dos Blocos de Serviço

Em seguida, foram implementados blocos de serviço conforme a IEC 61499 para a manipulação de entradas e saídas digitais e analógicas do ESP32-S3. Também foi desenvolvido bloco de comunicação utilizando o protocolo MQTT, com o objetivo de viabilizar a troca de dados entre os dispositivos físicos e sistemas externos, como servidores na nuvem ou aplicativos de controle remoto. Todos os blocos foram configurados dentro do Eclipse 4Diac IDE.

2.4 Montagem do Ambiente de Testes

Com os blocos funcionais prontos, foi montado um ambiente de testes simulando situações comuns da automação residencial, como acionamento de relés, sensores de presença, LEDs e demais dispositivos de entrada/saída. O sistema foi conectado à internet por meio da interface Wi-Fi do ESP32-S3 e vinculado a um *broker* MQTT para validação das funcionalidades em tempo real.

2.5 Execução do Estudo de Caso e Validação

Por fim, foi executado um estudo de caso prático, no qual a integração entre a ferramenta Eclipse 4Diac, o ESP32-S3 e os dispositivos do ambiente foram testadas de forma contínua. A validação considerou aspectos como o desempenho dos blocos de serviço, o tempo de resposta do sistema, a estabilidade da comunicação MQTT e a facilidade de configuração da aplicação. Os resultados permitiram avaliar a viabilidade técnica da norma IEC 61499 em cenários de automação residencial, destacando suas vantagens frente a soluções baseadas exclusivamente em dispositivos proprietários.

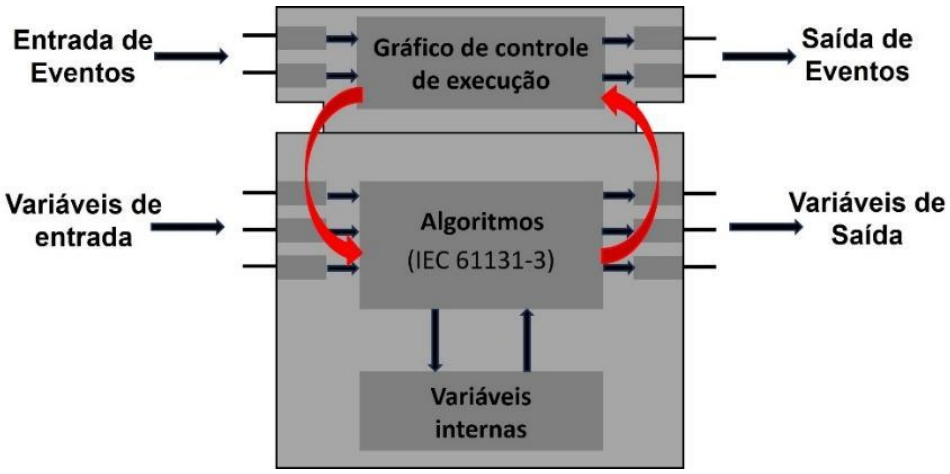
3 FUNDAMENTAÇÃO TEÓRICA

3.1 Norma IEC 61499

De acordo com Vyatkin (2011), a IEC 61499 foi desenvolvida para viabilizar a automação distribuída e orientada a eventos em sistemas industriais permitindo o desenvolvimento de sistemas industriais com controle descentralizado implementado por meio de componentes modulares de software. Ela define um modelo de arquitetura baseado no conceito de *Function Blocks* (FBs), que, conforme indicado por Dai e Vyatkin (2010), é a unidade estrutural básica para implementações da lógica descentralizada.

Entre as principais características distintivas dos FBs, conforme definido pela IEC 61499, estão a presença de eventos de entrada/saída desacoplados das variáveis de dados, variáveis internas e a descrição do comportamento interno por meio de algoritmos, os quais podem ser programados nas linguagens definidas pela IEC 61131-3 ou em qualquer outra linguagem de alto nível, como, por exemplo, C++. A representação gráfica de um bloco de função é mostrada na Figura 1.

Figura 1: Representação gráfica de um bloco de função.



Fonte: adaptado de Damaso (2011).

3.1.1 Tipos de Function Blocks

A norma IEC 61499 define quatro tipos principais de FBs, descritos a seguir (IEC 61499, 2012) (LEITÃO, 2022) (DAMASO, 2011):

- *Basic Function Block* (BFB): Este é o bloco de função mais simples. Sua principal característica é a existência de algoritmos internos, variáveis internas e um *Execution Control Chart* (ECC) obrigatório, que atua como uma máquina de estados, definindo o comportamento do BFB.
- *Composite Function Block* (CFB): Um CFB é uma instância de vários FBs. O encapsulamento é o principal benefício do CFB, que simplifica o design da aplicação substituindo uma rede de FBs por um único bloco.
- *Event Function Block* (EFB): Este tipo de FB é projetado para manipular eventos, seja para gerar um evento inicial ou para disparar simultaneamente várias cadeias de FBs.
- *Service Interface Function Block* (SIFB): O Bloco SIFB facilita a comunicação com o ambiente externo, permitindo a leitura de sensores, a escrita em atuadores e a troca de dados entre dispositivos remotos por redes industriais.

3.1.2 *Service Interface Function Blocks* (SIFBs)

De acordo com Leitão (2022), os SIFBs são blocos destinados à comunicação, tanto entre os periféricos dos controladores quanto à troca de mensagens por meio de protocolos em redes.

Os SIFBs possuem uma estrutura flexível e que não tem o ECC como definidor de comportamento. Essa característica é fundamental para o gerenciamento de mensagens recebidas por meio de uma rede wireless, por exemplo, que pode gerar eventos de saída a qualquer momento.

Damaso (2011) destaca que a distinção entre SIFB e os BFBs está, justamente, no grau de flexibilidade exigido. Um BFB tradicional possui uma estrutura mais rígida e controlada pelo ECC.

Por fim, é importante citar que há SIFBs de solicitação e SIFBs de resposta. Leitão (2022), evidencia que os SIFBs de solicitação são ativados pela aplicação, necessitando que um de seus eventos de entrada seja acionado para iniciar o processamento. Por outro lado, os blocos de resposta são acionados por recursos ou hardwares externos, gerando eventos de saída sempre que uma nova informação é recebida.

Este conceito pode ser demonstrado por meio de SIFBs utilizados no acesso a dispositivos físicos, como sensores. De acordo com Mendes e Parcianello (2022), a norma especifica o SIFB ***IO_Write*** que é utilizado para definir valores em atuadores, diretamente nas saídas físicas do controlador. Este é um bloco de solicitação e é ativado por eventos originados dentro da aplicação, realizando a escrita em um endereço físico definido.

Já o SIFB ***IO_Read*** é responsável pela leitura de valores provenientes de sensores físicos, podendo operar continuamente ou mediante solicitação, assumindo duas modalidades: ora como um bloco de resposta, quando configurado para reagir automaticamente a alterações detectadas na entrada física; ora como um bloco de solicitação, quando configurado para realizar leituras sob demanda, acionado por um evento específico da aplicação.

A norma especifica ainda os SIFBs ***Publish, Subscribe, Client*** e ***Server*** para comunicações. O ***Publish*** e o ***Subscribe*** são utilizados para comunicações não orientadas à conexão, como é o caso do protocolo MQTT. O bloco ***Publish***, é um bloco de solicitação, publica dados em uma rede para endereços e portas configurados, mas apenas quando um evento da aplicação o aciona. Já o bloco ***Subscribe***, classificado como bloco de resposta, monitora continuamente a rede para detectar mensagens enviadas por dispositivos externos.

Já os blocos ***Client*** e ***Server*** são desenvolvidos para suportar interações baseadas em conexões seguindo o modelo Cliente-Servidor. O ***Client*** é responsável por iniciar a comunicação ao enviar solicitações para um servidor remoto e aguardar uma resposta; já o ***Server*** irá processar esses pedidos e retornar as informações requeridas pelo ***Client***. Esses princípios são aplicados em protocolos como Modbus/TCP e OPC UA.

3.2 Protocolo MQTT (*Message Queuing Telemetry Transport*)

De acordo com Yassein et al. (2017), o MQTT é um protocolo de transmissão de mensagens que tem mecanismos para comunicação assíncrona entre dispositivos com memória e capacidade de processamento limitadas.

Ainda segundo Hillar (2017), o protocolo foi especificamente projetado para ser leve e funcionar de forma eficaz em redes instáveis e quando a conectividade é intermitente. É ideal para uma situação na qual diferentes dispositivos precisam se comunicar quase em tempo real pela internet, e quando o objetivo é reduzir o consumo de largura de banda.

O protocolo MQTT utiliza portas TCP/IP padronizadas. A porta 1883 é reservada para conexões sem criptografia, comumente utilizada em cenários onde a segurança é organizada por métodos alternativos, como em redes privadas. A IANA *(Internet Assigned Numbers Authority)* atribuiu o uso exclusivo dessa porta ao MQTT, evitando conflitos com outros protocolos.

Embora a porta 1883 seja a mais comum, existem outras portas padronizadas para o MQTT, como por exemplo a porta 8883, utilizada para conexões com criptografia TLS/SSL. Assim como a porta 1883, ela também foi registrada pela IANA para uso exclusivo do MQTT. Yassein et al. (2017), enfatizam que o uso dessas portas padronizadas simplifica a configuração do protocolo e assegura uma comunicação eficiente e segura, especialmente em sistemas de IoT.

3.2.1 Padrão Publish-Subscribe

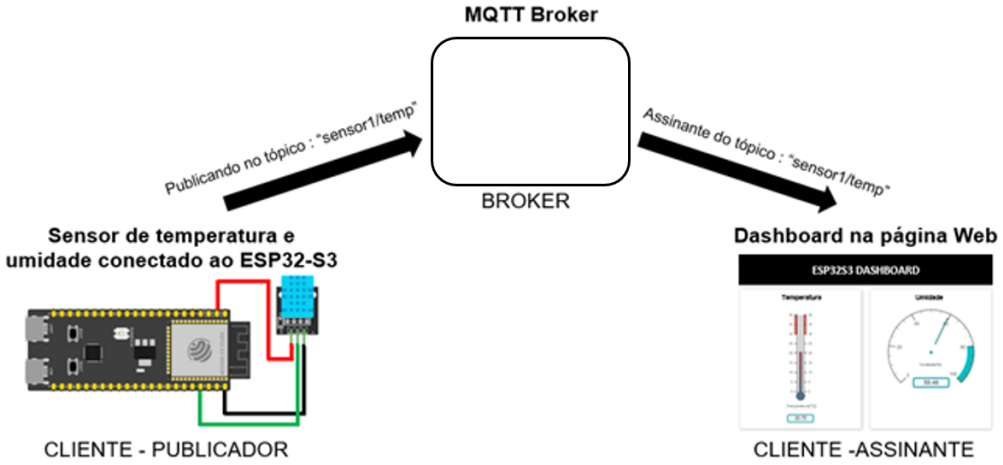
Ao utilizar o MQTT, é necessário compreender o padrão *publish-subscribe*, também conhecido como modelo pub-sub. De acordo com Hillar (2017), esse modelo permite que os clientes compartilhem mensagens de maneira independente. O cliente que envia a mensagem, conhecido como publicador, não precisa conhecer ou se conectar diretamente aos clientes que a recebem, conhecidos como assinantes.

O modelo pub-sub faz uso de um *broker* que intermedia as mensagens recebidas pelos publicadores e as distribui aos interessados. Segundo Eugster et al. (2003), essa abordagem é descrita como filtragem por tópicos. Nesse mecanismo, as

mensagens são publicadas associadas a um tópico, e os assinantes expressam interesse em tópicos específicos, recebendo apenas as mensagens associadas a esses tópicos.

A Figura 2 ilustra o modelo *publish-subscribe*, análogo ao que será posteriormente aplicado neste trabalho.

Figura 2: Exemplo do modelo *Publish-Subscribe* no protocolo MQTT.



Fonte: Desenvolvida pelo autor (2025).

3.3 Eclipse 4Diac (*Framework for Distributed Industrial Automation and Control*)

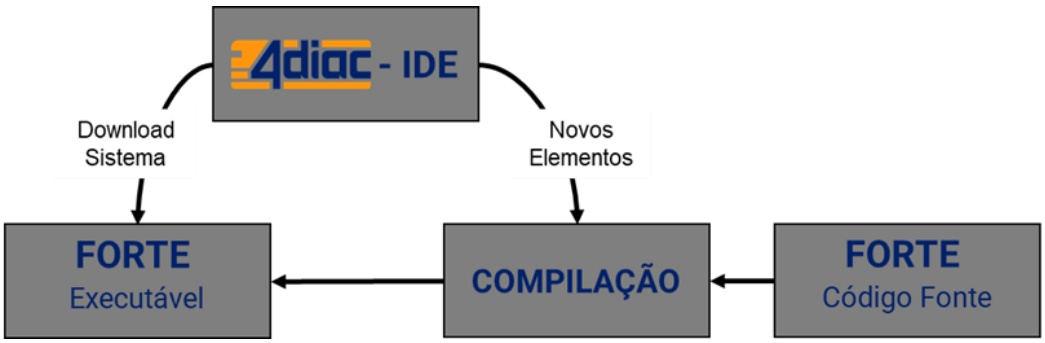
O Eclipse 4Diac é uma plataforma de código aberto desenvolvida para a modelagem, simulação e execução de processos industriais automatizados baseados na norma IEC 61499. Conforme Damaso (2011) descreve, a plataforma possui dois componentes principais: o 4Diac-IDE, utilizado para a modelagem e configuração dos sistemas, e o Forte, que atua como um RTE (*Runtime Environment*) para as aplicações desenvolvidas no IDE.

Na prática, o 4Diac-IDE permite a criação de novos FBs e de aplicações que representam o sistema de automação e controle. Esses FBs podem ser exportados em formato de código C++ para compor o sistema final. Nesse contexto, o Forte, que é o ambiente responsável pela execução das aplicações desenvolvidas, precisa ser compilado com os novos FBs personalizados que foram exportados pelo 4Diac-IDE.

Além dos arquivos exportados, Damaso (2011) explica que a compilação deve conter outros dois elementos principais: o código-fonte básico do Forte e as configurações da plataforma de destino.

A Figura 3 apresenta uma visão detalhada do funcionamento do *Framework* Eclipse-4Diac, destacando a interação entre seus principais componentes.

Figura 3: Diagrama descritivo de funcionamento do Eclipe-4DIAC.

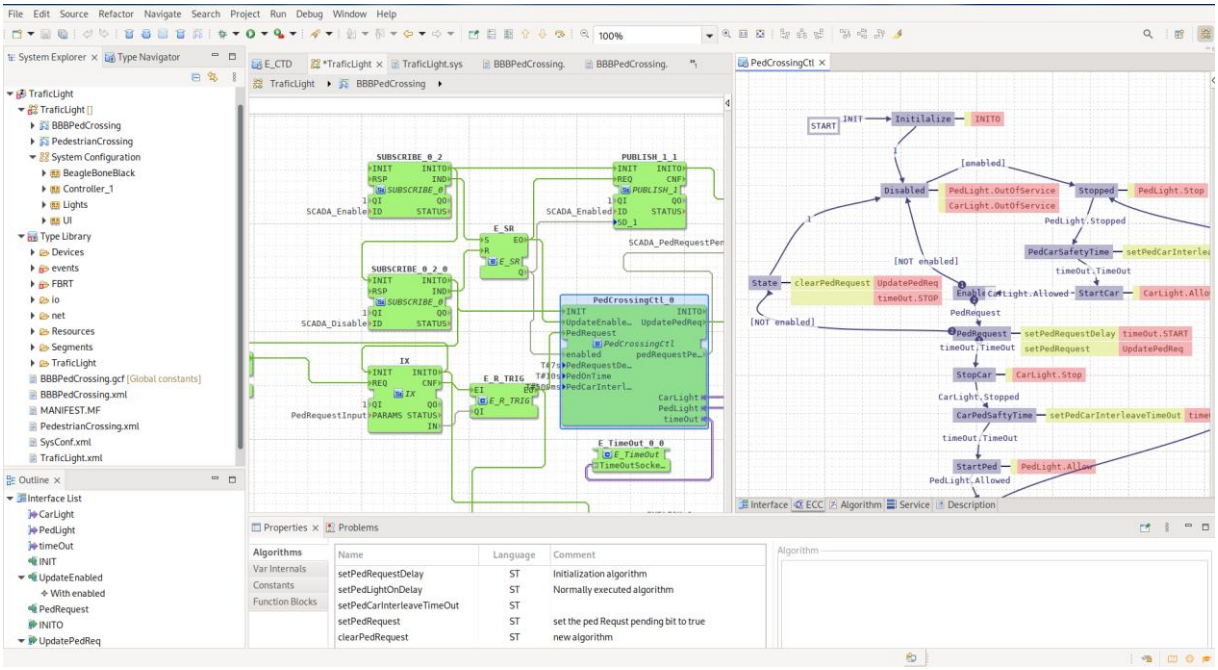


Fonte: Adaptado de Damaso (2011).

3.3.1 4Diac-IDE

O 4Diac-IDE é o ambiente de desenvolvimento integrado do projeto Eclipse 4Diac, projetado para criar, editar, importar e exportar elementos e sistemas baseados na norma IEC 61499. De acordo com Strasser et al. (2010), o 4Diac-IDE foi desenvolvido sobre o *Eclipse Framework*, aproveitando a ampla gama de *plugins* disponíveis na plataforma. Entre seus principais recursos destacam-se o gerenciamento de projetos, editores especializados, bibliotecas integradas, e funcionalidades avançadas para download e implementação de sistemas distribuídos.

Figura 4: Demonstração do modo de edição de aplicações do 4Diac-IDE.



Fonte: Retirado de ECLIPSE (2024).

O 4Diac-IDE pode ser obtido gratuitamente por meio da página oficial mantida pela Eclipse Foundation, responsável pelo desenvolvimento e manutenção da ferramenta. A versão mais recente do ambiente pode ser acessada em: [https://eclipse.dev/4diac/download/]. Essa página centraliza as atualizações mais atuais e as melhorias contínuas do projeto, garantindo uma experiência otimizada para os desenvolvedores.

O sistema já oferece bibliotecas pré-existentes compatíveis com o código-fonte básico do Forte, cobrindo operações fundamentais para sistemas de automação. A seguir, detalham-se as bibliotecas que são nativamente compatíveis com o código-fonte básico:

- *Convert*: Esta biblioteca contém blocos destinados à conversão de tipos de dados.
- IEC61131-3: Inclui blocos que implementam as funções definidas pela norma IEC61131-3, permitindo a utilização de funções como *timers*, contadores e operações matemáticas básicas.
- *Net*: Fornece blocos para comunicação em rede, implementando modelos Publicador-Subscritor e Cliente-Servidor.

- *Events e Rtevents*: A biblioteca *events* oferece blocos para realizar operações como unir ou dividir eventos, gerar eventos periódicos, etc. A biblioteca *rtevents* é similar, mas foi desenvolvida para sistemas de tempo real.
- *Reconfiguration*: Oferece blocos para reconfiguração dinâmica de sistemas, permitindo a alteração de comportamentos em tempo de execução sem a necessidade de reinicialização.

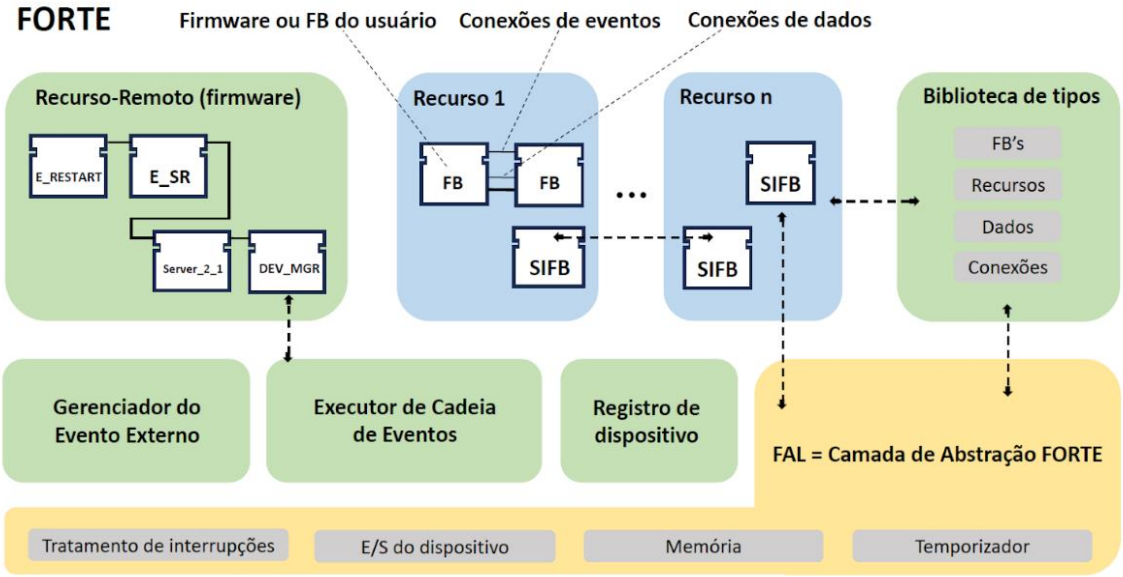
No 4Diac-IDE, há também suporte para a criação de novos tipos de FBs. Conforme destacado por Damaso (2011), esse processo oferece total controle sobre o FB, permitindo a definição de eventos, dados de entrada e saída, bem como a associação entre esses elementos. Após a criação, todos os FBs devem ser exportados e compilados no Forte, para garantir sua execução correta nos sistemas que os utilizam.

3.3.2 Runtime Forte

O Forte é uma implementação portátil em C++ do ambiente de execução da IEC 61499, desenvolvida para dispositivos de controle embarcados de pequeno porte com processadores de 16 ou 32 bits. É possível operar o Forte tanto em computadores com arquitetura x86 ou ARM quanto em sistemas embarcados como o FreeRTOS.

A Figura 5 detalha a arquitetura do ambiente de execução do Forte.

Figura 5: Arquitetura do *runtime* Forte.



Fonte: Adaptado de Damaso (2011).

O 4Diac Forte é projetado com uma arquitetura modular e o elemento centralizador que possibilita essa abordagem é a camada de abstração FAL (Forte *Abstraction Layer*), De acordo com Damaso (2011), a FAL funciona como um intermediário entre o software do Forte e o sistema operacional subjacente. Essa camada gerencia operações de baixo nível, como comunicação, *threads*, memória e interrupções, abstraindo essas funções para tornar as aplicações independentes do hardware ou sistema operacional.

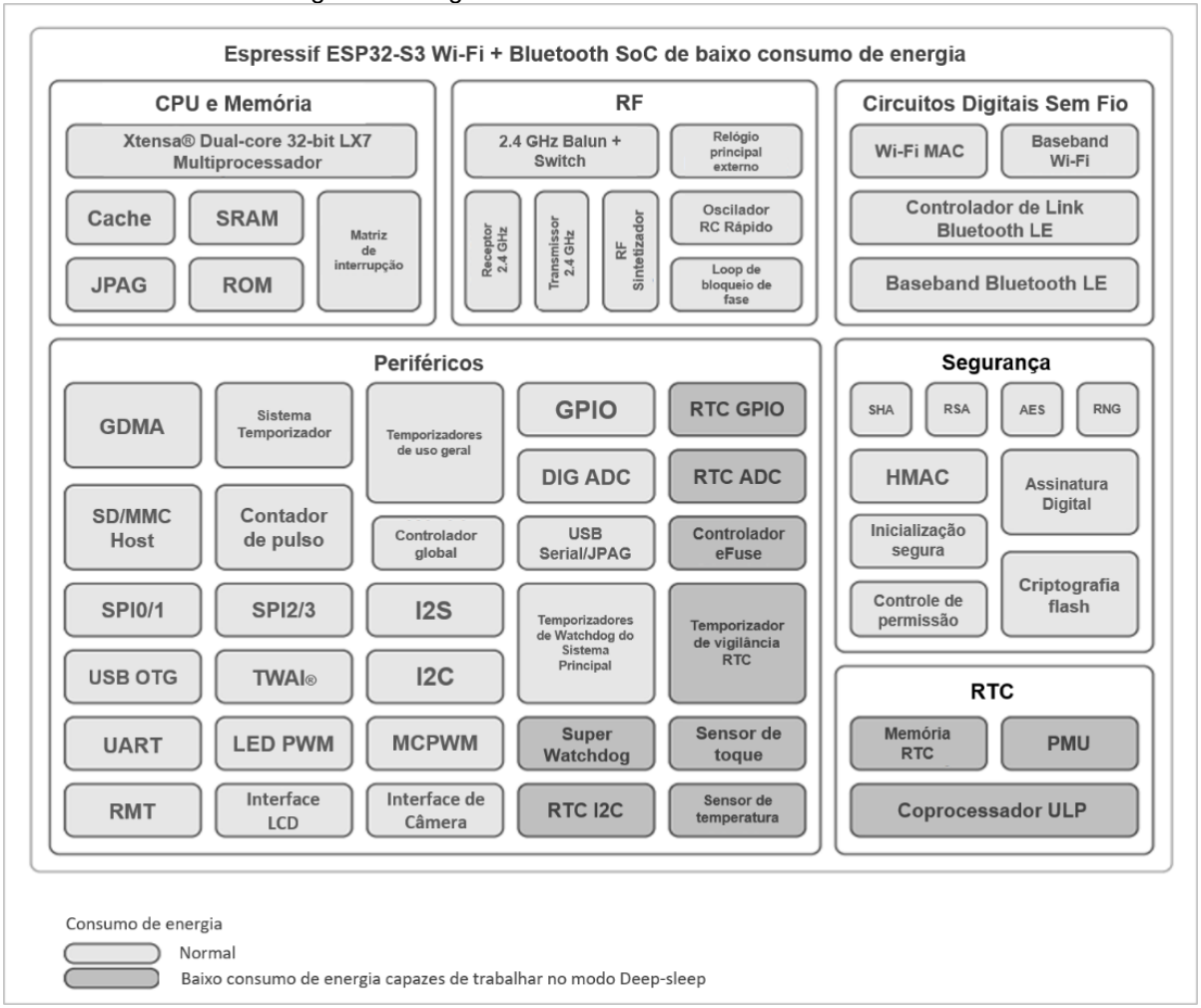
3.4 Microcontrolador ESP32-S3

A ESPRESSIF (2024), descreve o ESP32-S3 como um sistema em um *chip* (SoC) de baixo custo. Ele possui um microprocessador *dual-core* Xtensa LX7 de 32 bits que opera a até 240 MHz e um coprocessador para ultra baixo consumo. Além disso, o SoC inclui módulos de *baseband* para Wi-Fi e BLE *(Bluetooth Low Energy*) e possui 45 pinos GPIO (*General Purpose Input Output*) configuráveis, compatíveis com interfaces como SPI (*Serial Peripheral Interface)*, para comunicação rápida; I2C (Inter-Integrated Circuit), para conectar múltiplos dispositivos; UART *(Universal Asynchronous Receiver Transmitter)*, para comunicação serial; ADC (*Analog-to-*

Digital Converter), para leitura de sinais analógicos; e PWM (*Pulse Width Modulation*), para controle de LEDs e motores, entre outros.

A Figura 6 mostra o diagrama de blocos funcionais do SoC.

Figura 6: Diagrama de blocos funcional do ESP32-S



Fonte: Adaptado de ESPRESSIF (2024).

O ESP32-S3 está disponível em diferentes versões que variam de acordo com requisitos específicos de projetos como a memória *Flash* incorporada, PSRAM (*Pseudo Static Random-Access Memory*), temperatura de operação e tensão do barramento SPI.

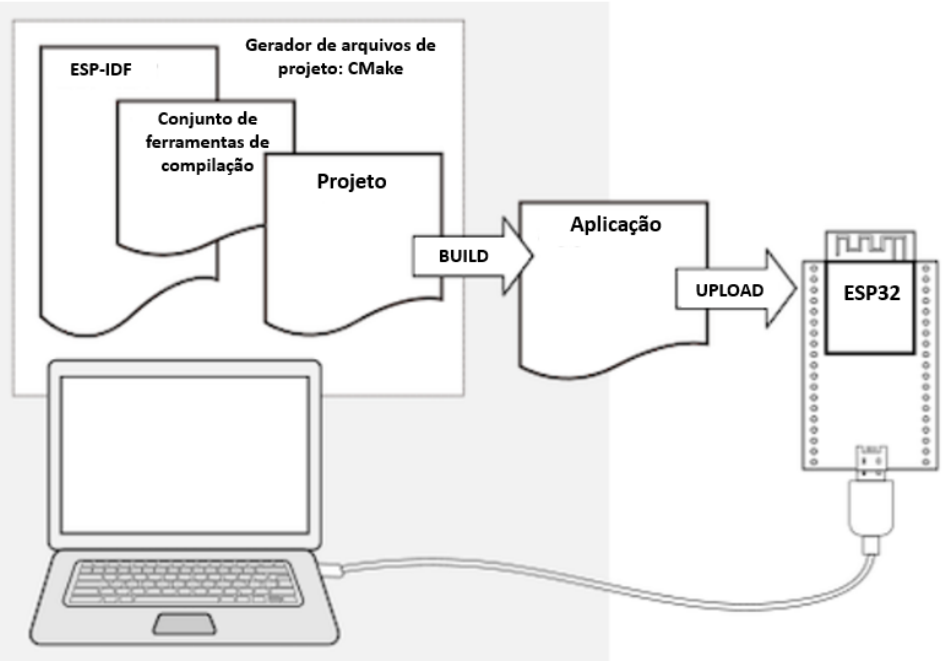
3.4.1 ESP-IDF (*Espressif IoT Development Framework)*

De acordo com EXPRESSIF (2024), o ESP-IDF é o seu *Framework* oficial de desenvolvimento para IoT, projetado para as séries de SoCs ESP32, ESP32-S, ESP32-C e ESP32-H. Que fornece um SDK (*Software Development Kit*) completo para o desenvolvimento de aplicações, utilizando linguagens de programação como C e C++. O SDK abrange a configuração do projeto, monitoramento e depuração, uma API (*Application Programming Interface)* simplificada para acessar as funcionalidades do hardware, e a integração com o sistema operacional de tempo real *FreeRTOS*.

O desenvolvimento com o ESP-IDF no ESP32 requer a configuração de três componentes essenciais. O primeiro é o *toolchain*, um conjunto de ferramentas acompanhado por scripts de operação, utilizado para compilar o código para o ESP32. Em seguida, estão as ferramentas de *build*, como o *CMake* e o *Ninja*, responsáveis pela construção completa da aplicação. Por fim, o próprio ESP-IDF, que fornece as bibliotecas de software e o código-fonte específicos para o projeto, garantindo o suporte necessário para o desenvolvimento na plataforma.

A Figura 7 mostra a estrutura de desenvolvimento do ESP-IDF.

Figura 7: Estrutura de desenvolvimento do ESP-IDF.



Fonte: Adaptado de ESPRESSIF (2024).

Ainda de acordo com a EXPRESSIF (2024), um projeto ESP-IDF pode ser considerado uma combinação de vários componentes. Os componentes podem incluir bibliotecas base do ESP-IDF, drivers de Wi-Fi, pilhas TCP/IP, sistema operacional *FreeRTOS*, servidores web e o código principal que integra todos esses elementos. O projeto pode ser configurado através de um menu baseado em texto, onde é possível ajustar cada componente.

O funcionamento do SDK do ESP-IDF pode ser resumido através de alguns conceitos chave fornecidos pela ESPRESSIF (2024). O projeto é um diretório que contém todos os arquivos e configurações necessárias para gerar um único aplicativo, incluindo tabela de partições, sistema de arquivos e *bootloader*. A configuração do projeto é armazenada no arquivo *sdkconfig* e pode ser modificada utilizando o comando "*idf.py menuconfig*" para personalizar o projeto.

O aplicativo gerado pelo ESP-IDF é um executável que normalmente consiste em dois programas: o aplicativo principal (*firmware* personalizado) e o *bootloader*, responsável pela inicialização do aplicativo. Os componentes são blocos modulares de código independente, compilados como bibliotecas estáticas e integrados ao aplicativo, sendo fornecidos tanto pela ESPRESSIF quanto por fontes externas. Por fim, o *target* refere-se ao hardware para o qual a aplicação é compilada.

3.4.2 Pulse Width Modulation (PWM) e Analog-Digital Conversor (ADC)

Sistemas embarcados necessitam frequentemente interagir com o ambiente externo, realizando controle e leitura de sinais elétricos, o ESP32 possui dois módulos importantes responsáveis por realizar essa interação: O PWM e o ADC.

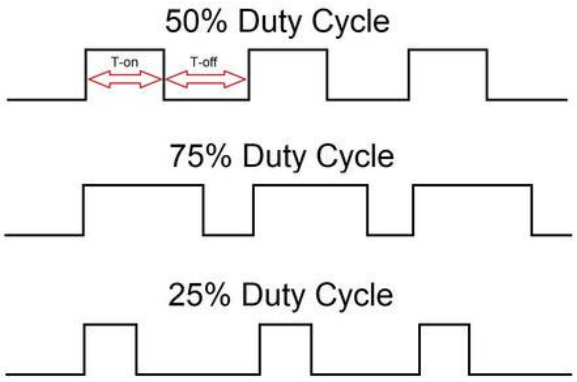
3.4.2.1 Pulse Width Modulation (PWM)

O PWM é uma técnica que controla a entrega de energia ou a transmissão de informações através da variação do ciclo de trabalho, também conhecido como *duty cycle*. Em outros termos, o *duty cycle* é a proporção do tempo em que um sinal está ativo em relação ao tempo total de um ciclo e pode variar de 0 a 100%. Essa variação possibilita que o fornecimento de energia a um dispositivo seja ajustado precisamente.

Segundo Silva (2019), o PWM gera um sinal de frequência e amplitude constantes, enquanto a largura do pulso é ajustada para regular a energia fornecida. Assim, o PWM se torna uma ferramenta valiosa em aplicações como controle de motores e iluminação.

A Figura 8 demostra exemplos de variação do *duty cycle*.

Figura 8: Variação do duty cycle.



Fonte: Adaptado de FIRGELLI (2025).

3.4.2.2 Analog-Digital Conversor (ADC)

O ADC realiza a representação digital discreta de um sinal analógico contínuo. Esse processo é necessário para que dispositivos digitais possam interpretar informações provenientes de grandezas físicas, como temperatura, luminosidade e variações de tensão.

De acordo com Haykin (2001), a quantização de um sinal analógico envolve a conversão do valor contínuo para uma representação discreta, e essa discretização depende diretamente da resolução do conversor, o que impacta na precisão do sinal digitalizado. Por exemplo, um ADC de 10 bits pode representar um sinal analógico em 1024 níveis discretos possíveis, enquanto um conversor de 12 bits proporciona uma precisão maior, oferecendo 4096 níveis possíveis.

No ESP32, os conversores ADC integrados possibilitam a leitura de sinais de sensores de temperatura, potenciômetros e fotocélulas.

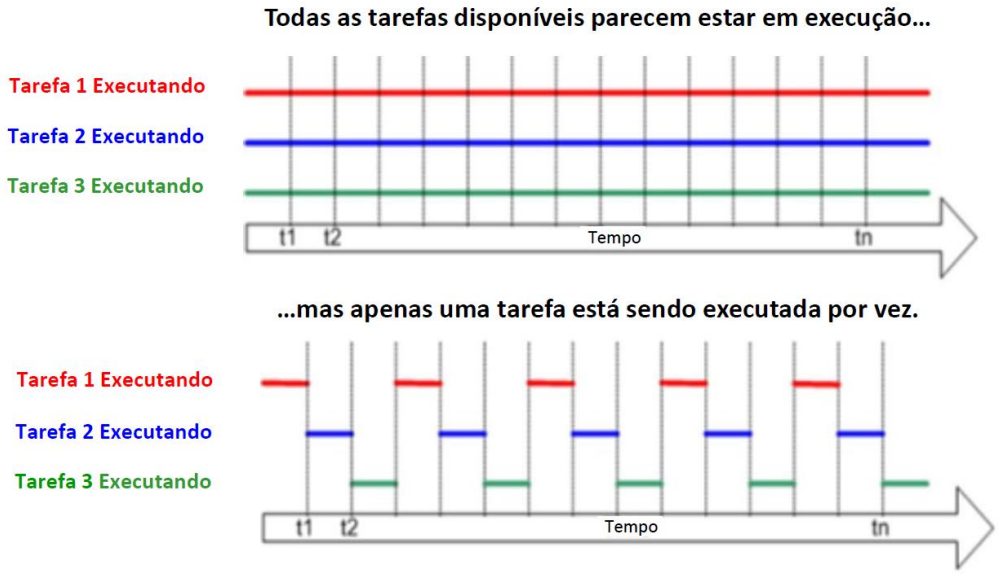
3.5 FreeRTOS

O *FreeRTOS* é um sistema operacional em tempo real que, de acordo com Barry (2010), fornece uma estrutura para sistemas embarcados que necessitam da gestão de várias tarefas simultâneas. No *FreeRTOS*, o gerenciamento das tarefas é realizado por um escalonador, que executa cada tarefa de acordo com as prioridades definidas.

Ainda conforme Barry (2010), o escalonador determina qual tarefa deve ser executada em cada ciclo, considerando duas políticas principais, a preemptiva e a cooperativa. Na política preemptiva, a tarefa de maior prioridade é sempre executada, interrompendo as tarefas de menor prioridade. Já na política cooperativa, as tarefas em execução só são interrompidas ao final de seu ciclo ou quando se encerram voluntariamente.

A documentação oficial do *FreeRTOS* (2025) detalha o gerenciamento multitarefa em contraste com o processamento convencional. Enquanto um processador com núcleo único executa apenas uma tarefa por vez, um sistema operacional multitarefa pode fazer com que várias tarefas pareçam ser executadas simultaneamente. Esse funcionamento está ilustrado na Figura 9, que apresenta o padrão de execução percebido (imagem a) e a execução real (imagem b) de três tarefas em relação ao tempo em um gerenciamento multitarefa.

Figura 9: Padrões de Execução no *FreeRTOS*: Concorrência Percebida vs. Multitarefa Real.



Fonte: Adaptado de FreeRTOS (2024).

Junto ao gerenciamento de tarefas, o *FreeRTOS* disponibiliza outros mecanismos como as filas (*Queues*) para o intercâmbio seguro de informações entre tarefas, os semáforos para sincronização entre tarefas, os *mutexes* para gerenciar acesso exclusivo a recursos compartilhados e os *timers* para executar ações programadas em intervalos regulares.

O *FreeRTOS* é muito utilizado no contexto da automação residencial possibilitando o controle de dispositivos inteligentes, como termostatos, lâmpadas e sistemas de segurança em controladores de processamento mais simples. De acordo com Barry (2010), a simplicidade deste sistema operacional e seu suporte a diversas arquiteturas de microcontroladores, são fatores que contribuem para sua popularidade.

3.6 LwIP

O LwIP (*Lightweight* Internet Protocol) é uma pilha de protocolos TCP/IP projetada para sistemas embarcados. Segundo Dunkels et al. (2004), o LwIP foi desenvolvido para minimizar o consumo de recursos, mantendo a conformidade com os principais protocolos da suíte TCP/IP. Essa eficiência permite que dispositivos com recursos limitados integrem funcionalidades de rede.

De acordo com a documentação disponibilizada pela *Eclipse Foundation* (2024), inicialmente, o *FreeRTOS* não possuía suporte nativo ao protocolo TCP/IP, o que levou à prática comum de integrá-lo ao LwIP. Embora o suporte ao TCP/IP tenha sido adicionado ao *FreeRTOS* posteriormente, muitas plataformas ainda utilizam o LwIP para sua compilação, como ocorre no caso do 4Diac Forte.

4 DESENVOLVIMENTO DO TRABALHO

Nesta seção, será exposto o desenvolvimento do trabalho, com a descrição de cada uma das fases realizadas, começando pela configuração do ambiente de trabalho até a aplicação e verificação das funcionalidades. O objetivo é demonstrar as ferramentas de hardware e software empregadas, além de relatar quais os obstáculos encontrados no processo e como foram resolvidos.

O desenvolvimento foi dividido em quatro etapas. A primeira etapa consistiu na configuração do ambiente de trabalho. Na segunda etapa, realizou-se a compilação do 4Diac Forte para execução no ESP32. A terceira etapa envolveu a implementação dos blocos de interface de serviço, projetados para utilizar a camada de abstração do Forte na comunicação com os GPIOs e na conexão com a rede via protocolo MQTT e compilação dos blocos criados para o ESP32. Por fim, a quarta etapa foi a realização de um estudo de caso, aplicando o projeto em um ambiente doméstico real.

4.1 Configuração do ambiente de trabalho

Inicialmente, foi utilizada uma ferramenta de virtualização para criar e gerenciar o sistema operacional destinado à implementação do projeto. O uso de uma máquina virtual nesse contexto visou facilitar a portabilidade do sistema e o gerenciamento de backups. A ferramenta escolhida foi o *VMWare*, na qual foi criada uma máquina virtual executando o sistema operacional Linux Ubuntu, versão 22.04.4 LTS. Uma distribuição Linux foi definida para o desenvolvimento do projeto devido à compatibilidade facilitada com ferramentas essenciais, como o *Cmake*, o interpretador *Python*, os compiladores de C/C++, o ESP-IDF e o suporte a bibliotecas e dependências comuns.

Após a Instalação do sistema operacional, foi feita a instalação dos softwares necessários para o projeto:

- *Python* 3.10.12: O ESP-IDF depende do *Python* para tarefas como a configuração do ambiente e a comunicação com o microcontrolador. Antes de instalar o ESP-IDF, é necessário ter o *Python* 3 instalado.

- *VScode IDE* 1.93.1: Para realizar a configuração e o desenvolvimento do código, é necessário utilizar uma IDE que possibilite a compilação dos códigos-fonte. O *VSCode IDE* é uma opção viável para ser utilizado com o ESP-IDF, pois oferece uma extensão que integra completamente o SDK com a IDE. Essa integração fornece recursos como autocompletar, depuração, e a execução direta de comandos para compilar e carregar o código no microcontrolador ESP32.
- *Framework* ESP-IDF 5.1.1: Os prerrequisitos e o passo a passo para a instalação do ESP-IDF no VSCode IDE estão disponibilizados em: [https://docs.espressif.com/projects/vscode-esp-idf-extension/en/latest/]. A Espressif lança novas versões do ESP-IDF com melhorias frequentemente, incluindo versões estáveis e beta. Versões principais são lançadas com menos frequência e costumam introduzir mudanças significativas, o que pode afetar a compatibilidade com projetos desenvolvidos em versões anteriores, a versão instalada foi a 5.1.1.

Por fim, foram realizados o download do executável do 4Diac-IDE e do código-fonte do Forte *Runtime*, versão 2.0.1, *Build* ID: 2024-01-23_181, a partir dos diretórios oficiais disponibilizados na plataforma Eclipse 4Diac.

4.2 Compilação do Forte 4Diac no ESP32

A iniciativa da plataforma 4Diac é *open source* e está sempre sendo atualizada pelos desenvolvedores, atualmente, de forma oficial a plataforma tem suporte para algumas placas de desenvolvimento como BeagleBone Black, Digi Connect ME (ARM7), Lego Mindstorms EV3 (ARM7), NXH 51-ETM e Raspberry PI.

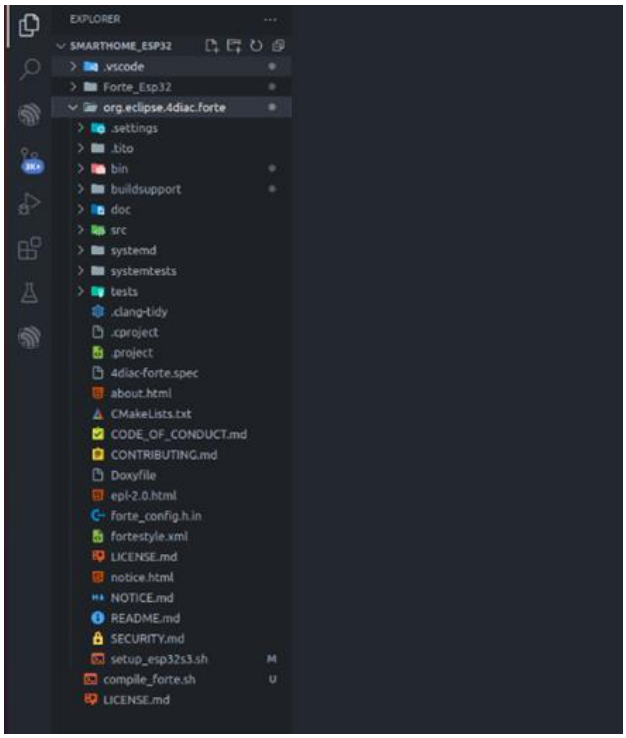
O suporte oficial possibilita a utilização de forma nativa dos blocos que trabalham com a FAL como os blocos de comunicação dos I/Os e blocos de comunicação via rede, sendo necessário apenas configurar a plataforma que vai ser utilizada no momento da compilação. Apesar do ESP32 não ser uma placa oficialmente suportada pela plataforma, possui os pré-requisitos necessários para compilar e trabalhar com o Forte como a arquitetura de 32 *bits*, capacidade de memória *RAM* e *Flash* suficientes

e suporte ao FreeRTOS que é um sistema operacional com suporte oficial da plataforma.

O repositório oficial do Eclipse 4Diac disponibiliza um guia detalhado sobre o processo de compilação do 4Diac Forte para o sistema operacional *FreeRTOS*, acessível em: [https://github.com/eclipse-4diac/4diac-documentation/blob/main/src/installation/freeRTOSLwIP.adoc]. Segundo essas instruções, para realizar a compilação, além de um compilador adequado, são necessários o código-fonte do 4Diac Forte e a biblioteca do FreeRTOS específica para o SoC utilizado.

O primeiro passo foi clonar o repositório oficial do 4Diac Forte em [https://git.eclipse.org/r/4diac/org.eclipse.4diac.forte.git] e criar um diretório de *build.* A Figura 10 ilustra a estrutura da árvore do diretório do projeto, destacando a organização dos arquivos. Além da compilação do 4Diac Forte, foi necessário integrar a biblioteca gerada ao código-fonte do ESP32. Para facilitar esse processo e manter uma estrutura unificada, todos os arquivos do Forte e do ESP32 foram organizados dentro de um mesmo diretório.

Figura 10: Estrutura de Diretórios do Projeto.



Fonte: Imagem gerada pelo autor (2025).

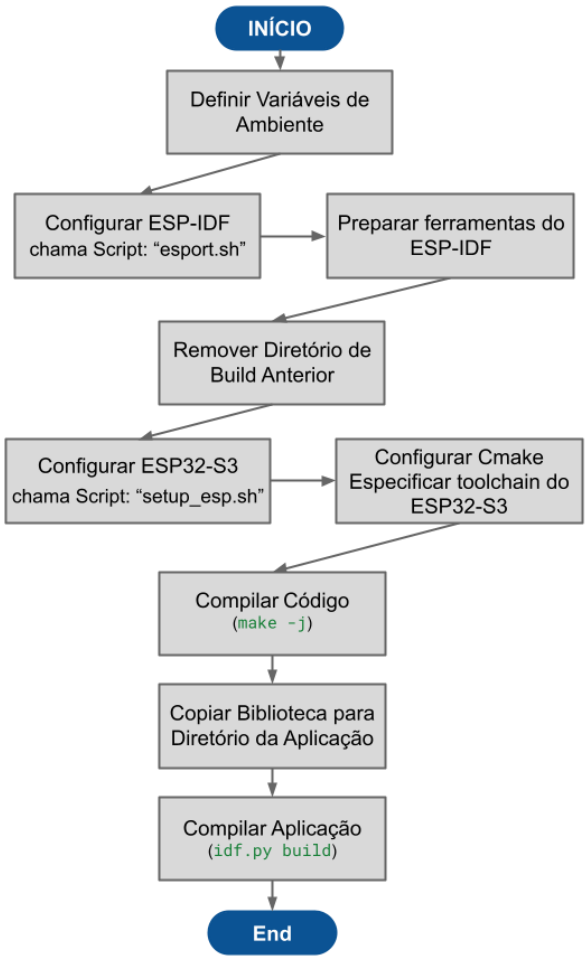
O código fonte do 4Diac Forte fornece alguns arquivos de configuração pré-definidos em *scripts* para as plataformas com suporte oficial. Esses *scripts* foram adaptados para compilação no ESP32, aproveitando a estrutura existente e ajustando os parâmetros necessários. Dessa forma, foi possível facilitar o processo de compilação.

Para a configuração e compilação do *framework* 4Diac Forte no ESP32-S3 foram usados dois *scripts* principais e um arquivo de configuração da *toolchain*. Esses *scripts* foram nomeados como "*compile_forte.sh*" e "*setup_esp.sh*", enquanto o arquivo de configuração foi nomeado "*esp-toolchain.cmake*".

O *script* "*compile_forte.sh*" inicia a compilação do Forte, configurando as variáveis de ambiente e executando o "*export.sh*", um *script* padrão do ESP-IDF que prepara as ferramentas necessárias, na sequência chama o script de configuração do ESP32 "*setup_esp.sh*". A Figura 11 apresenta o fluxo de utilização dos scripts.

O "*setup_esp.sh*" ajusta o ambiente para a arquitetura ESP32-S3. O "*esp-toolchain.cmake*" complementa esse processo ao especificar as *flags* de compilação, otimizações e os diretórios de inclusão necessários para a integração do Forte com o ESP-IDF, incorporando bibliotecas fundamentais como *FreeRTOS* e LwIP. Na sequência o *Cmake* é executado com as configurações realizadas. Após essa preparação, o "compile_forte.sh" remove arquivos de compilação antigos, gera a biblioteca estática "libforte-static.a" move para o diretório do projeto do ESP-IDF e finaliza o processo com o comando "idf.py build", que completa a integração do Forte à aplicação ESP-IDF.

Figura 11: Fluxograma geral de compilação do Forte no ESP32.



Fonte: Imagem gerada pelo autor (2025).

Após adicionar a biblioteca estática compilada (libforte-static.a) ao projeto, é necessário copiar o arquivo “forte_Init.h“, localizado em “src/arch/freeRTOS/” e incluí-lo na função principal do controlador, onde uma tarefa do FreeRTOS será criada para instanciar e executar o Forte continuamente. Essa configuração foi implementada através da criação de um componente para o projeto contendo a inclusão do arquivo “forte_Init.h” e o arquivo “forte_run.cpp”, que implementa e mantém a execução do Forte.

4.2.1 Desenvolvimento e Implementação dos SIFBs

Nesta seção, será abordada a criação e implementação dos blocos de interface de serviço (SIFBs) no 4Diac para o ESP32, possibilitando a interação do controlador

com os GPIOs e a comunicação via MQTT. Conforme abordado na Seção 4.2, a compilação do 4Diac Forte para o ESP32 foi realizada com o ESP-IDF, que oferece as ferramentas de compilação cruzada necessárias. O ESP-IDF também foi utilizado na implementação dos SIFBs, pois disponibiliza bibliotecas que facilitam a integração direta com os recursos do microcontrolador.

O desenvolvimento dos SIFBs iniciou-se com a definição do comportamento de cada bloco, especificando os eventos de entrada e saída responsáveis pela execução das operações definidas no bloco. Em seguida, foi realizada uma verificação de viabilidade com o hardware, assegurando que os blocos pudessem interagir corretamente com os GPIOs e estabelecer comunicação via MQTT.

Para essa integração, os SIFBs precisam de acesso direto às APIs do ESP32, para essas funções o ESP-IDF fornece bibliotecas nativas otimizadas. O controle dos GPIOs foi implementado por meio da API gpio.h, enquanto a comunicação via MQTT utilizou a biblioteca esp-mqtt. Cada SIFB foi implementado como uma nova classe em C++, seguindo a estrutura de código do Forte.

Inicialmente, os blocos foram definidos na 4Diac-IDE, onde foram configuradas apenas as entradas e saídas de eventos e dados. Em seguida, esses blocos foram exportados e editados para a implementação da lógica funcional, utilizando o ESP-IDF para acessar os recursos do ESP32. Por fim, os arquivos modificados foram integrados ao código-fonte do Forte e compilados, garantindo sua execução no ambiente embarcado.

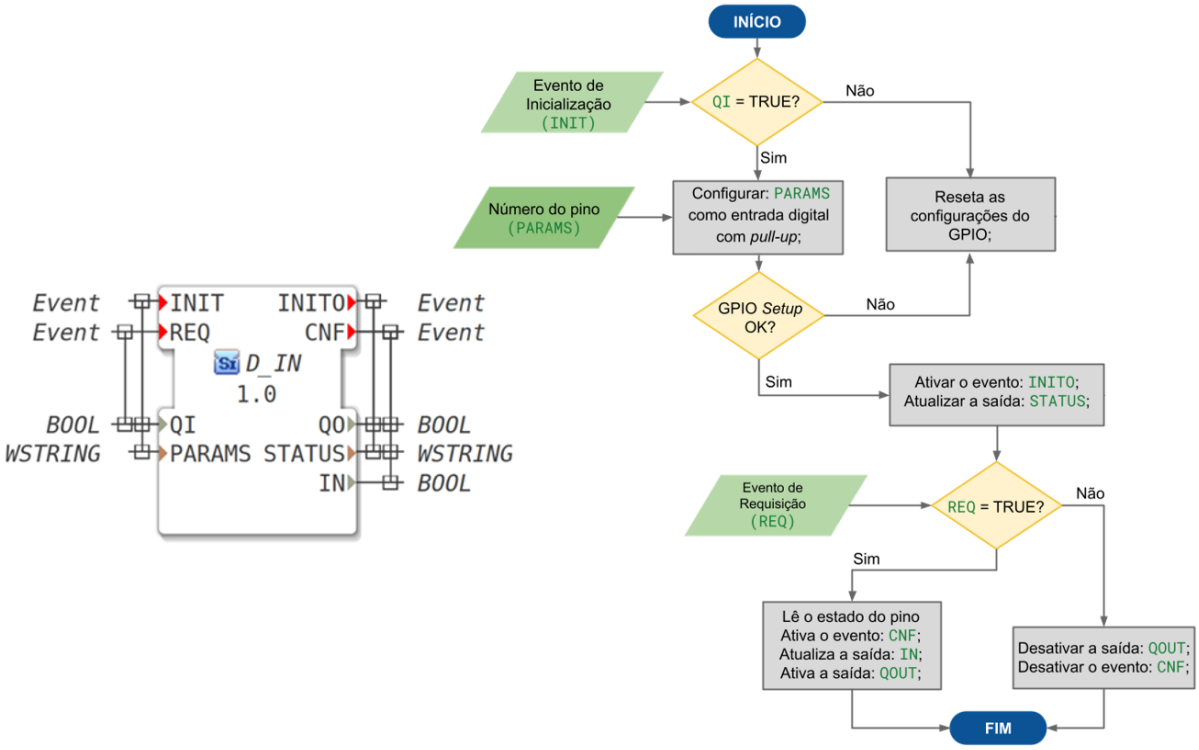
No total, foram desenvolvidos seis blocos de interface de serviço (SIFBs): **D_IN, D_OUT, A_IN, A_OUT, MQTT_PUBLISH e MQTT_SUBSCRIBE.**

- **D_IN**: Configura o GPIO como entrada digital e permite a leitura do estado lógico da entrada. Pode ser utilizado para monitorar botões, sensores de presença ou qualquer outro dispositivo que forneça um sinal binário (0 ou 1).
- **D_OUT**: Configura o GPIO como saída digital, permitindo ativar ou desativar dispositivos como LEDs, relés e atuadores. O bloco recebe um valor lógico (0 ou 1) e escreve o estado no pino correspondente.

- **A_IN**: Realiza a leitura de sinais analógicos através do conversor ADC do ESP32, possibilitando a interface com sensores analógicos, como sensores de temperatura e potenciômetros. A conversão ocorre na faixa de 0 a 3,3V, correspondente a um valor digital de 0 a 4095 (resolução de 12 bits).
- **A_OUT**: Implementa uma saída analógica simulada utilizando PWM, permitindo controlar a intensidade de LEDs, a velocidade de motores e outros dispositivos que respondem a sinais PWM.
- **MQTT_PUBLISH**: Envia mensagens para um tópico MQTT, permitindo a transmissão de dados dos GPIOs ou da lógica interna do sistema para um *broker* local ou na nuvem.
- **MQTT_SUBSCRIBE**: Inscreve-se em um tópico MQTT e recebe mensagens publicadas por outros dispositivos ou sistemas. Esse bloco permite a atualização de variáveis no ESP32 com base nos dados recebidos da rede, viabilizando comandos remotos e a sincronização com outros controladores.

A Figura 12 ilustra a estrutura do bloco **D_IN** e o fluxo lógico do bloco. Os eventos de entrada implementados são INIT e REQ. Ao receber o evento INIT, o bloco configura o GPIO especificado na entrada de dados PARAMS como entrada digital. A leitura do estado do pino ocorre apenas quando o bloco recebe um evento REQ. Os eventos de saída incluem INITO, que sinaliza a conclusão da configuração inicial, e CNF, que confirma a execução do evento REQ. O valor lido no GPIO é disponibilizado na saída IN, enquanto a saída STATUS é atualizada a cada evento, indicando o estado do bloco.

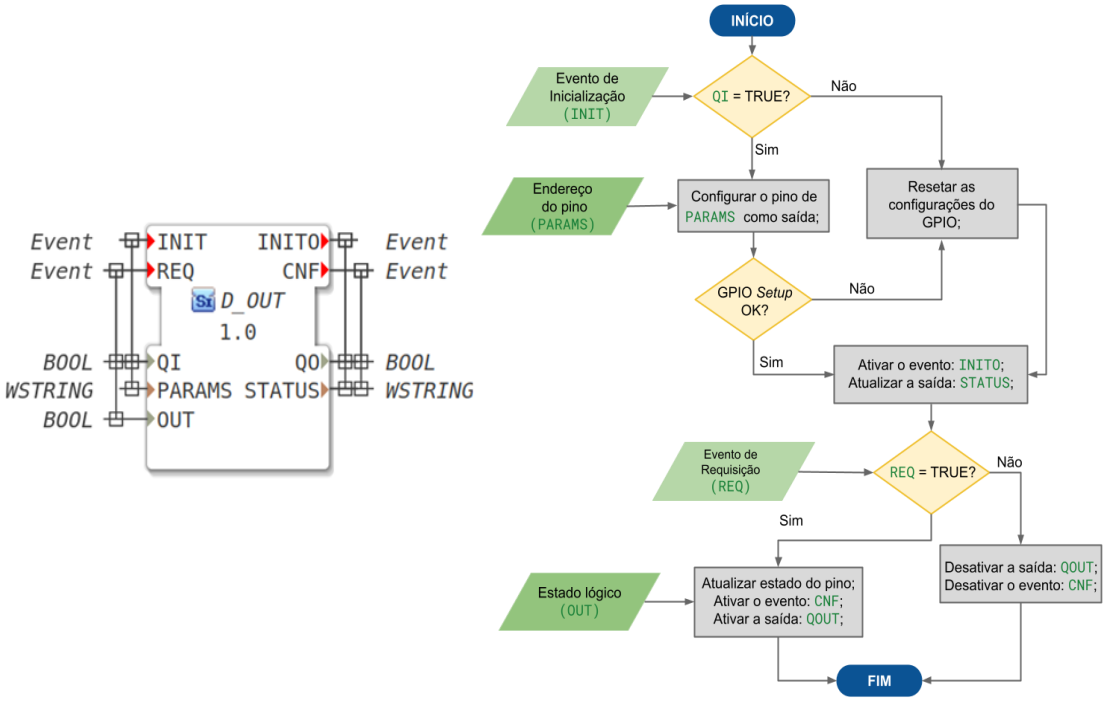
Figura 12: Visão Geral do Bloco de Função D_IN: Estrutura e Lógica.



Fonte: Imagem gerada pelo autor (2025).

O bloco **D_OUT**, ilustrado na Figura 13, segue o mesmo princípio de funcionamento do **D_IN**. O evento INIT realiza a configuração inicial do GPIO, definindo-o como saída digital. Já o evento REQ executa a escrita no pino especificado na entrada PARAMS, utilizando o valor fornecido na entrada OUT do bloco. Esse valor é então aplicado ao GPIO do controlador.

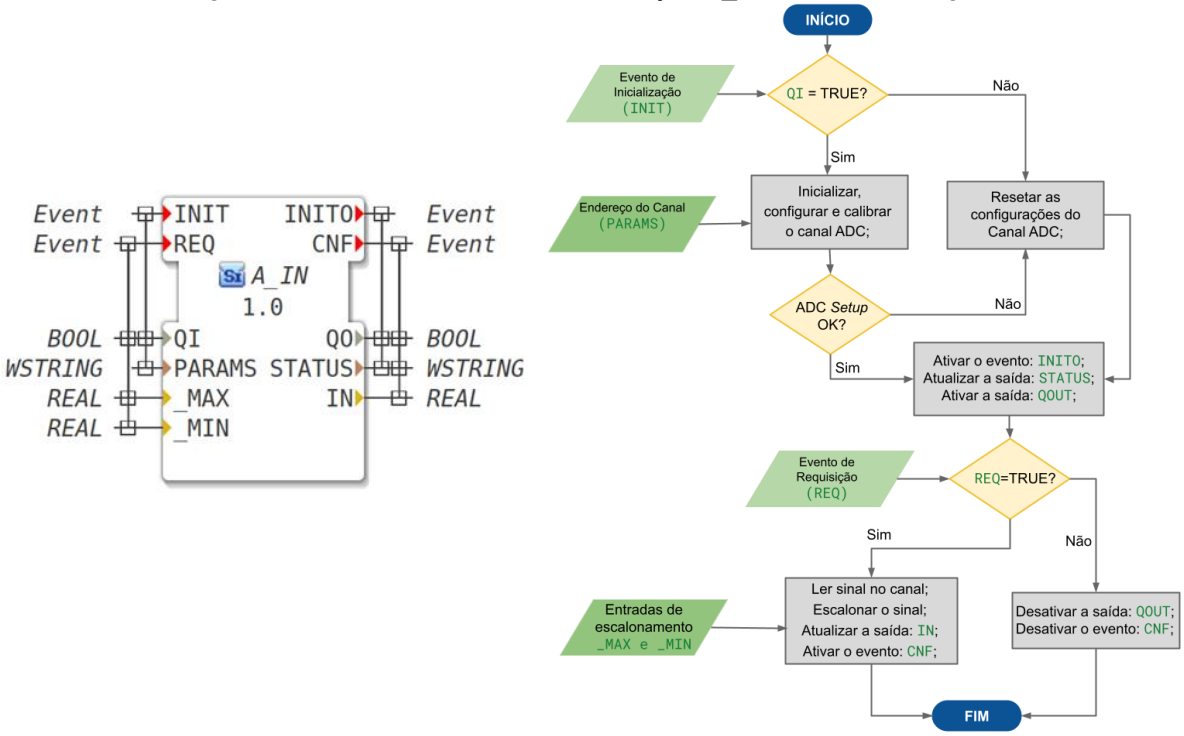
Figura 13: Visão Geral do Bloco de Função D_OUT: Estrutura e Lógica.



Fonte: Imagem gerada pelo autor (2025).

Os blocos **A_IN** e **A_OUT** seguem a mesma estrutura de eventos dos blocos de entrada e saída digital. No caso do bloco **A_IN**, demostrado na Figura 14, além da entrada PARAMS, foram adicionadas as entradas Min e Max, permitindo o escalonamento do sinal para a unidade de engenharia correspondente. Essa funcionalidade é essencial, considerando que o **A_IN** opera dentro da faixa de 0 a 3,3V, garantindo que os valores lidos pelo ADC sejam convertidos corretamente para a escala desejada.

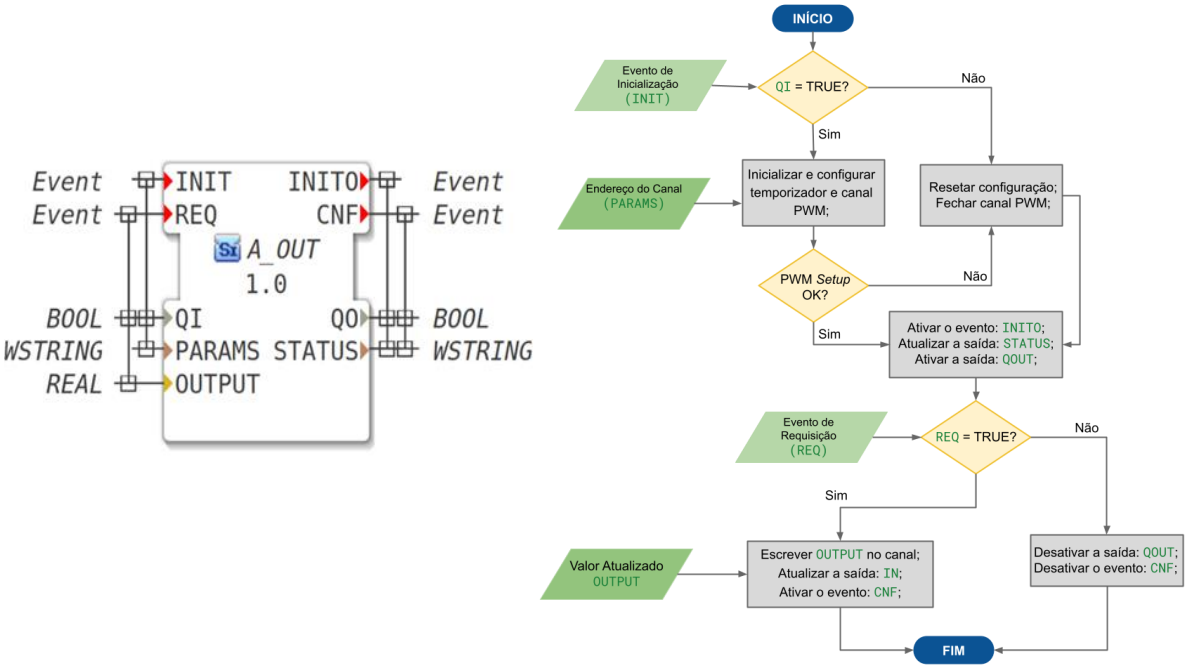
Figura 14: Visão Geral do Bloco de Função A_IN: Estrutura e Lógica.



Fonte: Imagem gerada pelo autor (2025).

No bloco **A_OUT**, demonstrado na Figura 15, a diferença adicional está nos parâmetros que devem ser configurados. A entrada PARAMS recebe os seguintes parâmetros em uma sequência predefinida, separados por ponto ("."): Pino, Temporizador, Canal, Resolução e Frequência. Esses parâmetros determinam a configuração do sinal PWM, definindo em qual GPIO será gerado, qual temporizador interno será utilizado, o canal associado, a resolução do ciclo de trabalho e a frequência do sinal.

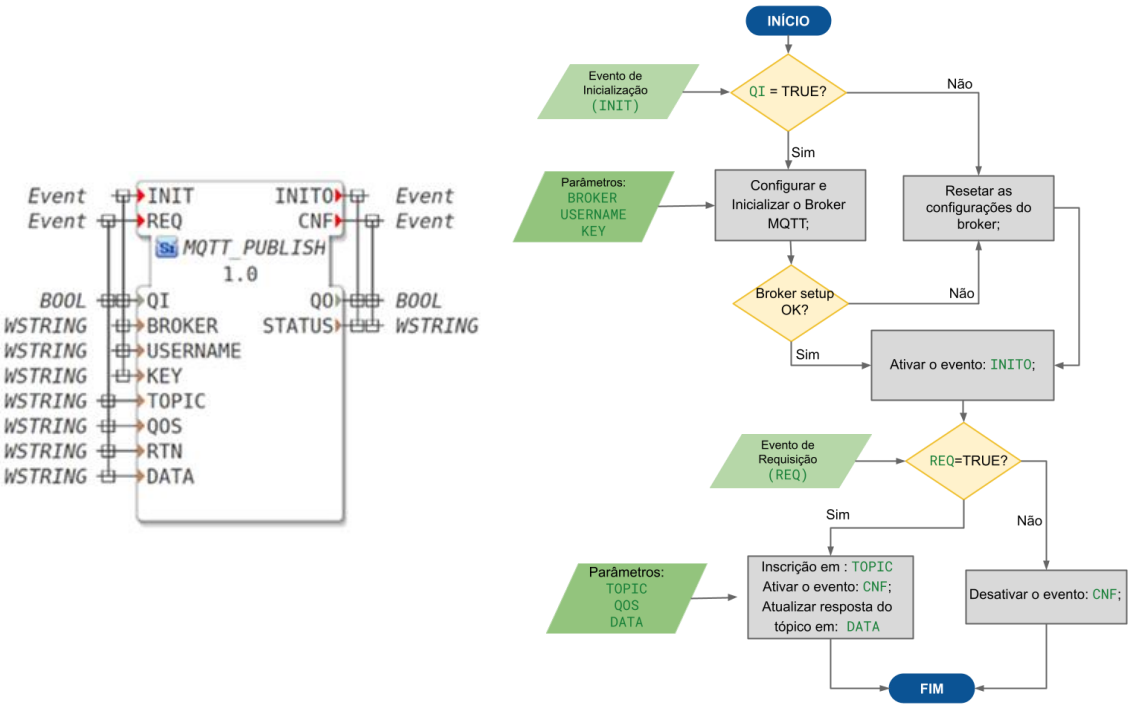
Figura 15: Visão Geral do Bloco de Função A_OUT: Estrutura e Lógica.



Fonte: Imagem gerada pelo autor (2025).

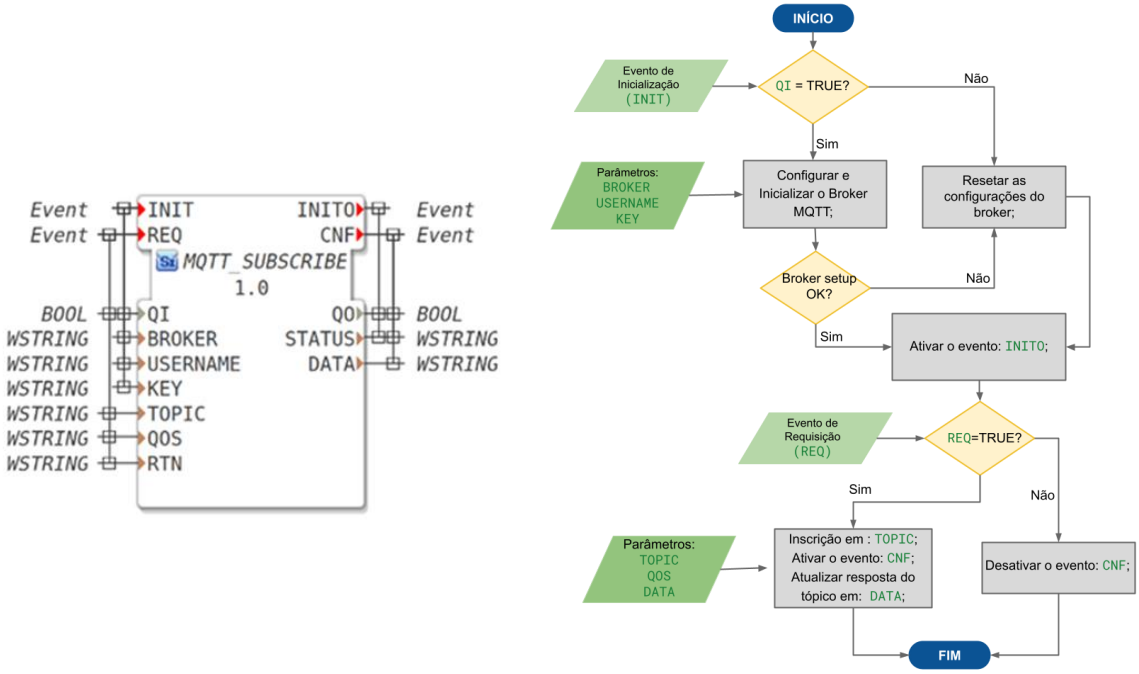
Por fim, os blocos de comunicação com o protocolo MQTT foram implementados seguindo a mesma lógica de eventos. O evento INIT realiza a configuração inicial com o *broker*, enquanto o evento REQ executa o processo de publicação e recebimento de dados nos blocos *publish* e *subscribe*, respectivamente. As Figuras 16 e 17 ilustram os blocos de comunicação MQTT, responsáveis pela publicação e assinatura de mensagens em tópicos de um *broker* MQTT.

Figura 16: Visão Geral do Bloco de Função MQTT_PUBLISH: Estrutura e Lógica.



Fonte: Imagem gerada pelo autor (2025).

Figura 17: Visão Geral do Bloco de Função MQTT_SUBSCRIBE: Estrutura e Lógica.



Fonte: Imagem gerada pelo autor (2025).

5 RESULTADOS

5.1 Estudo de caso

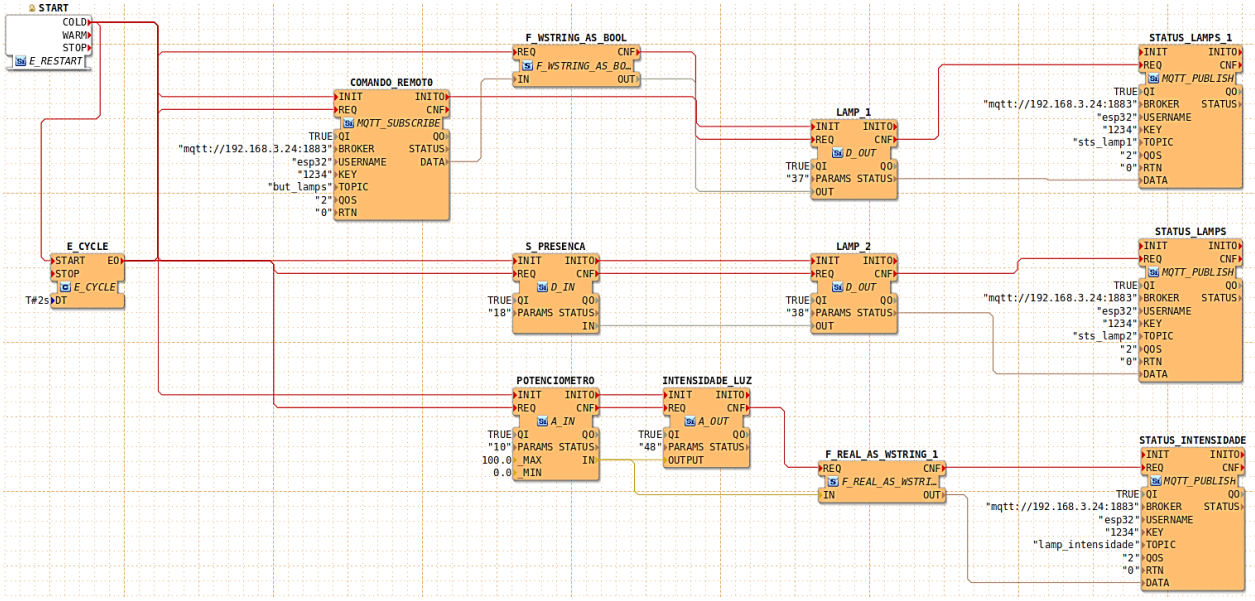
Para validar o funcionamento dos blocos desenvolvidos e a integração com o ESP32-S3, foi realizado um estudo de caso em um ambiente doméstico. O objetivo foi testar a interação dos blocos de interface com os GPIOs e a comunicação MQTT em um cenário simulado com situações comuns da automação residencial, verificando o funcionamento do sistema em conjunto com o Forte.

Os testes foram realizados em um sistema de automação residencial, abrangendo quatro funcionalidades principais: controle de iluminação, controle de intensidade via PWM, detecção de presença e supervisão remota.

O controle de iluminação foi implementado por meio de relés de interface, permitindo o acionamento e desligamento das lâmpadas. O controle de intensidade via PWM foi testado utilizando um potenciômetro para ajustar a luminosidade de um LED, simulando um *dimmer* digital. Além disso, foi integrado um sensor de presença para automatizar o acendimento das luzes em ambientes como corredores ou salas. O sensor detecta movimento e, ao ser ativado, envia um sinal digital ao sistema, acionando a iluminação.

A comunicação e supervisão do sistema foram realizadas via MQTT, utilizando o *broker* Aedes, configurado de forma local no Node-RED. A interface de supervisão também foi desenvolvida no Node-RED, permitindo o monitoramento e controle das variáveis do sistema em tempo real. A Figura 18 apresenta a implementação da lógica no 4Diac-IDE utilizando os blocos desenvolvidos para a comunicação com o hardware do ESP32-S3.

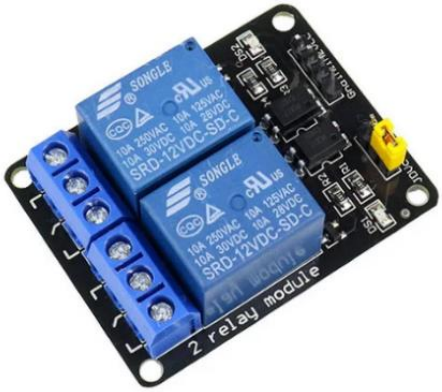
Figura 18: Estrutura da Lógica implementada no 4Diac-IDE para realização do estudo de caso.



Fonte: Imagem gerada pelo autor (2025).

O sistema de teste foi projetado de forma a não apenas validar a aplicação dos blocos, mas também demonstrar sua viabilidade em um ambiente doméstico. Para isso, o acionamento dos LEDs por meio de comandos via MQTT ou sensor de presença foi implementado em um módulo de relés do modelo JQC-3FF-S-Z, mostrado na Figura 19, permitindo o acionamento de cargas de até 250 VCA 10A ou 12 VCC 30A. Além disso, o sensor de presença HC-SR501 PIR, mostrado na Figura 20, foi incorporado, sendo utilizado em aplicações residenciais devido à sua robustez e confiabilidade.

Figura 19: Módulo Relé 2 Canais 12V.



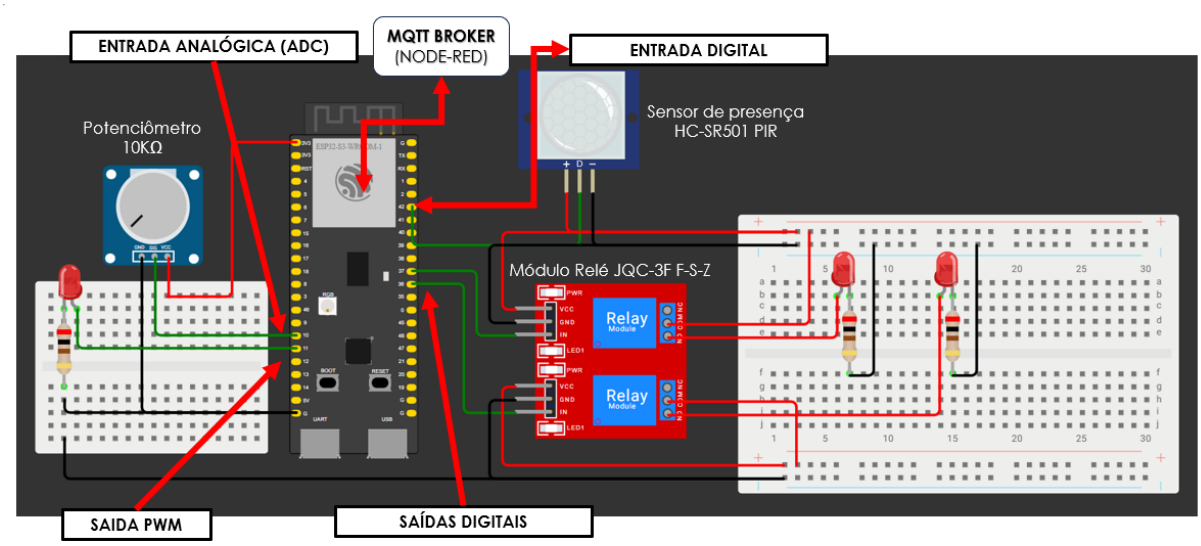
Fonte: Retirada de MONTIMPORT (2025).

Figura 20: Sensor de presença
PIR HC-SR501.

Fonte: Retirada de MONTIMPORT (2025).

A Figura 21 apresenta a organização do circuito montado na bancada de testes para explorar o funcionamento dos blocos desenvolvidos.

Figura 21: Circuito de Testes para Automação Residencial.



Fonte: Imagem gerada pelo autor (2025).

O teste do bloco de saída de controle PWM foi realizado utilizando um LED de 5V, para sua aplicação num ambiente de controle de intensidade lâmpadas de 110V/220V AC deve ser feito utilizando módulos dimmer compatíveis com o PWM gerado na saída do ESP32, como o RobotDyn AC *Dimmer* mostrado na Figura 22, que detecta a passagem por zero da onda senoidal e ajustam o acionamento de um

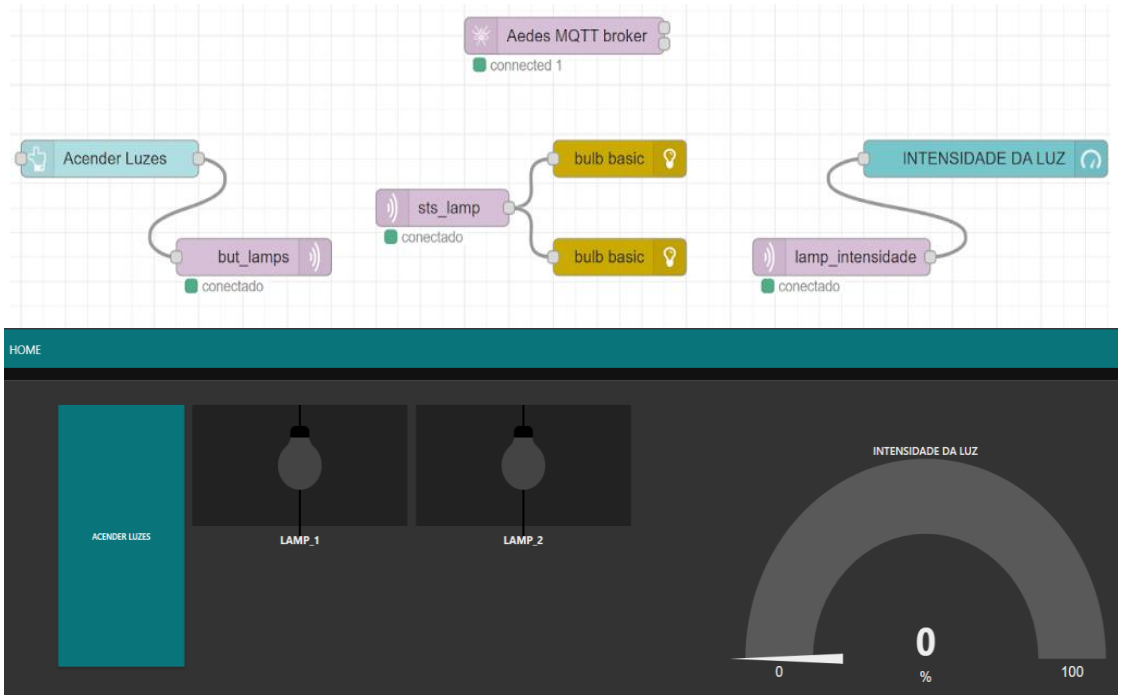
TRIAC (*Triode for Alternating Current*) conforme o *duty cycle* do PWM gerado pelo ESP32. Esse método possibilita variar a intensidade de forma eficiente, sem necessidade de circuitos adicionais.

Figura 22: Módulo *Dimmer* AC PWM da RobotDyn.

Fonte: Retirado de RobotDyn (2025).

A lógica do circuito implementado consistiu no acionamento de duas lâmpadas uma via MQTT e outra por detecção de presença. O estado atual das lâmpadas foi transmitido ao *dashboard* para monitoramento em tempo real. Além disso, foi configurado um potenciômetro para ajuste do ciclo de trabalho da saída PWM de 0 a 100%, permitindo o controle da intensidade luminosa de um terceiro LED. O valor ajustado também é enviado ao *dashboard*. A Figura 23 ilustra a configuração do *dashboard* realizada no Node-RED, incluindo a configuração do *Broker* MQTT.

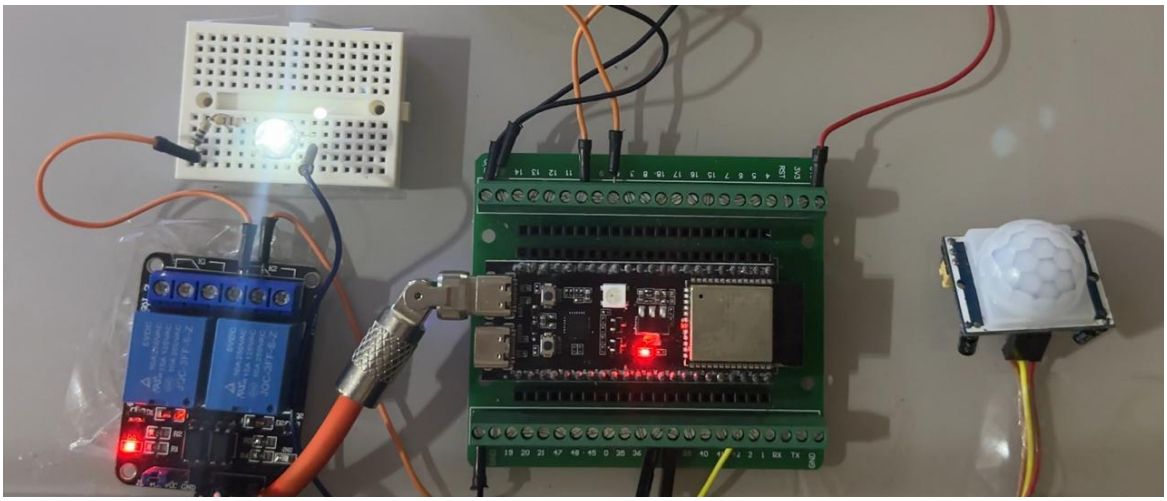
Figura 23: Implementação do *Broker* MQTT AEDES e *dashboard* no Node-RED para monitoramento do sistema em estudo.



Fonte: Imagem gerada pelo autor (2025).

Por fim é possível observar o sistema implementado na bancada de testes na Figura 24.

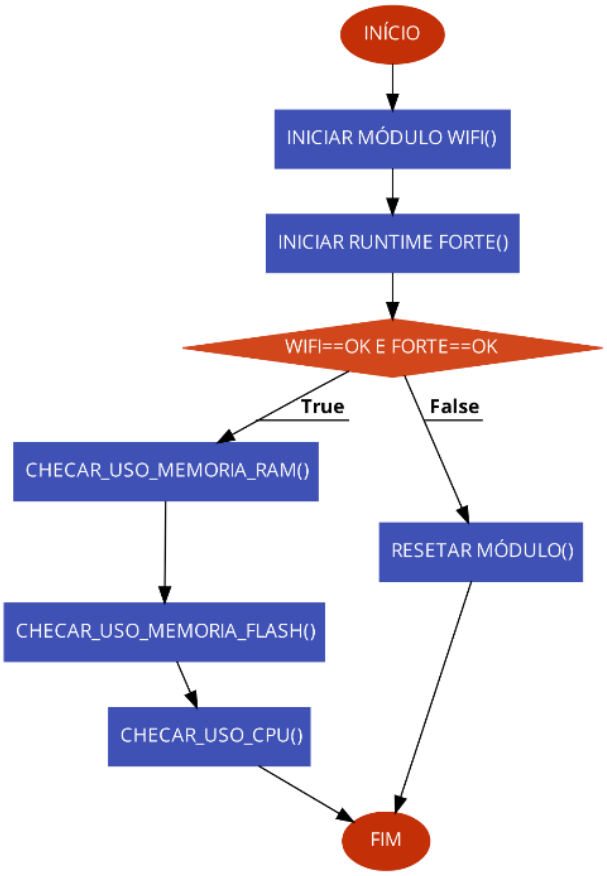
Figura 24: Circuito de teste para simulação do sistema.



Fonte: Imagem gerada pelo autor (2025).

Esse ambiente permitiu testar todos os blocos desenvolvidos, avaliando seu funcionamento, consumo de memória *RAM* e *Flash* do ESP32-S3 bem como a eficiência geral da plataforma. O fluxo do código implementado para avaliar o funcionamento e monitorar também consumo e desempenho do microcontrolador está demostrado na Figura 25. O funcionamento do sistema pode ser analisado no vídeo disponibilizado em: [https://www.youtube.com/watch?v=NU7pUaSENpg.]

Figura 25: Fluxo de funcionamento do código de teste no ESP32.



Fonte: Imagem gerada pelo autor (2025).

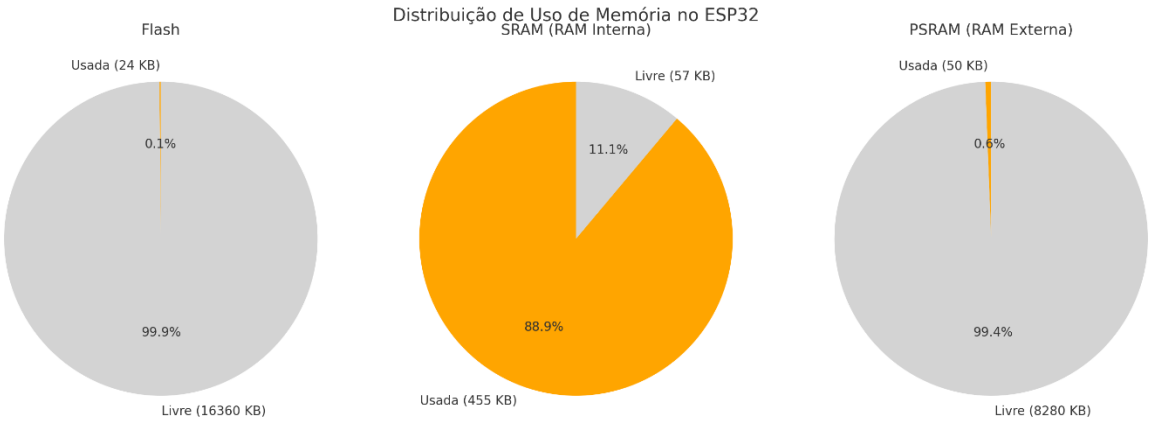
A avaliação de consumo de memórias *flash*, SRAM e PSRAM foi fundamental para avaliar o desempenho e a estabilidade da aplicação. A memória *flash* é responsável por armazenar o *firmware* e os dados persistentes, sendo necessária para comportar o código-fonte e bibliotecas utilizadas.

Já a SRAM, uma memória de acesso mais rápido, é utilizada no armazenamento de variáveis temporárias e estruturas em tempo de execução, essencial para a

operação lógica do programa. De forma complementar, a PSRAM, embora externa, oferece grande capacidade e é indicada para buffers, imagens e dados dinâmicos de maior volume. A análise detalhada dessas memórias permite identificar gargalos, e também direcionar possíveis otimizações na estrutura da aplicação.

A Figura 26 mostra o consumo medido das memórias após a implementação do programa de teste.

Figura 26: Análise de desempenho do ESP32 com o FORTE.



Fonte: Imagem gerada pelo autor (2025).

O consumo de memória no ESP32-S3 mostrou que há um potencial para o desenvolvimento de lógicas bem mais complexas sem comprometer o processamento. O consumo de memória *flash* de 24 KB mostra que há uma ampla gama de memória disponível para implementação de blocos futuros tendo em vista que a memorias *flash* total nessa versão do SoC é de 16MB.

A memória SRAM registrou o maior entre as memorias medidas, ocupando 455 KB dos 512 KB disponíveis, esse dado mostra um ponto crítico para aplicações mais robustas ou com múltiplas instâncias de blocos funcionais, sinalizando a necessidade de realocação de algumas estruturas lógicas para a PSRAM em um projeto de melhoria futura. A memória externa PSRAM, com 8,33 MB disponíveis, teve apenas 50 KB utilizados (≈ 0,6%), demonstrando seu potencial subaproveitado. Esses resultados mostram a viabilidade de melhorar o balanceamento dos recursos de memória do microcontrolador.

Durante a realização dos testes no ambiente doméstico, os componentes do sistema demonstraram um desempenho satisfatório, validando a abordagem proposta. O controle de iluminação funcionou conforme esperado, permitindo o acionamento e desligamento remoto das lâmpadas por meio dos blocos **D_IN** e **D_OUT.** A detecção do nível e o controle de intensidade via PWM, através dos blocos **A_IN** e **A_OUT,** apresentou uma resposta precisa e sem atrasos perceptíveis.

Não foi possível obter uma medida precisa durante a análise do consumo de processamento do ESP32. Essa dificuldade pode ter sido causada pela concorrência de recursos entre as diversas tarefas do sistema, uma vez que o ESP32 estava operando com interrupções para os eventos MQTT e comunicação Wi-Fi, enquanto as tarefas do Forte estavam em execução.

Então foram realizados testes de desempenho variando o tempo entre cada evento de requisição dos blocos e observando a resposta do controlador. Esse intervalo foi utilizado para analisar como o controlador lida com a carga de processamento, já que quanto menor o tempo entre as requisições, mais rápido o processamento precisa ser. Durante os testes, mesmo com a redução do intervalo entre eventos de requisição para até 100 ms, não foram observadas perdas de desempenho nas respostas, tanto nas leituras das entradas e saídas digitais quanto na leitura da entrada ADC e a saída PWM.

A comunicação via MQTT, utilizando o *broker* Aedes no Node-RED, ocorreu de forma estável, sem grandes perdas de mensagens ou inconsistências nos dados transmitidos. Todos os testes foram realizados com o monitoramento via Eclipse IDE e também pelo terminal do ESP-IDF. Durante os testes, observou-se que o uso do terminal serial pode causar instabilidade na comunicação MQTT, especialmente quando os tempos de requisição para publicação e inscrição são inferiores a 1 segundo.

A interface de supervisão no Node-RED possibilitou o monitoramento em tempo real dos status das lâmpadas, e leitura da intensidade desejada da luminosidade através do bloco **MQTT_PUBLISH**, possibilitando, também a detecção do comando remoto para acionamento de uma das lâmpadas através do bloco **MQTT_SUBSCRIBE**.

De forma geral, os blocos desenvolvidos apresentaram um bom funcionamento. No entanto, foram identificados dois pontos de melhoria. O primeiro ocorreu com o bloco **D_OUT**, que, ao ser utilizado mais de uma vez (ou seja, ao chamar um segundo bloco), causava o travamento de todo sistema, como se a segunda chamada do bloco gerasse algum tipo de sobrecarga na comunicação, resultando em uma exceção no FreeRTOS. Esse ponto deve ser analisado e corrigido em futuras atualizações do projeto.

Além disso, o bloco **MQTT_SUBSCRIBE** apresentou falhas na execução dos eventos de saída. Embora a comunicação tenha funcionado, não foi possível usar o bloco para transmitir sinais para outros blocos, o que também requer revisão nas próximas etapas de desenvolvimento.

A integração entre os blocos funcionais do Forte e o microcontrolador ESP32-S3 mostrou-se funcional, agregando valor e confiabilidade ao dispositivo uma vez que passou a funcionar sob o paradigma da norma da norma IEC 61499, revelando um bom desempenho na comunicação com sensores e atuadores, bem como na supervisão remota via MQTT.

Com os resultados obtidos observou-se que a arquitetura distribuída e modular proposta pela norma facilitou a expansão e organização da lógica do sistema, evidenciando seu potencial uso da norma para projetos residenciais de grande porte onde é necessário que seja desenvolvido uma lógica de controle especifica e escalável. Além disso, a análise de consumo de memória indicou que há espaço suficiente para o desenvolvimento de aplicações mais complexas, reforçando a robustez do ambiente escolhido mesmo considerando o uso de um SoC de baixio custo.

Apesar das falhas pontuais encontradas, os testes indicaram estabilidade, característica essencial para sistemas residenciais inteligentes. Dessa forma, os resultados alcançados reforçam a IEC 61499 como uma alternativa viável e promissora frente às soluções proprietárias, sobretudo em projetos que demandam padronização de lógica e alta adaptabilidade.

6 CONCLUSÕES E PROPOSTAS DE CONTINUIDADE

Neste trabalho, foi desenvolvido um sistema de automação residencial baseado na norma IEC 61499, integrando a ferramenta *open source Eclipse* 4Diac com o microcontrolador ESP32-S3. A integração foi realizada por meio da implementação de blocos de serviço para controle das entradas e saídas digitais e analógicas do ESP32-S3, bem como para comunicação via protocolo MQTT.

O estudo confirmou a viabilidade da aplicação da norma IEC 61499 no ESP32-S3, através da execução funcional e eficiente do *runtime* 4Diac Forte, proporcionando um ambiente facilitado para o desenvolvimento de lógica padronizada, com uma abordagem visual que minimiza o uso de programação convencional em microcontroladores, normalmente realizada em linguagens como C ou *Python*.

A contribuição central para a domótica consistiu na demonstração de que a norma IEC 61499 pode ser aplicada a dispositivos de baixo custo e com processamento reduzido, como o ESP32-S3, trazendo modularidade no desenvolvimento de sistemas residenciais inteligentes. A distinção entre a lógica de controle e o acesso ao hardware permite que as aplicações sejam desenvolvidas sem a necessidade de muitas adaptações para outras arquiteturas.

A norma IEC 61499 mostra sua viabilidade especialmente quando se busca a padronização da lógica de controle e o uso de uma arquitetura normatizada que traga mais confiabilidade e segurança aos sistemas de automação. Sua aplicação torna-se ainda mais relevante em cenários complexos, como grandes sistemas residenciais inteligentes, nos quais é necessário um controle robusto e escalável. Nesses casos, a utilização exclusiva de dispositivos proprietários — como relés, sensores ou tomadas inteligentes com controle próprio — pode não ser suficiente para atender às demandas específicas do projeto. Assim, a IEC 61499 surge como uma alternativa para o desenvolvimento de soluções completas a partir do zero, promovendo interoperabilidade e flexibilidade na construção de sistemas personalizados.

Como trabalhos futuros, propõe-se a realização de testes entre o ESP32-S3 e outros controladores, explorando o potencial da IEC 61499 para a integração de dispositivos heterogêneos. Além disso, é importante desenvolver uma ferramenta

mais precisa para monitorar o desempenho do hardware e corrigir os erros encontrados nos blocos **D_OUT** e **MQTT_SUBSCRIBE**.

Por fim, as funcionalidades do microcontrolador, como comunicação SPI, I2C, UART, saída analógica via conversor ADC e integração com protocolos industriais, como Modbus, além da utilização do *Framework* Paho MQTT nativo do FORTE, oferecem oportunidades significativas para futuras melhorias e expansão das capacidades do sistema.

REFERÊNCIAS

4DIAC. Open Source PLC Framework for Industrial Automation & Control. 2024. Disponível em: <www.eclipse.org/4diac/>.

ECLIPSE-4DIAC. FreeRTOS and LwIP installation. 2025. Disponível em: https://github.com/eclipse-4diac/4diac-documentation/blob/main/src/installation/freeRTOSLwIP.adoc. Acesso em: 13 jan. 2025.

AURESIDE, Associação Brasileira de Automação Residencial e Predial. Relatório de previsões para global de Automação residencial. Disponível em: http://www.aureside.org.br/noticias/automacao-residencial---riscos-e-oportunidades Acesso em: 08 de dez. 2024.

BARRY, R. (2010). Using the FreeRTOS Real Time Kernel - a Practical Guide. Real Time Engineers Ltd.

DAMASO, E. G. P. Plataforma Configurável para Gestão de Edifícios baseada em IEC 61499. Dissertação (Mestrado em Engenharia Electrotécnica e de Computadores) - Faculdade de Engenharia da Universidade do Porto (EUP), Portugal, 2011.

DE SOUSA, M. (2010). Analyzing the compatibility between ISA 88 and IEC 61499. Proceedings of the 15th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2010. https://doi.org/10.1109/ETFA.2010.5641308

DIAS, César Luiz de Azevedo; PIZZOLATO, Nélio Domingues. Domótica: Aplicabilidade e Sistemas de Automação Residencial. Revista Vértices, [S. l.], v. 6, n. 3, p. 9–32, 2010. DOI: 10.5935/1809-2667.20040015. Disponível em: https://editoraessentia.iff.edu.br/index.php/vertices/article/view/1809-2667.20040015. Acesso em: 29 set. 2024.

DIAS, César Luiz de Azevedo. Domótica: aplicabilidade às edificações residenciais. Dissertação de mestrado, Universidade Federal Fluminense. Niterói, 2004.

DOMINGUES, R. G. A DOMÓTICA COMO TENDÊNCIA NA HABITAÇÃO: Aplicação em Habitações de Interesse Social com Suporte aos Idosos e Incapacitados. Universidade Federal do Rio de Janeiro. Rio de Janeiro, p. 147. 2013.

DUNKELS, A., Grönvall, B., & Voigt, T. (2004). Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks.

EUGSTER, P. T.; FELBER, P. A.; GUERRAUI, R.; KERMARREC, A.-M. The many faces of publish/subscribe. ACM Computing Surveys, v. 35, n. 2, p. 114-131, 2003.

ESPRESSIF SYSTEMS. ESP-IDF Programming Guide. Disponível em: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/. Acesso em: 22 dez. 2024.

ESPRESSIF. ESP-IDF VS Code Extension. 2025. Disponível em: https://docs.espressif.com/projects/vscode-esp-idf-extension/en/latest/. Acesso em: 10 fev. 2025.

FIRGELLI AUTOMATIONS**.** What is a duty cycle in a linear actuator? Disponível em: https://www.firgelliauto.com/blogs/news/what-is-a-duty-cycle-in-a-linear-actuator. Acesso em: 15 fev. 2025.

FORTUNE BUSINESS INSIGHTS. *South America Smart Home Market Size, Share & Industry Analysis, By Device Type (Safety and Security Devices, Energy and Water Control, Climate Control, Lighting Control, Consumer Electronics), and By Housing Type (Multifamily Dwelling, Single Family Dwelling), Regional Forecast, 2023–2030*. 2025. Disponível em: https://www.fortunebusinessinsights.com/south-america-smart-home-market-107730. Acesso em: 11 abr. 2025.

FORTUNE BUSINESS INSIGHTS. *Home Automation Market Size, Share & Growth Analysis [2032]*. 2025a. Disponível em: https://www.fortunebusinessinsights.com/industry-reports/home-automation-market-100074. Acesso em: 11 abr. 2025.

FORTUNE BUSINESS INSIGHTS. *Smart Home Market Size, Share | Statistics Report [2032]*. 2025b. Disponível em: https://www.fortunebusinessinsights.com/industry-reports/smart-home-market-101900. Acesso em: 11 abr. 2025.

FREERTOS. RTOS fundamentals. Disponível em: https://www.freertos.org/Documentation/01-FreeRTOS-quick-start/01-Beginners-guide/01-RTOS-fundamentals. Acesso em: 17 Dez. 2024.

HAYKIN, Simon. *Signals and systems*. 2. ed. Nova York: John Wiley & Sons, 2001.

HILLAR, Gastón C. MQTT Essentials - A Lightweight IoT Protocol. Birmingham: Packt Publishing, 2017. ISBN 978-1-78728-781-5.

INTERNET ASSIGNED NUMBERS AUTHORITY (IANA). Service Name and Transport Protocol Port Number Registry. 2024. Disponível em: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml. Acesso em: 15 dez. 2024.

LEITÃO, Herbert Albérico de Sá. Metodologia de projeto de controle tolerante a falhas de SEDS aderente à IEC 61499. 2022. Tese (Doutorado em Engenharia Elétrica) – Universidade do Estado de Santa Catarina, Joinville, 2022.

MARIKYAN, Davit, PAPAGIANNIDIS, Savvas, ALAMANOS, Eleftherios. A systematic review of the smart home literature: A user perspective. Technological Forecasting and Social Change, v. 138, p. 139-154, 2019.

MENDES, Marcos Fonseca; PARCIANELLO, Bruna Pletikoszits Andrade. Proposta de automação distribuída de um banco de transformadores reguladores usando a norma IEC 61499. Itaipu Binacional, Divisão de Engenharia Eletrônica e Sistemas de Controle; Universidade Estadual do Oeste do Paraná – Unioeste, Centro de Engenharia e Ciências Exatas. Disponível em: https://educapes.capes.gov.br/bitstream/capes/700959/1/Engenharias.pdf. Acesso em: 12 dez. 2024.

MONTIMPORT. Sensor Detector Presença PIR HC-SR501 Arduino PIC Raspberry. Disponível em: https://www.montimport.com/sensor-detector-presenca-pir-hc-sr501-arduino-pic-raspberry/p/MLB46327924. Acesso em: 10 abr. 2025.

MURATORI, J.; DAL BÓ, P. Automação Residencial: conceitos e aplicações. 2. ed. Belo Horizonte: Educere, 2017.

NOVATRIDA ELETRÔNICA. Módulo Relé 2 Canais 12V. Disponível em: https://www.novatridaeletronica.com.br/modulo-rele-2-canais-12v. Acesso em: 10 abr. 2025.

ROBOTDYN**.** AC Light Dimmer Module 1 Channel 3.3V/5V Logic AC 50-60Hz 220V/110V. Disponível em: https://robotdyn.com/ac-light-dimmer-module-1-channel-3-3v-5v-logic-ac-50-60hz-220v-110v.html. Acesso em: 03 de fev. 2025.

SILVA, Erick Filipe Araújo da. Análise comparativa de estratégias de modulação PWM. 2019. Trabalho de Conclusão de Curso (Engenharia Elétrica) – Instituto Federal da Bahia, Campus Paulo Afonso, 2019.

YASSEIN, Bani; SHATNAWI, M. Q.; ALJWARNEH, S.; AL-HATMI, R. Internet of Things: Survey and open issues of MQTT Protocol. 2017.

APÊNDICES

APÊNDICE A – SCRIPT PRINCIPAL DE COMPILAÇÃO DO FORTE NO ESP32

```
# Define o diretório do código-fonte do FORTE
export forte_dir="org.eclipse.4diac.forte"
# Define o diretório binário de saída para o ESP32-S3
export forte_bin_dir="bin/esp32s3"

# Chama o script de configuração do ESP-IDF para configurar o ambiente de desenvolvimento
. $HOME/esp/esp-idf/export.sh

# Remove a pasta antiga de build (se existir) no diretório de saída
cd "./$forte_dir"
rm -r "./$forte_bin_dir"

# Chama o script de configuração específico para o ESP32-S3
./setup_esp.sh

# Entra no diretório de saída configurado para o ESP32-S3
cd "./$forte_bin_dir"

# Executa o comando make para compilar o código
make -j

# Copia a biblioteca estática compilada para o diretório da aplicação
cp ./src/libforte-static.a ../../../Forte_Esp32/components/forte-esp32-lib

# Volta para o diretório da aplicação
cd ../../../Forte_Esp32

# Executa a compilação final do projeto da aplicação usando o idf.py
idf.py build

echo "--------------------------------------------------------------------------------"
echo "COMPILAÇÃO FINALIZADA: OKAY!"
echo "--------------------------------------------------------------------------------"
```

APÊNDICE B – SCRIPT DE CONFIGURAÇÃO DO ESP32-S3

```
echo "-------------------------------------------------------------------------"
echo " Automatically set up development environment for POSIX-platform"
echo "-------------------------------------------------------------------------"
echo "-------------------------ADAPTED FOR ESP32 S3-----------------------------"
echo " Includes 64bit-datatypes, float-datatypes, Ethernet-Interface,"
echo " ASN1-encoding, ..."
echo ""
echo " To include tests set directories for boost-test-framework and "
echo " set FORTE_TESTS-option to 'ON'"
echo ""
echo "-------------------------------------------------------------------------"

# Define o diretório binário de saída para o ESP32-S3
export forte_bin_dir="bin/esp"
# Define o diretório de suporte de build
export FORTE_BUILDSUPPORT_DIRECTORY="${PWD}/buildsupport"
# Diretórios para inclusão da biblioteca boost para testes
export forte_boost_test_inc_dirs=""
# Diretórios para biblioteca boost para testes
export forte_boost_test_lib_dirs=""

# Cria o diretório binário se ele não existir
if [ ! -d "$forte_bin_dir" ]; then
  mkdir -p "$forte_bin_dir"
fi
# Verifica se o diretório binário existe, se sim, configura o ambiente com CMake
if [ -d "$forte_bin_dir" ]; then
  cd "./$forte_bin_dir"
  # Executa o comando CMake para configurar o projeto FORTE
  cmake -G "Unix Makefiles" \
    -
DCMAKE_TOOLCHAIN_FILE="${FORTE_BUILDSUPPORT_DIRECTORY}/toolchai
ns/esp-toolchain.cmake" \  # Caminho para o arquivo de toolchain do ESP32-S3
    -DCMAKE_C_FLAGS_DEBUG="-g ${CMAKE_C_FLAGS}" \  # Flags de
compilação para Debug
    -DCMAKE_C_FLAGS_MINSIZEREL="-Os -DNDEBUG ${CMAKE_C_FLAGS}" \  #
Flags de compilação para release com otimização
    -DCMAKE_C_FLAGS_RELEASE="-O3 -DNDEBUG ${CMAKE_C_FLAGS}" \  #
Flags de compilação para Release
    -DCMAKE_C_FLAGS_RELWITHDEBINFO="-O2 -g -DNDEBUG
${CMAKE_C_FLAGS}" \  # Flags de compilação para release com debug
    -DCMAKE_CXX_FLAGS_DEBUG="-g ${CMAKE_CXX_FLAGS}" \  # Flags de
compilação C++ para Debug
    -DCMAKE_CXX_FLAGS_MINSIZEREL="-Os -DNDEBUG
${CMAKE_CXX_FLAGS}" \  # Flags de compilação C++ para release
```

```
    -DCMAKE_CXX_FLAGS_RELEASE="-O3 -DNDEBUG ${CMAKE_CXX_FLAGS}"
\  # Flags de compilação C++ para Release
    -DCMAKE_CXX_FLAGS_RELWITHDEBINFO="-O2 -g -DNDEBUG
${CMAKE_CXX_FLAGS}" \  # Flags de compilação C++ para release com debug
    -DFORTE_LINKED_STRINGDICT=OFF \  # Desabilita o dicionário de strings
vinculado
    -DESP32_SDK_CONFIG_DIR="../../../Forte_Esp32/build/config/" \  # Diretório de
configuração do SDK do ESP32
    -DFORTE_FREERTOS_ALLOC="SPIRAM" \  # Alocação do FreeRTOS para
SPIRAM
    -DFORTE_FREERTOS_MINIMAL_STACK_SIZE=15000 \  # Tamanho mínimo da
pilha do FreeRTOS
    -DFORTE_BUILD_EXECUTABLE=OFF \  # Não cria executável, apenas a
biblioteca estática
    -DFORTE_BUILD_STATIC_LIBRARY=ON \  # Cria a biblioteca estática
    -DFORTE_IO=ON \  # Habilita a entrada/saída
    -DFORTE_MODULE_ESP32=ON \  # Habilita o módulo para ESP32
    -DCMAKE_BUILD_TYPE=MINSIZEREL \  # Tipo de build otimizado para menor
tamanho
    -DFORTE_LOGLEVEL=LOGINFO \  # Nível de log
    -DFORTE_COM_ETH=ON \  # Habilita a comunicação Ethernet
    -DFORTE_COM_FBDK=ON \  # Habilita a comunicação com o FBDK (Framework
for Distributed Automation)
    -DFORTE_COM_LOCAL=ON \  # Habilita a comunicação local
    -DFORTE_SUPPORT_BOOT_FILE=ON \  # Habilita suporte para arquivos de
boot
    -DFORTE_BootfileLocation="/data/test_FORTE_PC.fboot" \  # Localização do
arquivo de boot
    -DFORTE_TESTS=OFF \  # Desabilita testes
    -DFORTE_MODULE_CONVERT=ON \  # Habilita o módulo de conversão
    -DFORTE_MODULE_UTILS=ON \  # Habilita módulos utilitários
    -DFORTE_MODULE_EXTERNAL_ESPIO=ON \  # Habilita módulos externos de
IO para ESP32
    -DFORTE_MODULE_EXTERNAL_ESPNET=ON \  # Habilita módulos externos de
rede para ESP32
    -
DFORTE_EXTERNAL_MODULES_DIRECTORY="/home/filipe/Projetos/SmartHome
_esp32/org.eclipse.4diac.forte/src/external" \  # Diretório para módulos externos
    ../../  # Caminho para o diretório onde o código-fonte está localizado
else
  echo "não foi possível criar o diretório ${forte_bin_dir}"
  exit 1  # Caso o diretório não possa ser criado, encerra o script com erro
fi
```

APÊNDICE C – TOOLCHAIN PARA CONFIGURAÇÃO DO ESP32

```
include($ENV{IDF_PATH}/tools/cmake/esp-toolchain.cmake)
```

```
SET(FORTE_ARCHITECTURE "FreeRTOSLwIP")

set(CMAKE_C_FLAGS     "${CMAKE_C_FLAGS} -ffunction-sections -fdata-sections
-fno-threadsafe-statics -fno-rtti -fno-exceptions -Wno-deprecated-declarations -Wno-
attributes")
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -ffunction-sections -fdata-
sections -fno-threadsafe-statics -fno-rtti -fno-exceptions -Wno-deprecated-
declarations -Wno-attributes")
message("CMAKE_C_FLAGS   used for this build: ${CMAKE_C_FLAGS}")
message("CMAKE_CXX_FLAGS used for this build: ${CMAKE_CXX_FLAGS}")

SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/freertos/esp_additions/arch/xtensa/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/lwip/port/freertos/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/lwip/port/esp32xx/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/freertos/esp_additions/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/freertos/esp_additions/include/freertos)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/freertos/FreeRTOS-Kernel/portable/xtensa/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/freertos/FreeRTOS-Kernel/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/freertos/FreeRTOS-Kernel/include/freertos)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/esp_common/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/freertos/port/xtensa/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES $ENV{IDF_PATH}/components/xtensa/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES $ENV{IDF_PATH}/components/lwip/lwip/src)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES $ENV{IDF_PATH}/components/xtensa)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/xtensa/esp32s3/include)
```

```
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES $ENV{IDF_PATH}/components/esp_rom/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/esp_ringbuf/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/esp_rom/include/linux)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/esp_hw_support/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES $ENV{IDF_PATH}/components/hal/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/lwip/lwip/src/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/soc/esp32s3/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES $ENV{IDF_PATH}/components/lwip/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES $ENV{IDF_PATH}/components/lwip/port/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/lwip/port/esp32xx/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/hal/esp32s3/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES $ENV{IDF_PATH}/components/soc/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/esp_system/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/esp_timer/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/newlib/platform_include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES $ENV{IDF_PATH}/components/heap/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES $ENV{IDF_PATH}/components/freertos/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/freertos/include/esp_additions)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES $ENV{IDF_PATH}/components/driver/include)
```

```
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/driver/gpio/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/driver/rmt/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/driver/ledc/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES $ENV{IDF_PATH}/components/log/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/nvs_flash/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/spi_flash/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/esp_partition/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES $ENV{IDF_PATH}/components/esp_adc/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/esp_adc/esp32s3/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES
$ENV{IDF_PATH}/components/esp_event/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES $ENV{IDF_PATH}/components/mqtt/esp-mqtt)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES $ENV{IDF_PATH}/components/mqtt/esp-
mqtt/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES $ENV{IDF_PATH}/components/mqtt/esp-
mqtt/lib)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES $ENV{IDF_PATH}/components/mqtt/esp-
mqtt/lib/include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES $ENV{IDF_PATH}/components/esp-tls)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES $ENV{IDF_PATH}/components/esp-
tls/private_include)
SET_PROPERTY(GLOBAL APPEND PROPERTY
FORTE_INCLUDE_DIRECTORIES ../../Application/components/mqtt_main/include)
```

APÊNDICE D – LÓGICA DO SFIB D_IN

```
/*******************************************************************************
 *** FORTE Library Element
 ***
 *** This file was generated using the 4DIAC FORTE Export Filter V1.0.x NG!
 ***
 *** Name: D_IN
 *** Description: Service Interface Function Block Type
 *** Version:
 ***     1.0: 2023-12-12/filipe -  -
 *******************************************************************************/

#include "D_IN_fbt.h"
#ifdef FORTE_ENABLE_GENERATED_SOURCE_CPP
#include "D_IN_fbt_gen.cpp"
#endif

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <inttypes.h>
#include "driver/gpio.h"


#include "criticalregion.h"
#include "resource.h"

DEFINE_FIRMWARE_FB(FORTE_D_IN, g_nStringIdD_IN)

const CIEC_WSTRING FORTE_D_IN::scmOK("OK");
const CIEC_WSTRING FORTE_D_IN::scmCouldNotWrite("Could not write");
const CIEC_WSTRING FORTE_D_IN::scmCouldNotRead("Could not read");
const CIEC_WSTRING FORTE_D_IN::scmError("Error");
const CIEC_WSTRING FORTE_D_IN::scmNotInitialised("FB not initialized");
const CIEC_WSTRING FORTE_D_IN::scmPinInUse("Pin already in use by other
FB");

const CStringDictionary::TStringId FORTE_D_IN::scmDataInputNames[] =
{g_nStringIdQI, g_nStringIdPARAMS};
const CStringDictionary::TStringId FORTE_D_IN::scmDataInputTypeIds[] =
{g_nStringIdBOOL, g_nStringIdWSTRING};
const CStringDictionary::TStringId FORTE_D_IN::scmDataOutputNames[] =
{g_nStringIdQO, g_nStringIdSTATUS, g_nStringIdIN};
const CStringDictionary::TStringId FORTE_D_IN::scmDataOutputTypeIds[] =
{g_nStringIdBOOL, g_nStringIdWSTRING, g_nStringIdBOOL};
const TDataIOID FORTE_D_IN::scmEIWith[] = {0, 1, scmWithListDelimiter, 0,
scmWithListDelimiter};
const TForteInt16 FORTE_D_IN::scmEIWithIndexes[] = {0, 3};
```

```
const CStringDictionary::TStringId FORTE_D_IN::scmEventInputNames[] =
{g_nStringIdINIT, g_nStringIdREQ};
const TDataIOID FORTE_D_IN::scmEOWith[] = {0, 1, scmWithListDelimiter, 0, 1, 2,
scmWithListDelimiter};
const TForteInt16 FORTE_D_IN::scmEOWithIndexes[] = {0, 3};
const CStringDictionary::TStringId FORTE_D_IN::scmEventOutputNames[] =
{g_nStringIdINITO, g_nStringIdCNF};
const SFBInterfaceSpec FORTE_D_IN::scmFBInterfaceSpec = {
  2, scmEventInputNames, scmEIWith, scmEIWithIndexes,
  2, scmEventOutputNames, scmEOWith, scmEOWithIndexes,
  2, scmDataInputNames, scmDataInputTypeIds,
  3, scmDataOutputNames, scmDataOutputTypeIds,
  0, nullptr,
  0, nullptr
};

FORTE_D_IN::FORTE_D_IN(const CStringDictionary::TStringId paInstanceNameId,
CResource *const paSrcRes) :
    CFunctionBlock(paSrcRes, &scmFBInterfaceSpec, paInstanceNameId),
    var_conn_QO(var_QO),
    var_conn_STATUS(var_STATUS),
    var_conn_IN(var_IN),
    conn_INITO(this, 0),
    conn_CNF(this, 1),
    conn_QI(nullptr),
    conn_PARAMS(nullptr),
    conn_QO(this, 0, &var_conn_QO),
    conn_STATUS(this, 1, &var_conn_STATUS),
    conn_IN(this, 2, &var_conn_IN) {
};

void FORTE_D_IN::setInitialValues() {
  var_QI = 0_BOOL;
  var_PARAMS = u""_WSTRING;
  var_QO = 0_BOOL;
  var_STATUS = u""_WSTRING;
  var_IN = 0_BOOL;
}

bool FORTE_D_IN::initialise(bool paIsInput) {
    bool retVal = false;

    int io_num = std::stoi(var_PARAMS.getValue());

    gpio_config_t io_conf = {};
    //disable interrupt
    io_conf.intr_type = GPIO_INTR_DISABLE;

    if(paIsInput) {
    io_conf.mode = GPIO_MODE_INPUT;
```

```
    io_conf.pull_up_en = GPIO_PULLUP_ENABLE;
    }else {
    io_conf.mode = GPIO_MODE_OUTPUT;
    io_conf.pull_down_en = GPIO_PULLDOWN_DISABLE;
    io_conf.pull_up_en = GPIO_PULLUP_DISABLE;
    }
    //bit mask of the pins that you want to set,e.g.GPIO18/19
    io_conf.pin_bit_mask = (1ULL<<io_num);
    //configure GPIO with the given settings
    int configpin = gpio_config(&io_conf);

    if(!configpin){
      retVal = true;
    } else{
      DEVLOG_ERROR("[FORTE_D_IN::setConfig] - Parameter error ");
    }
  return retVal;
}


bool FORTE_D_IN::deinitialise() {
  bool retVal = false;
  int io_num = std::stoi(var_PARAMS.getValue());
  gpio_num_t pin = static_cast<gpio_num_t>(io_num);

  int result = gpio_reset_pin(pin);
    if(!result) {
      var_STATUS = scmOK;
      retVal = true;
    } else {
      var_STATUS = scmError;
      DEVLOG_ERROR("[FORTE_D_IN::deinitialise] - Error reseting PIN");
    }
  return retVal;
}


bool FORTE_D_IN::readPin() {
 bool retVal = false;
  int io_num = std::stoi(var_PARAMS.getValue());
  gpio_num_t pin = static_cast<gpio_num_t>(io_num);

  bool result = gpio_get_level(pin);
  var_STATUS = scmOK;

  return CIEC_BOOL(result);
}
```

```
void FORTE_D_IN::executeEvent(const TEventID paEIID,
CEventChainExecutionThread *const paECET) {
  switch(paEIID) {
    case scmEventINITID:
     if (var_QI) {
       var_QO = CIEC_BOOL(initialise(true)); //initialise as input
      } else {
       var_QO = CIEC_BOOL(deinitialise());
      }
      sendOutputEvent(scmEventINITOID, paECET);
      break;
    case scmEventREQID:
      if (var_QI) {
       var_IN = CIEC_BOOL(readPin());
      } else {
       var_IN = false_BOOL;
      }
      sendOutputEvent(scmEventCNFID, paECET);
      break;
  }
}

void FORTE_D_IN::readInputData(const TEventID paEIID) {
  switch(paEIID) {
    case scmEventINITID: {
      readData(0, var_QI, conn_QI);
      readData(1, var_PARAMS, conn_PARAMS);
      break;
    }
    case scmEventREQID: {
      readData(0, var_QI, conn_QI);
      break;
    }
    default:
      break;
  }
}

void FORTE_D_IN::writeOutputData(const TEventID paEIID) {
  switch(paEIID) {
    case scmEventINITOID: {
      writeData(0, var_QO, conn_QO);
      writeData(1, var_STATUS, conn_STATUS);
      break;
    }
    case scmEventCNFID: {
      writeData(0, var_QO, conn_QO);
      writeData(1, var_STATUS, conn_STATUS);
      writeData(2, var_IN, conn_IN);
      break;
```

```
    }
    default:
      break;
  }
}

CIEC_ANY *FORTE_D_IN::getDI(const size_t paIndex) {
  switch(paIndex) {
    case 0: return &var_QI;
    case 1: return &var_PARAMS;
  }
  return nullptr;
}

CIEC_ANY *FORTE_D_IN::getDO(const size_t paIndex) {
  switch(paIndex) {
    case 0: return &var_QO;
    case 1: return &var_STATUS;
    case 2: return &var_IN;
  }
  return nullptr;
}

CEventConnection *FORTE_D_IN::getEOConUnchecked(const TPortId paIndex) {
  switch(paIndex) {
    case 0: return &conn_INITO;
    case 1: return &conn_CNF;
  }
  return nullptr;
}

CDataConnection **FORTE_D_IN::getDIConUnchecked(const TPortId paIndex) {
  switch(paIndex) {
    case 0: return &conn_QI;
    case 1: return &conn_PARAMS;
  }
  return nullptr;
}

CDataConnection *FORTE_D_IN::getDOConUnchecked(const TPortId paIndex) {
  switch(paIndex) {
    case 0: return &conn_QO;
    case 1: return &conn_STATUS;
    case 2: return &conn_IN;
  }
  return nullptr;
}
```

APÊNDICE E – LÓGICA DO SFIB D_OUT

```
/*******************************************************************************
 *** FORTE Library Element
 ***
 *** This file was generated using the 4DIAC FORTE Export Filter V1.0.x NG!
 ***
 *** Name: D_OUT
 *** Description: Service Interface Function Block Type
 *** Version:
 ***     1.0: 2023-12-12/filipe -  -
 *******************************************************************************/

#include "D_OUT_fbt.h"
#ifdef FORTE_ENABLE_GENERATED_SOURCE_CPP
#include "D_OUT_fbt_gen.cpp"
#endif

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <inttypes.h>
#include "driver/gpio.h"

#include "criticalregion.h"
#include "resource.h"

DEFINE_FIRMWARE_FB(FORTE_D_OUT, g_nStringIdD_OUT)

const CIEC_WSTRING FORTE_D_OUT::scmOK("OK");
const CIEC_WSTRING FORTE_D_OUT::scmCouldNotWrite("Could not write");
const CIEC_WSTRING FORTE_D_OUT::scmCouldNotRead("Could not read");
const CIEC_WSTRING FORTE_D_OUT::scmError("Error");
const CIEC_WSTRING FORTE_D_OUT::scmNotInitialised("FB not initialized");
const CIEC_WSTRING FORTE_D_OUT::scmPinInUse("Pin already in use by other
FB");

const CStringDictionary::TStringId FORTE_D_OUT::scmDataInputNames[] =
{g_nStringIdQI, g_nStringIdPARAMS, g_nStringIdOUT};
const CStringDictionary::TStringId FORTE_D_OUT::scmDataInputTypeIds[] =
{g_nStringIdBOOL, g_nStringIdWSTRING, g_nStringIdBOOL};
const CStringDictionary::TStringId FORTE_D_OUT::scmDataOutputNames[] =
{g_nStringIdQO, g_nStringIdSTATUS};
const CStringDictionary::TStringId FORTE_D_OUT::scmDataOutputTypeIds[] =
{g_nStringIdBOOL, g_nStringIdWSTRING};
const TDataIOID FORTE_D_OUT::scmEIWith[] = {0, 1, scmWithListDelimiter, 0, 2,
scmWithListDelimiter};
const TForteInt16 FORTE_D_OUT::scmEIWithIndexes[] = {0, 3};
const CStringDictionary::TStringId FORTE_D_OUT::scmEventInputNames[] =
{g_nStringIdINIT, g_nStringIdREQ};
```

```
const TDataIOID FORTE_D_OUT::scmEOWith[] = {0, 1, scmWithListDelimiter, 0, 1,
scmWithListDelimiter};
const TForteInt16 FORTE_D_OUT::scmEOWithIndexes[] = {0, 3};
const CStringDictionary::TStringId FORTE_D_OUT::scmEventOutputNames[] =
{g_nStringIdINITO, g_nStringIdCNF};
const SFBInterfaceSpec FORTE_D_OUT::scmFBInterfaceSpec = {
  2, scmEventInputNames, scmEIWith, scmEIWithIndexes,
  2, scmEventOutputNames, scmEOWith, scmEOWithIndexes,
  3, scmDataInputNames, scmDataInputTypeIds,
  2, scmDataOutputNames, scmDataOutputTypeIds,
  0, nullptr,
  0, nullptr
};

FORTE_D_OUT::FORTE_D_OUT(const CStringDictionary::TStringId
paInstanceNameId, CResource *const paSrcRes) :
    CFunctionBlock(paSrcRes, &scmFBInterfaceSpec, paInstanceNameId),
    var_conn_QO(var_QO),
    var_conn_STATUS(var_STATUS),
    conn_INITO(this, 0),
    conn_CNF(this, 1),
    conn_QI(nullptr),
    conn_PARAMS(nullptr),
    conn_OUT(nullptr),
    conn_QO(this, 0, &var_conn_QO),
    conn_STATUS(this, 1, &var_conn_STATUS) {
};

void FORTE_D_OUT::setInitialValues() {
  var_QI = 0_BOOL;
  var_PARAMS = u""_WSTRING;
  var_OUT = 0_BOOL;
  var_QO = 0_BOOL;
  var_STATUS = u""_WSTRING;
}

bool FORTE_D_OUT::initialise(bool paIsInput) {
    bool retVal = false;

    int io_num = std::stoi(var_PARAMS.getValue());

    gpio_config_t io_conf = {};
    //disable interrupt
    io_conf.intr_type = GPIO_INTR_DISABLE;

    if(paIsInput) {
    io_conf.mode = GPIO_MODE_INPUT;
    io_conf.pull_up_en = GPIO_PULLUP_ENABLE;
    }else {
    io_conf.mode = GPIO_MODE_OUTPUT;
```

```
    io_conf.pull_down_en = GPIO_PULLDOWN_DISABLE;
    io_conf.pull_up_en = GPIO_PULLUP_DISABLE;
    }
    //bit mask of the pins that you want to set,e.g.GPIO18/19
    io_conf.pin_bit_mask = (1ULL<<io_num);
    //configure GPIO with the given settings
    int configpin = gpio_config(&io_conf);

    if(!configpin){
      retVal = true;
    } else{
      DEVLOG_ERROR("[FORTE_D_IN::setConfig] - Parameter error ");
    }
  return retVal;
}

bool FORTE_D_OUT::deinitialise() {
  bool retVal = false;
  int io_num = std::stoi(var_PARAMS.getValue());
  gpio_num_t pin = static_cast<gpio_num_t>(io_num);

  int result = gpio_reset_pin(pin);
    if(!result) {
      var_STATUS = scmOK;
      retVal = true;
    } else {
      var_STATUS = scmError;
      DEVLOG_ERROR("[FORTE_D_IN::deinitialise] - Error reseting PIN");
    }
  return retVal;
}

bool FORTE_D_OUT::writePin(bool paOut) {
  bool retVal = false;
  //unsigned int val = (false != OUT_X()) ? 1 : 0;

  int io_num = std::stoi(var_PARAMS.getValue());
  gpio_num_t pin = static_cast<gpio_num_t>(io_num);

  int result = gpio_set_level(pin,paOut);
  if (!result){
    var_STATUS = scmOK;
    retVal = true;
  } else {
      DEVLOG_ERROR("[CEsp32ProcessInterface::writePin] Could not write %u to
output pin\n", paOut);
      var_STATUS = scmCouldNotWrite;
    }
  return retVal;
}
```

```
void FORTE_D_OUT::executeEvent(const TEventID paEIID,
CEventChainExecutionThread *const paECET) {
  switch(paEIID) {
    case scmEventINITID:
     if (var_QI) {
       var_QO = CIEC_BOOL(initialise(false)); //initialise as input
      } else {
       var_QO = CIEC_BOOL(deinitialise());
      }
      sendOutputEvent(scmEventINITOID, paECET);
      break;
    case scmEventREQID:
      if (var_QI) {
        writePin(static_cast<bool>(var_OUT));
      } else {
       false_BOOL;
      }
      sendOutputEvent(scmEventCNFID, paECET);
      break;
  }
}

void FORTE_D_OUT::readInputData(const TEventID paEIID) {
  switch(paEIID) {
    case scmEventINITID: {
      readData(0, var_QI, conn_QI);
      readData(1, var_PARAMS, conn_PARAMS);
      break;
    }
    case scmEventREQID: {
      readData(0, var_QI, conn_QI);
      readData(2, var_OUT, conn_OUT);
      break;
    }
    default:
      break;
  }
}

void FORTE_D_OUT::writeOutputData(const TEventID paEIID) {
  switch(paEIID) {
    case scmEventINITOID: {
      writeData(0, var_QO, conn_QO);
      writeData(1, var_STATUS, conn_STATUS);
      break;
    }
    case scmEventCNFID: {
      writeData(0, var_QO, conn_QO);
```

```
      writeData(1, var_STATUS, conn_STATUS);
      break;
    }
    default:
      break;
  }
}

CIEC_ANY *FORTE_D_OUT::getDI(const size_t paIndex) {
  switch(paIndex) {
    case 0: return &var_QI;
    case 1: return &var_PARAMS;
    case 2: return &var_OUT;
  }
  return nullptr;
}

CIEC_ANY *FORTE_D_OUT::getDO(const size_t paIndex) {
  switch(paIndex) {
    case 0: return &var_QO;
    case 1: return &var_STATUS;
  }
  return nullptr;
}

CEventConnection *FORTE_D_OUT::getEOConUnchecked(const TPortId paIndex) {
  switch(paIndex) {
    case 0: return &conn_INITO;
    case 1: return &conn_CNF;
  }
  return nullptr;
}

CDataConnection **FORTE_D_OUT::getDIConUnchecked(const TPortId paIndex) {
  switch(paIndex) {
    case 0: return &conn_QI;
    case 1: return &conn_PARAMS;
    case 2: return &conn_OUT;
  }
  return nullptr;
}

CDataConnection *FORTE_D_OUT::getDOConUnchecked(const TPortId paIndex) {
  switch(paIndex) {
    case 0: return &conn_QO;
    case 1: return &conn_STATUS;
  }
  return nullptr;
}
```

APÊNDICE F – LÓGICA DO SFIB A_IN

```
/*******************************************************************************
 *** FORTE Library Element
 ***
 *** This file was generated using the 4DIAC FORTE Export Filter V1.0.x NG!
 ***
 *** Name: A_IN
 *** Description: Service Interface Function Block Type to ESP32 Module
 *** Version:
 ***     1.0: 2023-12-13/Filipe Gomes -  -
 *******************************************************************************/

#include "A_IN_fbt.h"
#ifdef FORTE_ENABLE_GENERATED_SOURCE_CPP
#include "A_IN_fbt_gen.cpp"
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "soc/soc_caps.h"
#include "esp_log.h"
#include "hal/adc_types.h"
#include "esp_adc/adc_oneshot.h"
#include "esp_adc/adc_cali.h"
#include "esp_adc/adc_cali_scheme.h"

#include "criticalregion.h"
#include "resource.h"

static int adc_raw;
static int value;
bool do_calibration_chan;
adc_oneshot_unit_handle_t adc1_handle;
adc_cali_handle_t handle = NULL;
static bool adc_calibration_init(adc_unit_t unit, adc_channel_t channel, adc_atten_t
atten, adc_cali_handle_t *out_handle);
static void adc_calibration_deinit(adc_cali_handle_t handle);

DEFINE_FIRMWARE_FB(FORTE_A_IN, g_nStringIdA_IN)

const CIEC_WSTRING FORTE_A_IN::scmOK("OK");
const CIEC_WSTRING FORTE_A_IN::scmCouldNotWrite("Could not write");
const CIEC_WSTRING FORTE_A_IN::scmCouldNotRead("Could not read");
const CIEC_WSTRING FORTE_A_IN::scmError("Error");
const CIEC_WSTRING FORTE_A_IN::scmNotInitialised("FB not initialized");
```

```
const CIEC_WSTRING FORTE_A_IN::scmPinInUse("Pin already in use by other
FB");

const CStringDictionary::TStringId FORTE_A_IN::scmDataInputNames[] =
{g_nStringIdQI, g_nStringIdPARAMS, g_nStringId_MAX, g_nStringId_MIN};
const CStringDictionary::TStringId FORTE_A_IN::scmDataInputTypeIds[] =
{g_nStringIdBOOL, g_nStringIdWSTRING, g_nStringIdREAL, g_nStringIdREAL};
const CStringDictionary::TStringId FORTE_A_IN::scmDataOutputNames[] =
{g_nStringIdQO, g_nStringIdSTATUS, g_nStringIdIN};
const CStringDictionary::TStringId FORTE_A_IN::scmDataOutputTypeIds[] =
{g_nStringIdBOOL, g_nStringIdWSTRING, g_nStringIdREAL};
const TDataIOID FORTE_A_IN::scmEIWith[] = {0, 1, scmWithListDelimiter, 0, 2, 3,
scmWithListDelimiter};
const TForteInt16 FORTE_A_IN::scmEIWithIndexes[] = {0, 3};
const CStringDictionary::TStringId FORTE_A_IN::scmEventInputNames[] =
{g_nStringIdINIT, g_nStringIdREQ};
const TDataIOID FORTE_A_IN::scmEOWith[] = {0, 1, scmWithListDelimiter, 0, 1, 2,
scmWithListDelimiter};
const TForteInt16 FORTE_A_IN::scmEOWithIndexes[] = {0, 3};
const CStringDictionary::TStringId FORTE_A_IN::scmEventOutputNames[] =
{g_nStringIdINITO, g_nStringIdCNF};
const SFBInterfaceSpec FORTE_A_IN::scmFBInterfaceSpec = {
  2, scmEventInputNames, scmEIWith, scmEIWithIndexes,
  2, scmEventOutputNames, scmEOWith, scmEOWithIndexes,
  4, scmDataInputNames, scmDataInputTypeIds,
  3, scmDataOutputNames, scmDataOutputTypeIds,
  0, nullptr,
  0, nullptr
};

FORTE_A_IN::FORTE_A_IN(const CStringDictionary::TStringId paInstanceNameId,
CResource *const paSrcRes) :
    CFunctionBlock(paSrcRes, &scmFBInterfaceSpec, paInstanceNameId),
    var_QI(false_BOOL),
    var_PARAMS(u""_WSTRING),
    var__MAX(0_SINT),
    var__MIN(0_SINT),
    var_QO(false_BOOL),
    var_STATUS(u""_WSTRING),
    var_IN(0_SINT),
    var_conn_QO(var_QO),
    var_conn_STATUS(var_STATUS),
    var_conn_IN(var_IN),
    conn_INITO(this, 0),
    conn_CNF(this, 1),
    conn_QI(nullptr),
    conn_PARAMS(nullptr),
    conn__MAX(nullptr),
    conn__MIN(nullptr),
    conn_QO(this, 0, &var_conn_QO),
```

```
    conn_STATUS(this, 1, &var_conn_STATUS),
    conn_IN(this, 2, &var_conn_IN) {
};

void FORTE_A_IN::setInitialValues() {
  var_QI = false_BOOL;
  var_PARAMS = u""_WSTRING;
  var__MAX = 0.0_REAL;
  var__MIN = 0.0_REAL;
  var_QO = false_BOOL;
  var_STATUS = u""_WSTRING;
  var_IN = 0.0_REAL;
}

bool FORTE_A_IN::initialise() {

    bool retVal =false;
    bool res_1,res_2;

  //-------------ADC1 Init---------------//
    adc_oneshot_unit_init_cfg_t init_config1 = {
        .unit_id = ADC_UNIT_1,
    };
    res_1 =
ESP_ERROR_CHECK_WITHOUT_ABORT(adc_oneshot_new_unit(&init_config1,
&adc1_handle));

    //-------------ADC1 Config---------------//
    adc_oneshot_chan_cfg_t config = {

        .atten = ADC_ATTEN_DB_11,
        .bitwidth = ADC_BITWIDTH_DEFAULT,

    };

    int io_num = std::stoi(var_PARAMS.getValue());
    adc_channel_t pinAI = static_cast<adc_channel_t>(io_num);

    res_2 =
ESP_ERROR_CHECK_WITHOUT_ABORT(adc_oneshot_config_channel(adc1_han
dle, pinAI, &config));

    //-------------ADC1 Calibration Init---------------//
    adc_cali_handle_t adc1_cali_chan_handle = NULL;
    do_calibration_chan = adc_calibration_init(ADC_UNIT_1, pinAI,
ADC_ATTEN_DB_2_5, &adc1_cali_chan_handle);

    if((!res_1)&&(!res_2)){
      //&&(do_calibration_chan)
      retVal = true;
```

```
    } else{
      var_STATUS = scmError;
      DEVLOG_ERROR("[FORTE_A_IN::setConfig] - Parameter error ");
    }
  return retVal;
}
/*-----------------------------------------------------------------
       ADC Calibration
---------------------------------------------------------------*/
static bool adc_calibration_init(adc_unit_t unit, adc_channel_t channel, adc_atten_t
atten, adc_cali_handle_t *out_handle)
{
    esp_err_t ret = ESP_FAIL;
    bool calibrated = false;

    if (!calibrated) {
        adc_cali_curve_fitting_config_t cali_config = {
            .unit_id = unit,
            .chan = channel,
            .atten = atten,
            .bitwidth = ADC_BITWIDTH_DEFAULT,
        };
        ret = adc_cali_create_scheme_curve_fitting(&cali_config, &handle);
        if (ret == ESP_OK) {
            calibrated = true;
        }
    }
    *out_handle = handle;
    if (ret == ESP_OK) {
         DEVLOG_ERROR("Calibration Success");
    } else if (ret == ESP_ERR_NOT_SUPPORTED || !calibrated) {
         DEVLOG_ERROR("eFuse not burnt, skip software calibration");
    } else {
         DEVLOG_ERROR("Invalid arg or no memory");
    }
    return calibrated;
}

bool FORTE_A_IN::deinitialise() {
  bool retVal = false;
    bool res_3 =
ESP_ERROR_CHECK_WITHOUT_ABORT(adc_oneshot_del_unit(adc1_handle));
    if (do_calibration_chan) {
      if(!res_3 &&
!ESP_ERROR_CHECK_WITHOUT_ABORT(adc_cali_delete_scheme_curve_fitting(
handle)));
          retVal = true;
       }
  return retVal;
}
```

```
float FORTE_A_IN::read_adc() {

  int io_num = std::stoi(var_PARAMS.getValue());
  adc_channel_t pinAI = static_cast<adc_channel_t>(io_num);


  bool res_4 =
ESP_ERROR_CHECK_WITHOUT_ABORT(adc_oneshot_read(adc1_handle, pinAI,
&adc_raw));
  float value = 0;

  if (!res_4 && do_calibration_chan) {

        int original_max = 4095;
        float nova_min = static_cast<float>(var__MIN);
        float nova_max = static_cast<float>(var__MAX);
        float a = (nova_max - nova_min) / 4095;
        value =  a * adc_raw + nova_min;
        //value = adc_raw;
        var_STATUS = scmOK;

      } else var_STATUS = scmCouldNotRead;

      //vTaskDelay(pdMS_TO_TICKS(1000));
      //deinitialise();


  return value;
}

void FORTE_A_IN::executeEvent(const TEventID paEIID,
CEventChainExecutionThread *const paECET) {
  switch(paEIID) {
    case scmEventINITID:

      if (var_QI) {
        var_QO = CIEC_BOOL(initialise()); //initialise as input
      } else {
        //var_QO = CIEC_BOOL(deinitialise());
      }
      sendOutputEvent(scmEventINITOID, paECET);
      break;

    case scmEventREQID:
       if (var_QI) {
        var_IN = CIEC_REAL(read_adc());
      } else {
        //var_IN = CIEC_REAL(deinitialise());
      }
```

```
      sendOutputEvent(scmEventCNFID, paECET);
      break;
  }
}

void FORTE_A_IN::readInputData(const TEventID paEIID) {
  switch(paEIID) {
    case scmEventINITID: {
      readData(0, var_QI, conn_QI);
      readData(1, var_PARAMS, conn_PARAMS);
      break;
    }
    case scmEventREQID: {
      readData(0, var_QI, conn_QI);
      readData(2, var__MAX, conn__MAX);
      readData(3, var__MIN, conn__MIN);
      break;
    }
    default:
      break;
  }
}

void FORTE_A_IN::writeOutputData(const TEventID paEIID) {
  switch(paEIID) {
    case scmEventINITOID: {
      writeData(0, var_QO, conn_QO);
      writeData(1, var_STATUS, conn_STATUS);
      break;
    }
    case scmEventCNFID: {
      writeData(0, var_QO, conn_QO);
      writeData(1, var_STATUS, conn_STATUS);
      writeData(2, var_IN, conn_IN);
      break;
    }
    default:
      break;
  }
}

CIEC_ANY *FORTE_A_IN::getDI(const size_t paIndex) {
  switch(paIndex) {
    case 0: return &var_QI;
    case 1: return &var_PARAMS;
    case 2: return &var__MAX;
    case 3: return &var__MIN;
  }
  return nullptr;
}
```

```
CIEC_ANY *FORTE_A_IN::getDO(const size_t paIndex) {
  switch(paIndex) {
    case 0: return &var_QO;
    case 1: return &var_STATUS;
    case 2: return &var_IN;
  }
  return nullptr;
}

CEventConnection *FORTE_A_IN::getEOConUnchecked(const TPortId paIndex) {
  switch(paIndex) {
    case 0: return &conn_INITO;
    case 1: return &conn_CNF;
  }
  return nullptr;
}

CDataConnection **FORTE_A_IN::getDIConUnchecked(const TPortId paIndex) {
  switch(paIndex) {
    case 0: return &conn_QI;
    case 1: return &conn_PARAMS;
    case 2: return &conn__MAX;
    case 3: return &conn__MIN;
  }
  return nullptr;
}

CDataConnection *FORTE_A_IN::getDOConUnchecked(const TPortId paIndex) {
  switch(paIndex) {
    case 0: return &conn_QO;
    case 1: return &conn_STATUS;
    case 2: return &conn_IN;
  }
  return nullptr;
}
```

APÊNDICE G – LÓGICA DO SFIB A_OUT

```
/*******************************************************************************
 *** FORTE Library Element
 ***
 *** This file was generated using the 4DIAC FORTE Export Filter V1.0.x NG!
 ***
 *** Name: A_OUT
 *** Description: Service Interface Function Block Type to ESP32 Module
 *** Version:
 ***     1.0: 2023-12-20/Filipe Gomes -  -
 *******************************************************************************/

#include "A_OUT_fbt.h"
#ifdef FORTE_ENABLE_GENERATED_SOURCE_CPP
#include "A_OUT_fbt_gen.cpp"
#endif

#include <iostream>
#include <sstream>
#include <string>
#include <stdio.h>
#include "driver/ledc.h"
#include "esp_err.h"
#include <cmath>


#include "criticalregion.h"
#include "resource.h"

// Variáveis para armazenar cada informação
int pin, tim, chnl, res, duty;
uint32_t freq;
ledc_channel_t channel;
ledc_timer_bit_t resol;
ledc_timer_t timer;

DEFINE_FIRMWARE_FB(FORTE_A_OUT, g_nStringIdA_OUT)

const CStringDictionary::TStringId FORTE_A_OUT::scmDataInputNames[] =
{g_nStringIdQI, g_nStringIdPARAMS, g_nStringIdOUTPUT};
const CStringDictionary::TStringId FORTE_A_OUT::scmDataInputTypeIds[] =
{g_nStringIdBOOL, g_nStringIdWSTRING, g_nStringIdREAL};
const CStringDictionary::TStringId FORTE_A_OUT::scmDataOutputNames[] =
{g_nStringIdQO, g_nStringIdSTATUS};
const CStringDictionary::TStringId FORTE_A_OUT::scmDataOutputTypeIds[] =
{g_nStringIdBOOL, g_nStringIdWSTRING};
const TDataIOID FORTE_A_OUT::scmEIWith[] = {0, 1, scmWithListDelimiter, 0, 2,
scmWithListDelimiter};
const TForteInt16 FORTE_A_OUT::scmEIWithIndexes[] = {0, 3};
```

```
const CStringDictionary::TStringId FORTE_A_OUT::scmEventInputNames[] =
{g_nStringIdINIT, g_nStringIdREQ};
const TDataIOID FORTE_A_OUT::scmEOWith[] = {0, 1, scmWithListDelimiter, 0, 1,
scmWithListDelimiter};
const TForteInt16 FORTE_A_OUT::scmEOWithIndexes[] = {0, 3};
const CStringDictionary::TStringId FORTE_A_OUT::scmEventOutputNames[] =
{g_nStringIdINITO, g_nStringIdCNF};
const SFBInterfaceSpec FORTE_A_OUT::scmFBInterfaceSpec = {
  2, scmEventInputNames, scmEIWith, scmEIWithIndexes,
  2, scmEventOutputNames, scmEOWith, scmEOWithIndexes,
  3, scmDataInputNames, scmDataInputTypeIds,
  2, scmDataOutputNames, scmDataOutputTypeIds,
  0, nullptr,
  0, nullptr
};

FORTE_A_OUT::FORTE_A_OUT(const CStringDictionary::TStringId
paInstanceNameId, CResource *const paSrcRes) :
    CFunctionBlock(paSrcRes, &scmFBInterfaceSpec, paInstanceNameId),
    var_conn_QO(var_QO),
    var_conn_STATUS(var_STATUS),
    conn_INITO(this, 0),
    conn_CNF(this, 1),
    conn_QI(nullptr),
    conn_PARAMS(nullptr),
    conn_OUTPUT(nullptr),
    conn_QO(this, 0, &var_conn_QO),
    conn_STATUS(this, 1, &var_conn_STATUS) {
};

void FORTE_A_OUT::setInitialValues() {
  var_QI = 0_BOOL;
  var_PARAMS = u""_WSTRING;
  var_OUTPUT = 0_REAL;
  var_QO = 0_BOOL;
  var_STATUS = u""_WSTRING;
}

bool FORTE_A_OUT::initialise() {
    bool retVal =false;

    std::string params_pwm = var_PARAMS.getValue();
 // Stream para ler a string
    std::stringstream stream(params_pwm);
// Extrair informações
    char delimiter;
    stream >> pin >> delimiter >> tim >> delimiter >> chnl >> delimiter >> res >>
delimiter >> freq;

    channel = static_cast<ledc_channel_t>(chnl);
```

```
    resol = static_cast<ledc_timer_bit_t>(res);
    timer = static_cast<ledc_timer_t>(tim);

// Prepara e aplica as configuracoes do timer
    ledc_timer_config_t ledc_timer = {
        .speed_mode       = LEDC_MODE,
        .duty_resolution  = resol,
        .timer_num        = timer,
        .freq_hz          = freq,  // Set output frequency
        .clk_cfg          = LEDC_AUTO_CLK
    };
    ESP_ERROR_CHECK(ledc_timer_config(&ledc_timer));

    // Prepara e aplica as configuracoes do canal LEDC PWM
    ledc_channel_config_t ledc_channel = {

        .gpio_num       = pin,
        .speed_mode     = LEDC_MODE,
        .channel        = channel,
        .intr_type      = LEDC_INTR_DISABLE,
        .timer_sel      = timer,
        .duty           = 0, // Set duty to 0%
        .hpoint         = 0
    };


ESP_ERROR_CHECK_WITHOUT_ABORT(ledc_channel_config(&ledc_channel));

 return retVal;

}

void FORTE_A_OUT::write_pwm() {

  int resultado = std::pow(2, res) - 1;
  int percent_duty = static_cast<int>(var_OUTPUT);
  int ledc_duty = (resultado*percent_duty)/100;

ESP_ERROR_CHECK_WITHOUT_ABORT(ledc_set_duty(LEDC_MODE,channel,le
dc_duty));
    // Update duty to apply the new value

ESP_ERROR_CHECK_WITHOUT_ABORT(ledc_update_duty(LEDC_MODE,channe
l));
}



void FORTE_A_OUT::executeEvent(const TEventID paEIID,
CEventChainExecutionThread *const paECET) {
```

```
  switch(paEIID) {
    case scmEventINITID:

      if (var_QI) {
       var_QO = CIEC_BOOL(initialise()); //initialise as input
      } else {

      }
      sendOutputEvent(scmEventINITOID, paECET);
      break;

    case scmEventREQID:
     if (var_QI) {
       write_pwm();
      } else {

      }
      sendOutputEvent(scmEventCNFID, paECET);
      break;
  }
}

void FORTE_A_OUT::readInputData(const TEventID paEIID) {
  switch(paEIID) {
    case scmEventINITID: {
      readData(0, var_QI, conn_QI);
      readData(1, var_PARAMS, conn_PARAMS);
      break;
    }
    case scmEventREQID: {
      readData(0, var_QI, conn_QI);
      readData(2, var_OUTPUT, conn_OUTPUT);
      break;
    }
    default:
      break;
  }
}

void FORTE_A_OUT::writeOutputData(const TEventID paEIID) {
  switch(paEIID) {
    case scmEventINITOID: {
      writeData(0, var_QO, conn_QO);
      writeData(1, var_STATUS, conn_STATUS);
      break;
    }
    case scmEventCNFID: {
      writeData(0, var_QO, conn_QO);
      writeData(1, var_STATUS, conn_STATUS);
      break;
```

```
    }
    default:
      break;
  }
}

CIEC_ANY *FORTE_A_OUT::getDI(const size_t paIndex) {
  switch(paIndex) {
    case 0: return &var_QI;
    case 1: return &var_PARAMS;
    case 2: return &var_OUTPUT;
  }
  return nullptr;
}

CIEC_ANY *FORTE_A_OUT::getDO(const size_t paIndex) {
  switch(paIndex) {
    case 0: return &var_QO;
    case 1: return &var_STATUS;
  }
  return nullptr;
}

CEventConnection *FORTE_A_OUT::getEOConUnchecked(const TPortId paIndex) {
  switch(paIndex) {
    case 0: return &conn_INITO;
    case 1: return &conn_CNF;
  }
  return nullptr;
}

CDataConnection **FORTE_A_OUT::getDIConUnchecked(const TPortId paIndex) {
  switch(paIndex) {
    case 0: return &conn_QI;
    case 1: return &conn_PARAMS;
    case 2: return &conn_OUTPUT;
  }
  return nullptr;
}

CDataConnection *FORTE_A_OUT::getDOConUnchecked(const TPortId paIndex) {
  switch(paIndex) {
    case 0: return &conn_QO;
    case 1: return &conn_STATUS;
  }
  return nullptr;
}
```

APÊNDICE H – LÓGICA DO SFIB MQTT_PUBLISH

```
/*******************************************************************************
 *** FORTE Library Element
 ***
 *** This file was generated using the 4DIAC FORTE Export Filter V1.0.x NG!
 ***
 *** Name: MQTT_PUBLISH
 *** Description: Service Interface Function Block Type
 *** Version:
 ***     1.0: 2024-09-19/filipe -  -
 *******************************************************************************/

#include "MQTT_PUBLISH_fbt.h"
#ifdef FORTE_ENABLE_GENERATED_SOURCE_CPP
#include "MQTT_PUBLISH_fbt_gen.cpp"
#endif

#include "criticalregion.h"
#include "resource.h"
#include <string>

DEFINE_FIRMWARE_FB(FORTE_MQTT_PUBLISH, g_nStringIdMQTT_PUBLISH)

const CStringDictionary::TStringId FORTE_MQTT_PUBLISH::scmDataInputNames[]
= {g_nStringIdQI, g_nStringIdBROKER, g_nStringIdUSERNAME, g_nStringIdKEY,
g_nStringIdTOPIC, g_nStringIdQOS, g_nStringIdRTN, g_nStringIdDATA};
const CStringDictionary::TStringId FORTE_MQTT_PUBLISH::scmDataInputTypeIds[]
= {g_nStringIdBOOL, g_nStringIdWSTRING, g_nStringIdWSTRING,
g_nStringIdWSTRING, g_nStringIdWSTRING, g_nStringIdWSTRING,
g_nStringIdWSTRING, g_nStringIdWSTRING};
const CStringDictionary::TStringId
FORTE_MQTT_PUBLISH::scmDataOutputNames[] = {g_nStringIdQO,
g_nStringIdSTATUS};
const CStringDictionary::TStringId
FORTE_MQTT_PUBLISH::scmDataOutputTypeIds[] = {g_nStringIdBOOL,
g_nStringIdWSTRING};
const TDataIOID FORTE_MQTT_PUBLISH::scmEIWith[] = {0, 1, 2, 3,
scmWithListDelimiter, 0, 4, 5, 6, 7, scmWithListDelimiter};
const TForteInt16 FORTE_MQTT_PUBLISH::scmEIWithIndexes[] = {0, 5};
const CStringDictionary::TStringId
FORTE_MQTT_PUBLISH::scmEventInputNames[] = {g_nStringIdINIT,
g_nStringIdREQ};
const TDataIOID FORTE_MQTT_PUBLISH::scmEOWith[] = {0, 1,
scmWithListDelimiter, 0, 1, scmWithListDelimiter};
const TForteInt16 FORTE_MQTT_PUBLISH::scmEOWithIndexes[] = {0, 3};
const CStringDictionary::TStringId
FORTE_MQTT_PUBLISH::scmEventOutputNames[] = {g_nStringIdINITO,
g_nStringIdCNF};
const SFBInterfaceSpec FORTE_MQTT_PUBLISH::scmFBInterfaceSpec = {
```

```
  2, scmEventInputNames, scmEIWith, scmEIWithIndexes,
  2, scmEventOutputNames, scmEOWith, scmEOWithIndexes,
  8, scmDataInputNames, scmDataInputTypeIds,
  2, scmDataOutputNames, scmDataOutputTypeIds,
  0, nullptr,
  0, nullptr
};

FORTE_MQTT_PUBLISH::FORTE_MQTT_PUBLISH(const
CStringDictionary::TStringId paInstanceNameId, CResource *const paSrcRes) :
    CFunctionBlock(paSrcRes, &scmFBInterfaceSpec, paInstanceNameId),
    var_conn_QO(var_QO),
    var_conn_STATUS(var_STATUS),
    conn_INITO(this, 0),
    conn_CNF(this, 1),
    conn_QI(nullptr),
    conn_BROKER(nullptr),
    conn_USERNAME(nullptr),
    conn_KEY(nullptr),
    conn_TOPIC(nullptr),
    conn_QOS(nullptr),
    conn_RTN(nullptr),
    conn_DATA(nullptr),
    conn_QO(this, 0, &var_conn_QO),
    conn_STATUS(this, 1, &var_conn_STATUS) {
};

void FORTE_MQTT_PUBLISH::setInitialValues() {
  var_QI = 0_BOOL;
  var_BROKER = u""_WSTRING;
  var_USERNAME = u""_WSTRING;
  var_KEY = u""_WSTRING;
  var_TOPIC = u""_WSTRING;
  var_QOS = u""_WSTRING;
  var_RTN = u""_WSTRING;
  var_DATA = u""_WSTRING;
  var_QO = 0_BOOL;
  var_STATUS = u""_WSTRING;
}

void FORTE_MQTT_PUBLISH::executeEvent(const TEventID paEIID,
CEventChainExecutionThread *const paECET) {

      std::string BROKER_S = var_BROKER.getValue();
      const char* BROKER = BROKER_S.c_str();

      std::string USERNAME_S = var_USERNAME.getValue();
      const char* USERNAME = USERNAME_S.c_str();

      std::string KEY_S = var_KEY.getValue();
```

```
      const char* KEY = KEY_S.c_str();

      std::string TOPIC_S = var_TOPIC.getValue();
      const char* TOPIC = TOPIC_S.c_str();

      std::string QOS_S = var_QOS.getValue();
      int QOS = std::stoi(QOS_S);

      std::string RTN_S = var_RTN.getValue();
      int RTN = std::stoi(RTN_S);

      std::string DATA_S = var_DATA.getValue();
      const char* DATA = DATA_S.c_str();

      std::string STATUS_S = var_STATUS.getValue();
      const char* STATUS = STATUS_S.c_str();

  switch(paEIID) {
    case scmEventINITID:
      mqtt_app_start(BROKER,USERNAME,KEY);
      sendOutputEvent(scmEventINITOID, paECET);

      break;
    case scmEventREQID:
      mqtt_publisher(TOPIC,DATA, QOS, RTN);
      sendOutputEvent(scmEventCNFID, paECET);

      break;
  }
}

void FORTE_MQTT_PUBLISH::readInputData(const TEventID paEIID) {
  switch(paEIID) {
    case scmEventINITID: {
      readData(0, var_QI, conn_QI);
      readData(1, var_BROKER, conn_BROKER);
      readData(2, var_USERNAME, conn_USERNAME);
      readData(3, var_KEY, conn_KEY);
      break;
    }
    case scmEventREQID: {
      readData(0, var_QI, conn_QI);
      readData(4, var_TOPIC, conn_TOPIC);
      readData(5, var_QOS, conn_QOS);
      readData(6, var_RTN, conn_RTN);
      readData(7, var_DATA, conn_DATA);
      break;
    }
    default:
      break;
```

```
  }
}

void FORTE_MQTT_PUBLISH::writeOutputData(const TEventID paEIID) {
  switch(paEIID) {
    case scmEventINITOID: {
      writeData(0, var_QO, conn_QO);
      writeData(1, var_STATUS, conn_STATUS);
      break;
    }
    case scmEventCNFID: {
      writeData(0, var_QO, conn_QO);
      writeData(1, var_STATUS, conn_STATUS);
      break;
    }
    default:
      break;
  }
}

CIEC_ANY *FORTE_MQTT_PUBLISH::getDI(const size_t paIndex) {
  switch(paIndex) {
    case 0: return &var_QI;
    case 1: return &var_BROKER;
    case 2: return &var_USERNAME;
    case 3: return &var_KEY;
    case 4: return &var_TOPIC;
    case 5: return &var_QOS;
    case 6: return &var_RTN;
    case 7: return &var_DATA;
  }
  return nullptr;
}

CIEC_ANY *FORTE_MQTT_PUBLISH::getDO(const size_t paIndex) {
  switch(paIndex) {
    case 0: return &var_QO;
    case 1: return &var_STATUS;
  }
  return nullptr;
}

CEventConnection *FORTE_MQTT_PUBLISH::getEOConUnchecked(const TPortId
paIndex) {
  switch(paIndex) {
    case 0: return &conn_INITO;
    case 1: return &conn_CNF;
  }
  return nullptr;
}
```

```
CDataConnection **FORTE_MQTT_PUBLISH::getDIConUnchecked(const TPortId
paIndex) {
  switch(paIndex) {
    case 0: return &conn_QI;
    case 1: return &conn_BROKER;
    case 2: return &conn_USERNAME;
    case 3: return &conn_KEY;
    case 4: return &conn_TOPIC;
    case 5: return &conn_QOS;
    case 6: return &conn_RTN;
    case 7: return &conn_DATA;
  }
  return nullptr;
}

CDataConnection *FORTE_MQTT_PUBLISH::getDOConUnchecked(const TPortId
paIndex) {
  switch(paIndex) {
    case 0: return &conn_QO;
    case 1: return &conn_STATUS;
  }
  return nullptr;
}
```

APÊNDICE I – LÓGICA DO SFIB MQTT_SUBSCRIBE

```
/*******************************************************************************
 *** FORTE Library Element
 ***
 *** This file was generated using the 4DIAC FORTE Export Filter V1.0.x NG!
 ***
 *** Name: MQTT_SUBSCRIBE
 *** Description: Service Interface Function Block Type
 *** Version:
 ***     1.0: 2024-09-19/filipe -  -
 *******************************************************************************/
#include "MQTT_SUBSCRIBE_fbt.h"
#ifdef FORTE_ENABLE_GENERATED_SOURCE_CPP
#include "MQTT_SUBSCRIBE_fbt_gen.cpp"
#endif

#include "criticalregion.h"
#include "resource.h"

char data_buffer[100];
char topic_buffer[100];

DEFINE_FIRMWARE_FB(FORTE_MQTT_SUBSCRIBE,
g_nStringIdMQTT_SUBSCRIBE)

const CStringDictionary::TStringId
FORTE_MQTT_SUBSCRIBE::scmDataInputNames[] = {g_nStringIdQI,
g_nStringIdBROKER, g_nStringIdUSERNAME, g_nStringIdKEY, g_nStringIdTOPIC,
g_nStringIdQOS, g_nStringIdRTN};
const CStringDictionary::TStringId
FORTE_MQTT_SUBSCRIBE::scmDataInputTypeIds[] = {g_nStringIdBOOL,
g_nStringIdWSTRING, g_nStringIdWSTRING, g_nStringIdWSTRING,
g_nStringIdWSTRING, g_nStringIdWSTRING, g_nStringIdWSTRING};
const CStringDictionary::TStringId
FORTE_MQTT_SUBSCRIBE::scmDataOutputNames[] = {g_nStringIdQO,
g_nStringIdSTATUS, g_nStringIdDATA};
const CStringDictionary::TStringId
FORTE_MQTT_SUBSCRIBE::scmDataOutputTypeIds[] = {g_nStringIdBOOL,
g_nStringIdWSTRING, g_nStringIdWSTRING};
const TDataIOID FORTE_MQTT_SUBSCRIBE::scmEIWith[] = {0, 1, 2, 3,
scmWithListDelimiter, 0, 4, 5, 6, scmWithListDelimiter};
const TForteInt16 FORTE_MQTT_SUBSCRIBE::scmEIWithIndexes[] = {0, 5};
const CStringDictionary::TStringId
FORTE_MQTT_SUBSCRIBE::scmEventInputNames[] = {g_nStringIdINIT,
g_nStringIdREQ};
const TDataIOID FORTE_MQTT_SUBSCRIBE::scmEOWith[] = {0, 1,
scmWithListDelimiter, 0, 1, 2, scmWithListDelimiter};
const TForteInt16 FORTE_MQTT_SUBSCRIBE::scmEOWithIndexes[] = {0, 3};
```

```
const CStringDictionary::TStringId
FORTE_MQTT_SUBSCRIBE::scmEventOutputNames[] = {g_nStringIdINITO,
g_nStringIdCNF};
const SFBInterfaceSpec FORTE_MQTT_SUBSCRIBE::scmFBInterfaceSpec = {
  2, scmEventInputNames, scmEIWith, scmEIWithIndexes,
  2, scmEventOutputNames, scmEOWith, scmEOWithIndexes,
  7, scmDataInputNames, scmDataInputTypeIds,
  3, scmDataOutputNames, scmDataOutputTypeIds,
  0, nullptr,
  0, nullptr
};

FORTE_MQTT_SUBSCRIBE::FORTE_MQTT_SUBSCRIBE(const
CStringDictionary::TStringId paInstanceNameId, CResource *const paSrcRes) :
    CFunctionBlock(paSrcRes, &scmFBInterfaceSpec, paInstanceNameId),
    var_conn_QO(var_QO),
    var_conn_STATUS(var_STATUS),
    var_conn_DATA(var_DATA),
    conn_INITO(this, 0),
    conn_CNF(this, 1),
    conn_QI(nullptr),
    conn_BROKER(nullptr),
    conn_USERNAME(nullptr),
    conn_KEY(nullptr),
    conn_TOPIC(nullptr),
    conn_QOS(nullptr),
    conn_RTN(nullptr),
    conn_QO(this, 0, &var_conn_QO),
    conn_STATUS(this, 1, &var_conn_STATUS),
    conn_DATA(this, 2, &var_conn_DATA) {
};

void FORTE_MQTT_SUBSCRIBE::setInitialValues() {
  var_QI = 0_BOOL;
  var_BROKER = u""_WSTRING;
  var_USERNAME = u""_WSTRING;
  var_KEY = u""_WSTRING;
  var_TOPIC = u""_WSTRING;
  var_QOS = u""_WSTRING;
  var_RTN = u""_WSTRING;
  var_QO = 0_BOOL;
  var_STATUS = u""_WSTRING;
  var_DATA = u""_WSTRING;
}

void FORTE_MQTT_SUBSCRIBE::executeEvent(const TEventID paEIID,
CEventChainExecutionThread *const paECET) {

   std::string BROKER_S = var_BROKER.getValue();
      const char* BROKER = BROKER_S.c_str();
```

```
    std::string USERNAME_S = var_USERNAME.getValue();
    const char* USERNAME = USERNAME_S.c_str();

    std::string KEY_S = var_KEY.getValue();
    const char* KEY = KEY_S.c_str();

    std::string TOPIC_S = var_TOPIC.getValue();
    const char* TOPIC = TOPIC_S.c_str();

    std::string QOS_S = var_QOS.getValue();
    int QOS = std::stoi(QOS_S);

    std::string RTN_S = var_RTN.getValue();
    int RTN = std::stoi(RTN_S);


  switch(paEIID) {
    case scmEventINITID:
      mqtt_app_start(BROKER,USERNAME,KEY);

      break;
    case scmEventREQID:
      mqtt_subscriber(TOPIC, QOS);

        if (topic_buffer == TOPIC_S){
         var_DATA = CIEC_WSTRING(data_buffer);
          }
      break;
  }
}

void FORTE_MQTT_SUBSCRIBE::readInputData(const TEventID paEIID) {
  switch(paEIID) {
    case scmEventINITID: {
      readData(0, var_QI, conn_QI);
      readData(1, var_BROKER, conn_BROKER);
      readData(2, var_USERNAME, conn_USERNAME);
      readData(3, var_KEY, conn_KEY);
      break;
    }
    case scmEventREQID: {
      readData(0, var_QI, conn_QI);
      readData(4, var_TOPIC, conn_TOPIC);
      readData(5, var_QOS, conn_QOS);
      readData(6, var_RTN, conn_RTN);
      break;
    }
    default:
      break;
```

```
  }
}

void FORTE_MQTT_SUBSCRIBE::writeOutputData(const TEventID paEIID) {
  switch(paEIID) {
    case scmEventINITOID: {
      writeData(0, var_QO, conn_QO);
      writeData(1, var_STATUS, conn_STATUS);
      break;
    }
    case scmEventCNFID: {
      writeData(0, var_QO, conn_QO);
      writeData(1, var_STATUS, conn_STATUS);
      writeData(2, var_DATA, conn_DATA);
      break;
    }
    default:
      break;
  }
}

CIEC_ANY *FORTE_MQTT_SUBSCRIBE::getDI(const size_t paIndex) {
  switch(paIndex) {
    case 0: return &var_QI;
    case 1: return &var_BROKER;
    case 2: return &var_USERNAME;
    case 3: return &var_KEY;
    case 4: return &var_TOPIC;
    case 5: return &var_QOS;
    case 6: return &var_RTN;
  }
  return nullptr;
}

CIEC_ANY *FORTE_MQTT_SUBSCRIBE::getDO(const size_t paIndex) {
  switch(paIndex) {
    case 0: return &var_QO;
    case 1: return &var_STATUS;
    case 2: return &var_DATA;
  }
  return nullptr;
}

CEventConnection *FORTE_MQTT_SUBSCRIBE::getEOConUnchecked(const
TPortId paIndex) {
  switch(paIndex) {
    case 0: return &conn_INITO;
    case 1: return &conn_CNF;
  }
  return nullptr;
```

```
}

CDataConnection **FORTE_MQTT_SUBSCRIBE::getDIConUnchecked(const
TPortId paIndex) {
  switch(paIndex) {
    case 0: return &conn_QI;
    case 1: return &conn_BROKER;
    case 2: return &conn_USERNAME;
    case 3: return &conn_KEY;
    case 4: return &conn_TOPIC;
    case 5: return &conn_QOS;
    case 6: return &conn_RTN;
  }
  return nullptr;
}

CDataConnection *FORTE_MQTT_SUBSCRIBE::getDOConUnchecked(const
TPortId paIndex) {
  switch(paIndex) {
    case 0: return &conn_QO;
    case 1: return &conn_STATUS;
    case 2: return &conn_DATA;
  }
  return nullptr;
}
```

APÊNDICE J – LÓGICA AUXILIAR PARA OS SFIB MQTT

```
#include <stdio.h>
#include "mqtt_main.h"
#include "nvs_flash.h"
#include "esp_log.h"
#include "mqtt_client.h"

static const char *TAG = "MQTT_MAIN";
esp_mqtt_client_handle_t client;


static void mqtt_event_handler(void *handler_args, esp_event_base_t base, int32_t
event_id, void *event_data)
{
    esp_mqtt_event_handle_t event = event_data;
    client = event->client;
    switch ((esp_mqtt_event_id_t)event_id) {
    case MQTT_EVENT_CONNECTED:
        ESP_LOGI(TAG,"MQTT_EVENT_CONNECTED");
        ESP_LOGI(TAG,"sent publish successful");
        break;
    case MQTT_EVENT_DISCONNECTED:
         ESP_LOGI(TAG,"MQTT_EVENT_DISCONNECTED");
        break;

    case MQTT_EVENT_SUBSCRIBED:
         ESP_LOGI(TAG,"MQTT_EVENT_SUBSCRIBED");
        break;
    case MQTT_EVENT_UNSUBSCRIBED:
         ESP_LOGI(TAG,"MQTT_EVENT_UNSUBSCRIBED");
        break;
    case MQTT_EVENT_PUBLISHED:
         ESP_LOGI(TAG,"MQTT_EVENT_PUBLISHED");
        break;
    case MQTT_EVENT_DATA:
        ESP_LOGI(TAG,"MQTT_EVENT_DATA");

        printf("TOPIC=%.*s\r\n", event->topic_len, event->topic);
        printf("DATA=%.*s\r\n", event->data_len, event->data);

        snprintf(data_buffer, sizeof(topic_buffer), "%.*s", event->data_len, event->data);
        snprintf(topic_buffer, sizeof(topic_buffer), "%.*s", event->topic_len, event-
>topic);

        break;

    case MQTT_EVENT_ERROR:
         ESP_LOGI(TAG,"MQTT_EVENT_ERROR");
```

```
        if (event->error_handle->error_type ==
MQTT_ERROR_TYPE_TCP_TRANSPORT) {
            ESP_LOGI(TAG,"Last error code reported from esp-tls: 0x%x", event-
>error_handle->esp_tls_last_esp_err);
            ESP_LOGI(TAG,"Last tls stack error number: 0x%x", event->error_handle-
>esp_tls_stack_err);
            ESP_LOGI(TAG,"Last captured errno : %d (%s)",  event->error_handle-
>esp_transport_sock_errno,
                     strerror(event->error_handle->esp_transport_sock_errno));
        } else if (event->error_handle->error_type ==
MQTT_ERROR_TYPE_CONNECTION_REFUSED) {
            ESP_LOGI(TAG,"Connection refused error: 0x%x", event->error_handle-
>connect_return_code);
        } else {
            ESP_LOGI(TAG,"Unknown error type: 0x%x", event->error_handle-
>error_type);
        }
        break;
    default:
         ESP_LOGI(TAG,"Other event id:%d", event->event_id);
        break;
    }
}
void mqtt_app_start(const char *uri, const char *user, const char *pass)
{
   const esp_mqtt_client_config_t mqtt_cfg = {
        .broker = {
            .address.uri =
uri,//"mqtt://FilipeGSZ:aio_ZDmL257TLgO3xWxJcP2QC1jCulXB@io.adafruit.com"
            },
        .credentials = {
                    .username = user, //"FilipeGSZ",
                    .authentication = {
                        .password = pass,//"aio_ZDmL257TLgO3xWxJcP2QC1jCulXB"},
                        }
        }
    };

    client = esp_mqtt_client_init(&mqtt_cfg);
    /* The last argument may be used to pass data to the event handler, in this
example mqtt_event_handler */
    esp_mqtt_client_register_event(client, MQTT_EVENT_ANY,
mqtt_event_handler,NULL);
    esp_mqtt_client_start(client);
}

void mqtt_subscriber(const char* topic , int qos)
{
    int result;
    result = esp_mqtt_client_subscribe(client,topic,qos);
```

```
}

char* mqtt_publisher(const char* topic ,const char* data, int qos, int retain)
{
    int result;
    result = esp_mqtt_client_publish(client, topic, data, 0, qos, retain);

    switch (result)
    {
    case -2:
        return "Full OutBox";
    case -1:
        return "Failure";
    default:
        return "Success";
    }
}
```