Trabalho de Conclusão de Curso

Jigas de teste em protótipos de placas de circuito impresso: Estudo de método e desenvolvimento de uma IDE

de **Arquimedes Vinícius Pereira de França Moura**

orientado por
Prof. Dr. Erick de Andrade Barboza

Universidade Federal de Alagoas
Instituto de Computação
Maceió, Alagoas
5 de Dezembro de 2024

UNIVERSIDADE FEDERAL DE ALAGOAS
Instituto de Computação

JIGAS DE TESTE EM PROTÓTIPOS DE PLACAS DE CIRCUITO IMPRESSO: ESTUDO DE MÉTODO E DESENVOLVIMENTO DE UMA IDE

Trabalho de Conclusão de Curso submetido ao Instituto de Computação da Universidade Federal de Alagoas como requisito parcial para a obtenção do grau de Cientista da Computação.

Arquimedes Vinícius Pereira de França Moura

Orientador: Prof. Dr. Erick de Andrade Barboza

Banca Avaliadora:

Rodrigo José Sarmento Peixoto	Prof. Me., IC-UFAL
Vergilio Torezan Silingardi Del Claro	Prof. Dr., DEM-UFBA

Maceió, Alagoas
5 de Dezembro de 2024

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecária: Girlaine da Silva Santos – CRB-4 – 1127

M929j Moura, Arquimedes Vinícius Pereira de França.
Jigas de teste em protótipos de placas de circuito impresso: estudo de método e desenvolvimento de uma IDE / Arquimedes Vinícius Pereira de França Moura. – 2025.
44 f.: il. color.

Orientador: Erick de Andrade Barboza.
Monografia (Trabalho de Conclusão de Curso em Computação) - Universidade Federal de Alagoas, Instituto de Computação. Maceió, 2025.

Bibliografia: f. 42- 44.

1. Jiga de teste. 2. Placas de circuito impresso. 3. Validação de hardware. 4. Automação de testes. I. Título.

CDU: 004.5

UNIVERSIDADE FEDERAL DE ALAGOAS
Instituto de Computação

JIGAS DE TESTE EM PROTÓTIPOS DE PLACAS DE CIRCUITO IMPRESSO: ESTUDO DE MÉTODO E DESENVOLVIMENTO DE UMA IDE

Trabalho de Conclusão de Curso submetido ao Instituto de Computação da Universidade Federal de Alagoas como requisito parcial para a obtenção do grau de Cientista da Computação.

Aprovado em 5 de Dezembro de 2024:

gov.br
Documento assinado digitalmente
ERICK DE ANDRADE BARBOZA
Data: 09/12/2024 13:02:11-0300
Verifique em https://validar.iti.gov.br

Erick de Andrade Barboza,
Prof. Dr., Orientador

gov.br
Documento assinado digitalmente
RODRIGO JOSE SARMENTO PEIXOTO
Data: 08/01/2025 13:58:43-0300
Verifique em https://validar.iti.gov.br

Rodrigo José Sarmento Peixoto,
Prof. Me., IC-UFAL

gov.br
Documento assinado digitalmente
VERGILIO TOREZAN SILINGARDI DEL CLARO
Data: 22/12/2024 17:22:30-0300
Verifique em https://validar.iti.gov.br

Vergilio Torezan Silingardi Del Claro,
Prof. Dr., DEM-UFBA

Dedicatória

À minha família

A engenharia é a arte de moldar o físico, mas são os testes que moldam nossa confiança no que criamos.

Bill Nye, Cientista e Engenheiro

Resumo

Este trabalho apresenta um estudo sobre jigas de teste aplicada a placas de circuito impresso (PCB) e propõe uma prova de conceito integrada que visa facilitar e simplificar o desenvolvimento de cadernos de teste através de uma IDE. São consideradas as PCB que possuem pontos de teste em sua superfície superior e/ou inferior. Jigas de teste são essenciais para a verificação e validação de circuitos eletrônicos, garantindo que os componentes funcionem conforme especificações técnicas. A IDE desenvolvida busca facilitar a validação de circuitos, bem como depurações de lógica de programa, a partir da variação de nível de sinais digitais nos pontos de teste de uma PCB. Através do estudo das soluções atuais, são avaliadas as principais funcionalidades e a viabilidade técnica da solução proposta. Os resultados demonstram que a IDE desenvolvida tem o potencial de facilitar a implementação de testes de hardware em protótipos, através da adição de uma camada de abstração entre a jiga física e a jiga lógica, reduzindo o tempo de desenvolvimento da mesma, destacando-se como um complemento para o setor de validação de hardware.

***Palavras-chave*: *Jiga de teste*; *Placas de circuito impresso*; *Automação de testes*; *IDE*; *Validação de hardware*.**

Abstract

This work presents a study on test jigs applied to printed circuit boards (PCBs) and proposes an integrated proof of concept aimed at facilitating and simplifying the development of test plans through an IDE. The focus is on PCBs with test points on their upper and/or lower surfaces. Test jigs are essential for the verification and validation of electronic circuits, ensuring that components function according to technical specifications. The developed IDE seeks to streamline circuit validation and program logic debugging by analyzing variations in digital signal levels at the PCB test points. Through the study of current solutions, the main functionalities and technical feasibility of the proposed solution are evaluated. The results demonstrate that the developed IDE has the potential to simplify the implementation of hardware tests on prototypes by adding a layer of abstraction between the physical jig and the logical jig, reducing its development time and standing out as a valuable complement to the hardware validation sector.

Keywords*: *Test Jigs*; *Printed Circuit Board*; *Test Automation*; *IDE*; *Hardware validation.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

PCB *Prited Circuit Board*

ICT *In-Circuit-Test*

FT *Functional Test*

AXI *Automated X-ray Inspection*

IDE *Integrated Development Environment*

GPIO *General Purpose Input Output*

GPIOD *General Purpose Input Output Daemon*

SSL *Secure Sockets Layer*

USB *Universal Serial Bus*

USBIP *Universal Serial Bus Internet Protocol*

GUI *Graphical User Interface*

TDR *Time-domain reflectometer*

MVC *Model View Controller*

BGA *Ball Grid Array*

QFN *Quad Flat No-leads*

ATE *Automated Test Equipment*

Sumário

Capítulo 1

Introdução

No cenário atual da indústria eletrônica, o desenvolvimento e a validação de *Placas de Circuito Impresso* (PCB) são etapas cruciais para garantir a confiabilidade e a funcionalidade de dispositivos eletrônicos. A crescente complexidade dos circuitos e a demanda por produtos de alta qualidade intensificam a necessidade de métodos eficientes e precisos de teste[Todorov and Asparuhova, 2023]. Ao se considerar que estratégia de teste seguir, nos deparamos com uma gama de tecnologias, cada uma com suas coberturas de falhas e características de desempenho[Verma, 2002], este trabalho foca nos sistemas de teste ICT (in-circuit-test) e FT (functional test), portanto não cobre todo o processo de teste de uma PCB, mais especificamente, não cobre a parte de inspeção de solda, na qual seria necessário a adição de outras tecnologias, como por exemplo a AXI (Automated X-ray Inspection)[Verma, 2002].

A jiga de teste, equipada com uma cama de pregos, consiste em uma matriz de pinos metálicos alinhados precisamente com pontos de teste ou pads expostos na superfície de uma PCB[Vempati et al., 2008]. Essa configuração permite a aplicação de sinais elétricos e a leitura de respostas em tempo real, essencial para a realização de ICTs e FTs. O ICT é um método que verifica a integridade elétrica dos componentes individuais e das conexões na placa, assegurando que resistores, capacitores, circuitos integrados e outros componentes estejam operando dentro de suas especificações. Já o FT avalia o desempenho operacional da PCB como um todo, simulando condições reais de uso e verificando se a placa executa as funções para as quais foi projetada.

A elaboração de cadernos de teste é uma etapa crítica nesse processo, pois define a sequência de procedimentos a serem executados durante a fase de validação. Esses cadernos detalham os parâmetros de teste, os critérios de aprovação e reprovação, e as ações corretivas em caso de detecção de falhas. A precisão e a abrangência dos cadernos de teste impactam diretamente na eficácia dos testes realizados, contribuindo para a melhoria contínua dos processos produtivos e para a redução de custos associados a retrabalhos e garantias.

Entretanto, a elaboração de procedimentos de teste e a interação com o hardware cor-

respondente podem se tornar tarefas complexas e suscetíveis a erros. A falta de ferramentas integradas que facilitem a criação, gestão e execução desses procedimentos contribui para o aumento do tempo de desenvolvimento e para potenciais falhas no processo de validação.

Visando abordar esses desafios, este trabalho propõe o desenvolvimento de um *Ambiente de Desenvolvimento Integrado* (IDE) e um servidor, como prova de conceito, voltado para a facilitação dos ICTs e FTs de uma PCB, bem como a criação de cadernos de teste para protótipos de PCBs. Diferentemente de soluções que focam na parte física da jiga ou na conexão física entre a PCB e a cama de pregos, este projeto concentra-se na camada de software dos testes, o que permite, dada uma conexão estabelecida, controlar e monitorar sinais elétricos através de pontos de teste expostos em uma PCB através de um servidor remoto, embarcado em uma jiga de testes.

A IDE proposta permitirá que seus usuários acessem remotamente um servidor Linux, embarcado em qualquer hardware compatível que possua GPIOs acessíveis, facilitando o controle centralizado dos testes e a integração com o hardware de teste, bem como a adição de uma camada de abstração para quem irá realizar a criação do caderno de testes. Será possível visualizar os dispositivos conectados às portas USB do servidor, como gravadores de placa (por exemplo, JLink e ST-Link), e exportar essas portas para o computador local do usuário. Assim, será possível programar o firmware da PCB a ser testada, caso ela possua uma porta USB para gravação.

Além disso, a IDE permitirá visualizar e interagir com todos os GPIOs do dispositivo que executa o servidor Linux. Os GPIOs do servidor estarão conectados, através de uma cama de pregos aos pontos de teste da PCB em análise. Dessa forma, será possível realizar operações de leitura, escrita e detecção de transições de nível lógico, proporcionando um meio eficaz de testar o comportamento da PCB. Por fim, o programa permitirá a criação de cadernos de teste, definindo sequências de operações nos GPIOs do servidor. Isso possibilitará a realização de testes específicos na PCB, com operações condicionais e fluxos lógicos, o que por sua vez, podem servir para validar que seus componentes estejam funcionando conforme o especificado.

A comunicação entre a IDE e o servidor será realizada por meio de uma conexão socket, garantindo a integridade e a confiabilidade dos dados transmitidos. A IDE deverá oferecer uma interface gráfica amigável e responsiva, enquanto o servidor opera com uma interface de terminal eficiente e de baixo custo computacional, permitindo uma interação eficaz entre os sistemas.

1.1 Justificativa

A evolução constante dos dispositivos eletrônicos tem levado a um aumento na complexidade dos circuitos e, consequentemente, na necessidade de métodos de teste mais robustos

e eficientes, porém, os métodos de teste se concentram mais na camada de produção em massa, inviabilizando boa parte destes para teste de protótipos e produção em pequenas quantidades. A garantia da qualidade e funcionalidade das PCBs é essencial para o sucesso de produtos eletrônicos em um mercado altamente competitivo. No entanto, a falta de ferramentas que integrem de forma simplificada o controle de hardware e a criação de procedimentos de teste representa um desafio significativo para engenheiros e técnicos da área[Drew et al., 2016].

Este projeto busca criar uma base a partir da qual será possível suprir a falta de ferramentas integradas que simplifiquem o processo de teste de protótipos de PCBs, potencialmente reduzindo o tempo de desenvolvimento e minimizando erros. Ao abstrair a complexidade inerente à interação com hardwares de teste e fornecer uma plataforma simples para a criação de cadernos de teste, espera-se que a IDE contribua para a eficiência dos processos de validação de protótipos de PCBs.

A implementação de uma IDE especializada poderá proporcionar uma plataforma que unifica o gerenciamento de dispositivos de teste, o controle de GPIOs e a elaboração de cadernos de teste personalizados. Ao facilitar esses processos, a IDE contribui para a redução de custos, otimização do tempo de desenvolvimento e aumento da confiabilidade dos testes realizados.

Além disso, a adoção de tecnologias seguras de comunicação como SSL e a utilização de ferramentas de software consolidadas, como USBIP e GPIOD, reforçam a relevância e a viabilidade técnica do projeto, alinhando-se às práticas gerais da indústria.

1.2 Objetivos

1.2.1 Objetivos Gerais

Desenvolver uma prova de conceito na forma de uma IDE e um servidor Linux que facilite a depuração de protótipos de PCBs e a criação e execução de cadernos de teste para jigas de PCBs.

1.2.2 Objetivos Específicos

- Implementar um servidor Linux, que possa ser embarcado em uma jiga de testes.
- Implementar o acesso remoto ao servidor Linux através de uma IDE.
- Implementar a funcionalidade de compartilhamento de portas USB, do servidor para uma IDE.
- Desenvolver funcionalidades na IDE para interagir com os GPIOs do servidor, podendo realizar leituras e escritas nos mesmos.

- Desenvolver uma interface "prova de conceito"na IDE para a elaboração de cadernos de teste, permitindo a definição de sequencias lógicas de operações em GPIOs.

1.3 Organização do Trabalho

Este estudo está organizado em capítulos que descrevem cada etapa do desenvolvimento do trabalho, fornecendo detalhes teóricos e aplicados da implementação da solução. O Capítulo 2 apresenta os conceitos teóricos e as tecnologias relevantes para o projeto, incluindo detalhes sobre PCBs, jigas de teste, GPIOs, comunicação via SOCKET e SSL, além das bibliotecas e ferramentas utilizadas. O Capítulo 3 descreve a abordagem adotada para o desenvolvimento da IDE, incluindo a arquitetura do sistema, a implementação das funcionalidades e os critérios de validação. O Capítulo 4 detalha o processo de implementação da IDE, os desafios enfrentados e os resultados obtidos, incluindo capturas de tela e exemplos de cadernos de teste criados. Por fim, são apresentadas as conclusões do trabalho, ressaltando as contribuições e o impacto da IDE desenvolvida, além de sugestões para melhorias e expansões futuras.

Capítulo 2

Fundamentação Teórica

Este capítulo define conceitos básicos sobre jigas de testes, sistemas embarcados, arquitetura de comunicação entre aplicações e os programas que foram utilizados no desenvolvimento de uma IDE, enfatizando suas aplicações em placas de circuito impresso.

2.1 Testes de Placas de Circuito Impresso

A produção de dispositivos eletrônicos envolve etapas críticas de teste e validação das placas de circuito impresso (PCB) para garantir o funcionamento correto e a qualidade dos produtos finais. Os testes em PCBs visam detectar defeitos de fabricação, falhas de componentes e erros de montagem que possam comprometer o desempenho do dispositivo [Geier, 2015].

Os Métodos de teste podem ser categorizados em testes funcionais e estruturais. Os FT verificam o comportamento operacional da placa, enquanto os testes estruturais avaliam a integridade física e elétrica dos componentes e conexões. A eficiência nesses processos é fundamental para reduzir custos e tempo de produção, bem como para assegurar a confiabilidade dos produtos [Nayak et al., 2017].

2.1.1 Testes Estruturais

Teste de Continuidade e Isolamento

O teste de continuidade verifica se todas as conexões elétricas estão corretamente estabelecidas entre os pontos designados. Já o teste de isolamento assegura que não há conexões indesejadas (curtos-circuitos) entre trilhas que devem estar isoladas. Equipamentos de teste de circuito aberto e curto-circuito são usados para medir a resistência entre pontos específicos da PCB, garantindo que os valores estejam dentro das tolerâncias aceitáveis.

Teste In-Circuit (ICT)

O teste in-circuit (ICT) é realizado para verificar individualmente cada componente montado na PCB. Utiliza-se equipamentos automatizados que aplicam estímulos elétricos diretamente nos componentes e medem as respostas. O ICT permite detectar componentes defeituosos, valores incorretos, polaridades invertidas e outros problemas que podem surgir durante o processo de montagem [Buckroyd, 2013].

As tecnologias empregadas no ICT incluem a utilização de fixadores com "cama de pregos"(bed-of-nails), que são dispositivos mecânicos que facilitam o acesso elétrico aos pontos de teste da PCB. Sistemas ICT modernos também podem incorporar recursos de boundary scan (IEEE 1149.1 JTAG) para testar componentes com acesso físico limitado [Cheung, 2010]. Na Figura 2.1 podemos observar um exemplo de linha de teste ICT, semi-automático, onde é necessário que um operador realize a fixação de modo manual de uma ou mais PCBs em uma jiga de testes, e execute manualmente seu caderno de teste, sendo necessário o acompanhamento dos resultados por parte do operador da jiga.

Adicionalmente, a adoção de sistemas ICT automatizados tem contribuído significativamente para a melhoria da eficiência na detecção de falhas em processos de fabricação de PCBs. A integração de técnicas de automação e controle permite a execução de testes com maior rapidez e precisão, reduzindo a dependência da intervenção humana e minimizando a possibilidade de erros operacionais. A evolução dos sistemas ICT, aliada à implementação de protocolos de comunicação mais rápidos e eficientes, tem impulsionado a capacidade das indústrias em atender às demandas por produtos eletrônicos de alta qualidade e confiabilidade [Singh et al., 2024].

Figura 2.1: Exemplo de linha de teste de PCB, utilizando tecnologias ICT".[1]

Teste de Impedância Controlada

Em PCBs de alta velocidade, o controle de impedância é crítico para garantir a integridade dos sinais. O teste de impedância controlada mede as características de impedância das trilhas para assegurar que elas correspondem aos valores projetados. Equipamentos especializados, como reflectômetros no domínio do tempo (TDR), são utilizados para esse fim [Johnson, 2003].

Um TDR, mede as reflexões que ocorrem ao longo de um condutor. Para conseguir medir estas reflexões, o TDR irá transmitir um sinal incidente ao condutor e ouvirá suas reflexões. Caso o condutor tenha impedância uniforme e esteja adequadamente terminado, não haverá reflexões ao longo do condutor e o sinal incidente será absorvido na extremidade oposta pela terminação do condutor. Caso contrário, o sinal incidente será parcialmente refletido de volta para sua fonte. Podemos observar na Figura 2.2 um exemplo de dispositivo TDR.

[1]Fonte: Site da fabricante GREATPCB. Disponível em: `https://greatpcb.com/pt/technical/assembly-testing` Acesso em 01 de dez. 2024

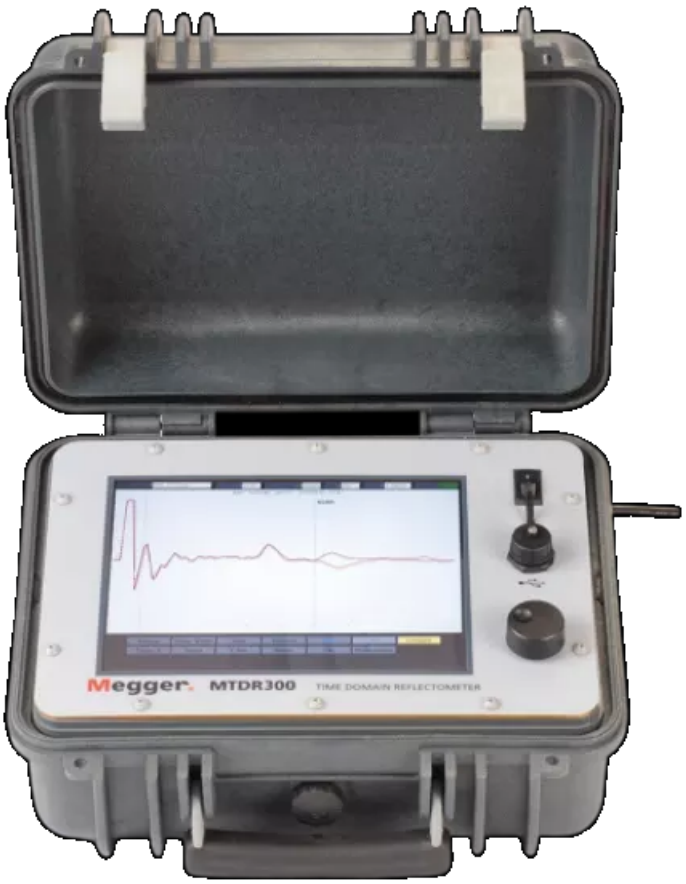


Figura 2.2: Exemplo de reflectrômetro no domínio do tempo para detecção de falhas em cabos".[2]

Inspeção Óptica Automatizada (AOI)

A AOI utiliza câmeras de alta resolução e algoritmos de processamento de imagem para inspecionar automaticamente as PCBs em busca de defeitos como componentes ausentes, desalinhados, soldas defeituosas e polaridades incorretas. O sistema compara as imagens capturadas com um modelo de referência para detectar discrepâncias [Taha et al., 2014].

A eficácia da AOI na detecção de defeitos depende significativamente da qualidade das imagens adquiridas e da precisão dos algoritmos de processamento utilizados [Taha et al., 2014]. Avanços recentes em técnicas de visão computacional têm permitido o desenvolvimento de sistemas AOI mais sofisticados, capazes de identificar uma variedade maior de defeitos com maior precisão. Além disso, a integração de algoritmos de

[2]Fonte: Site da Fabricante Megger. Disponível em: `https://www.megger.com/en-us/products/mtdr300` Acesso em 05 de dez. 2024

inteligência artificial e aprendizado de máquina permite que os sistemas AOI se adaptem a novas condições de produção e a variações nos componentes, melhorando continuamente seu desempenho.

Os sistemas AOI modernos também são capazes de realizar inspeções em alta velocidade, compatíveis com as taxas de produção das linhas de montagem automatizadas. A implementação da AOI não apenas reduz o tempo necessário para inspeções manuais, mas também minimiza erros humanos, aumentando a eficiência geral do processo de fabricação[Taha et al., 2014]. Ademais, a AOI pode ser integrada a sistemas de controle de qualidade estatísticos, fornecendo dados valiosos para a análise de tendências de defeitos e para a otimização contínua dos processos de produção. A Figura 2.3 mostra um exemplo de sistema AOI em execução.

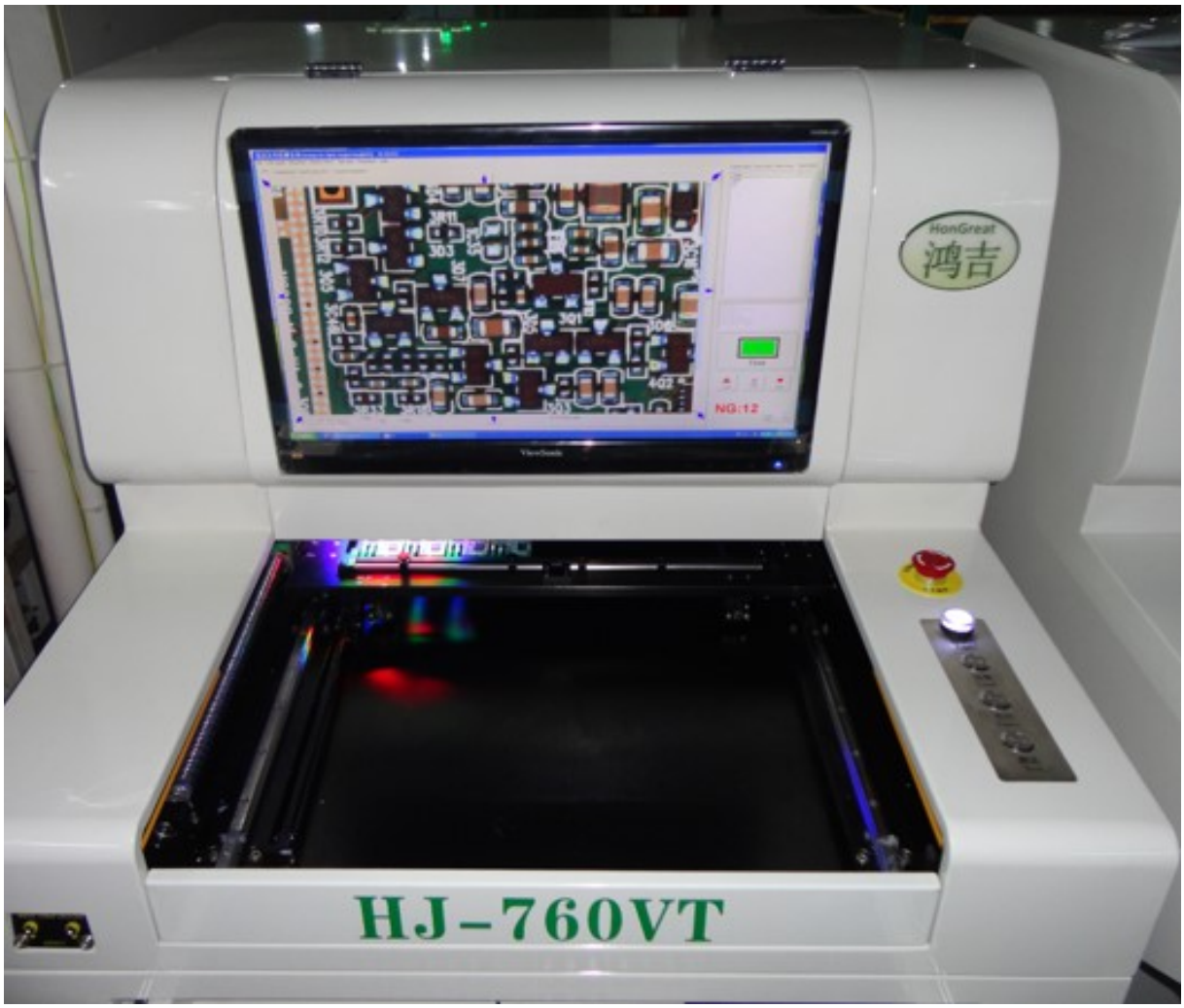


Figura 2.3: Exemplo de sistema AOI.[3]

[3]Fonte: Site da Fabricante AllPoint. Disponível em: `https://www.allpoint.com.br/aoi-automatic-optical-inspection-hj-760.html` Acesso em 01 de dez. 2024

Inspeção por Raios X

A inspeção por raios X é uma técnica utilizada para examinar componentes e soldas que não são visíveis externamente, como aquelas presentes em componentes com terminais ocultos, tais como BGA (Ball Grid Array) e QFN (Quad Flat No-leads). Esta metodologia permite a visualização interna das estruturas da PCB, possibilitando a detecção de defeitos internos, inclusões, vazios (voids), bolhas e outros problemas que não são detectáveis por inspeção visual convencional [Liu et al., 2016].

Ao penetrar através dos materiais da PCB e dos componentes montados, os raios X produzem imagens que revelam irregularidades na densidade e na integridade das conexões de solda e dos próprios componentes. Isso é crucial para identificar defeitos como juntas de solda frias, excesso ou falta de solda, deslocamento de componentes e defeitos internos nos semicondutores. A precisão e a confiabilidade da inspeção por raios X tornam-na uma ferramenta indispensável na garantia da qualidade em processos de fabricação de PCBs com alta densidade de componentes [Singh et al., 2024].

Além disso, com o avanço tecnológico, sistemas de inspeção por raios X têm se tornado cada vez mais sofisticados, incorporando recursos como tomografia computadorizada em 3D, que permite uma análise volumétrica das estruturas internas da PCB. Isso possibilita a detecção e a caracterização mais detalhada de defeitos, contribuindo para a melhoria contínua dos processos de fabricação e para a redução de custos associados a falhas em campo [Taha et al., 2014]. A integração da inspeção por raios X nos fluxos de produção é, portanto, essencial para atender aos rigorosos padrões de qualidade exigidos pela indústria eletrônica moderna. A Figura 2.4 mostra uma imagem de raio X de uma PCB em ambiente de testes.

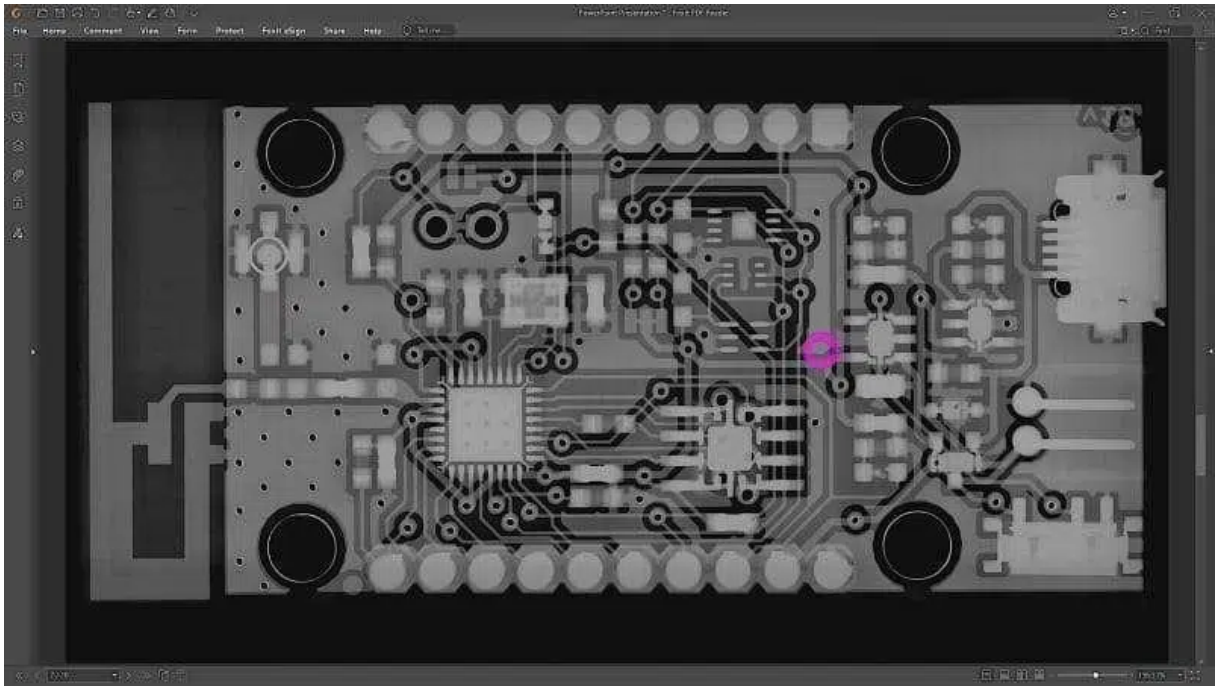


Figura 2.4: Imagem de inspeção de raio X de uma PCB.[4]

2.1.2 Testes Funcionais

Os testes funcionais são essenciais para avaliar o desempenho operacional global da PCB sob condições reais ou simuladas de uso. Esses testes asseguram que a placa não apenas esteja eletricamente correta em termos de conexões e componentes, mas também execute todas as funções para as quais foi projetada, atendendo aos requisitos especificados no projeto.

A execução dos testes funcionais envolve a aplicação de sinais de entrada específicos e a observação detalhada das respostas de saída da PCB. Esse processo pode incluir a verificação de protocolos de comunicação, processamento de dados, interfaces de usuário e interação com outros sistemas ou componentes externos. O objetivo é reproduzir o ambiente operacional ao qual a PCB será submetida em sua aplicação final, permitindo a identificação de falhas de funcionalidade que não seriam detectadas por testes elétricos ou estruturais [Buckroyd, 2013].

Para aumentar a eficiência e a precisão dos testes funcionais, são frequentemente utilizados Sistemas de Teste Automatizados (ATE - *Automated Test Equipment*). Esses sistemas são capazes de executar sequências de teste predefinidas com alta repetibilidade, utilizando instrumentação avançada que pode simular uma variedade de condições operacionais, tais como variações de tensão, frequência, temperatura e carga. Os ATEs permitem a integração de múltiplos instrumentos de medição e controle, facilitando a

[4]Fonte: Site da Fabricante Venture-MFG. Disponível em: `https://www.venture-mfg.com/principles-of-pcb-x-ray-inspection/` Acesso em 02 de dez. 2024

coleta e análise de dados em tempo real. A Figura 2.5 ilustra um exemplo de teste funcional sendo realizado em uma PCB, demonstrando a complexidade e abrangência das configurações de teste empregadas [Moganti and Ercal, 1995].

O desenvolvimento de procedimentos de teste funcional robustos requer uma compreensão aprofundada do design da PCB, incluindo tanto o hardware quanto o software embarcado. É fundamental que os procedimentos de teste sejam cuidadosamente planejados e documentados, assegurando que todas as funcionalidades críticas sejam verificadas e que os resultados sejam rastreáveis. A implementação de testes funcionais contribui significativamente para a garantia da qualidade e confiabilidade do produto final, reduzindo custos associados a falhas em campo e aumentando a satisfação do cliente [Blanchard and Fabrycky, 2010].

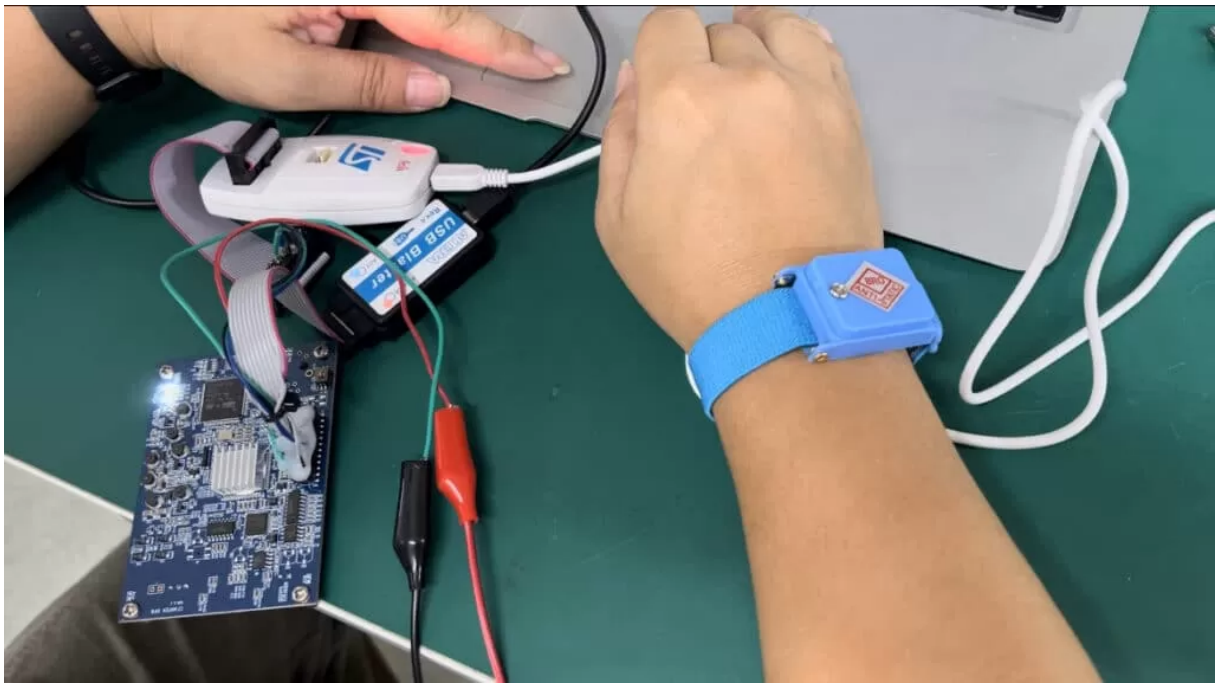


Figura 2.5: Engenheiro conduzindo um teste funcional de firmware em uma PCB.[5]

2.2 Jigas de Teste e a Técnica *Cama de Pregos*

As Jigas de teste são dispositivos que facilitam o processo de verificação das PCBs, permitindo acesso simultâneo a múltiplos pontos de teste [Velasco, 2023]. A técnica conhecida como "cama de pregos"(bed of nails) utiliza uma matriz de pinos de contato que se alinham com pontos de teste específicos em uma PCB, estabelecendo as conexões elétricas necessárias para a execução de testes automatizados [Groover, 2015].

[5]Fonte: Site ROWSUM. Disponível em: `https://www.rowsum.com/pcb-functional-testing-product-reliability/` Acesso em 01 de dez. 2024

Essa abordagem é amplamente adotada na indústria eletrônica devido à sua capacidade de acelerar o processo de teste e aumentar a precisão na detecção de falhas. A integração com sistemas computacionais permite a automatização dos testes, melhorando a eficiência e reduzindo a dependência de intervenção humana. Na Figura 2.6 podemos observar um exemplo de jiga de teste que utiliza a técnica de cama de pregos.

O uso da cama de pregos possibilita a realização de testes elétricos e funcionais de maneira abrangente, permitindo a detecção de defeitos não apenas na montagem física da PCB, mas também em sua funcionalidade operacional. A precisão mecânica e a confiabilidade dos contatos elétricos dos pinos são essenciais para garantir resultados consistentes e minimizar a ocorrência de falsos positivos ou negativos durante o processo de teste. Além disso, o design modular das jigas de teste facilita a adaptação a diferentes modelos de PCBs, tornando o processo de teste mais flexível e econômico.

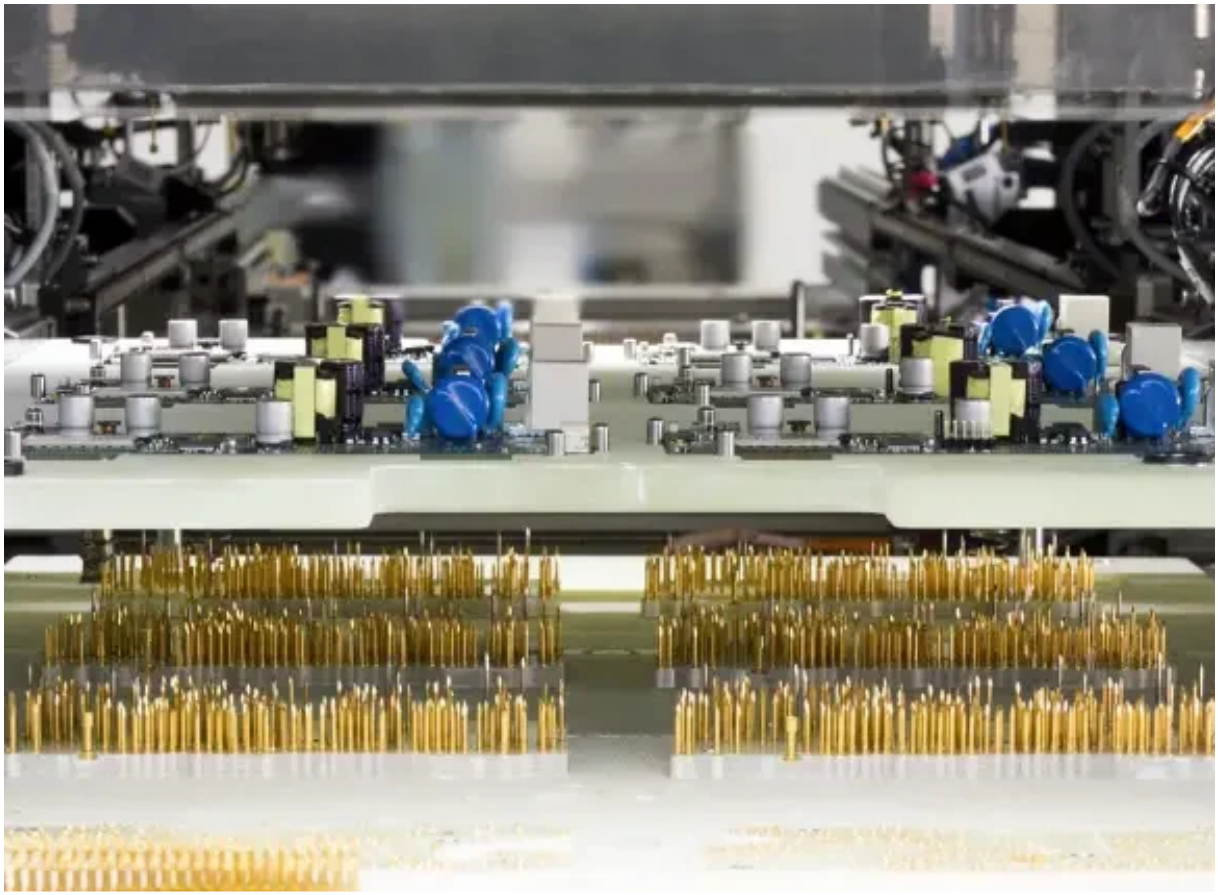


Figura 2.6: Exemplo de jiga de testes usando a técnica de "Cama de Pregos".[6]

[6]Fonte: Site da ATS Engineering Ltd. Disponível em: `https://www.ats-eng.com/bed-of-nails-ict/` Acesso em 29 de nov. 2024

2.3 Sistemas Linux Embarcados e Raspberry Pi

Os sistemas Linux embarcados são versões customizadas do sistema operacional GNU/LINUX projetadas para operar em dispositivos com recursos computacionais limitados, atendendo às restrições de memória, processamento e consumo de energia típicas de sistemas embarcados[Streif, 2016]. Esses sistemas oferecem flexibilidade e o vasto ecossistema de software do Linux, adaptados para plataformas de hardware específicas, permitindo o desenvolvimento de soluções eficientes e escaláveis em diversas áreas da tecnologia.

A Raspberry Pi é um exemplo notável de plataforma de hardware que utiliza Linux embarcado, oferecendo uma solução de baixo custo e alto desempenho para uma ampla gama de aplicações, incluindo automação, controle industrial, educação e prototipagem rápida[Wolfram Donat, 2021]. Dotada de um processador ARM, interfaces de comunicação variadas (Ex GPIO, I^2C, SPI, UART, USB e Ethernet) e suporte a sistemas operacionais baseados em Linux, a Raspberry Pi tornou-se uma ferramenta útil para engenheiros, pesquisadores e entusiastas em todo o mundo.

A utilização de sistemas Linux embarcados em aplicações de teste de PCBs permite a execução de scripts e programas complexos, bem como o uso de ferramentas avançadas de desenvolvimento e depuração, facilitando a integração com outros sistemas através de redes e interfaces padrões[Wolfram Donat, 2021]. A capacidade de executar softwares personalizados, aliada ao acesso direto aos recursos de hardware, torna essas plataformas ideais para implementar soluções de teste automatizados, capazes de interagir diretamente com os circuitos sob teste. Além disso, a comunidade ativa em torno da Raspberry Pi contribui com inúmeros recursos, documentações e projetos de código aberto.

2.4 GPIOs e a Biblioteca libgpiod

Os GPIOs são pinos configuráveis presentes em microcontroladores e microprocessadores que podem ser programados como entradas ou saídas digitais. Eles são fundamentais em aplicações que requerem interação direta com hardware externo, permitindo o controle de dispositivos e a leitura de sinais.

A biblioteca *libgpiod*[7] fornece uma interface moderna para o subsistema de GPIO do Linux, substituindo a antiga interface baseada em *sysfs*. Ela permite o acesso aos GPIOs através de uma API consistente e eficiente, suportando operações de leitura, escrita e detecção de eventos de borda (subida e descida de borda). Essa biblioteca é essencial para aplicações que requerem alto desempenho e acesso confiável aos GPIOs de um sistema Linux.

A adoção da *libgpiod* promove um modelo de programação mais seguro e eficiente para a manipulação de GPIOs, expondo funcionalidades avançadas do subsistema de GPIO do

[7]Disponível em `https://git.kernel.org/pub/scm/libs/libgpiod/libgpiod.git`

Linux de forma unificada. A biblioteca suporta a operação simultânea em múltiplas linhas de GPIO e oferece mecanismos para tratamento de eventos com baixa latência, o que é crucial em sistemas embarcados que exigem respostas em tempo real. Além disso, a *libgpiod* facilita a portabilidade de aplicações entre diferentes plataformas de hardware que executam o kernel Linux, pois abstrai detalhes específicos do hardware subjacente. Conforme destacado por Golaszewski [Golaszewski, 2022], o advento da *libgpiod* representa um avanço significativo na interface de GPIO do Linux, atendendo às demandas atuais por interfaces de programação mais robustas e flexíveis.

2.5 Arquitetura Cliente-Servidor e Comunicação Segura

A arquitetura client-servidor é um modelo de design de software que separa as responsabilidades entre os fornecedores de recursos ou serviços (servidores) e solicitantes desses recursos (clientes) [Tanenbaum and Wetherall, 2010]. Essa arquitetura é amplamente utilizada em redes de computadores e sistemas distribuídos devido à sua escalabilidade e modularidade.

Para garantir a segurança na comunicação entre cliente e servidor, protocolos como SSL e TLS são empregados. Eles fornecem criptografia e autenticação, assegurando que os dados transmitidos não sejam interceptados ou alterados por terceiros. A utilização de sockets seguros é fundamental em aplicações que envolvem o controle remoto de hardware, como no caso da IDE proposta, onde os comandos e informações sensíveis são transmitidos entre o cliente (IDE) e o servidor (dispositivo embarcado). [Stallings, 2016].

Além dos aspectos de criptografia e autenticação proporcionados pelo SSL e TLS, é crucial considerar a gestão adequada de certificados digitais e chaves criptográficas para manter a integridade do sistema de comunicação [Rescorla, 2000]. A implementação correta desses protocolos requer atenção às configurações de segurança, como a escolha de algoritmos criptográficos robustos, o uso de certificados emitidos por autoridades confiáveis e a atualização periódica dos componentes de software para proteger contra vulnerabilidades conhecidas. No contexto da IDE proposta, a adoção de práticas recomendadas de segurança cibernética é essencial para prevenir ataques como man-in-the-middle, negação de serviço e outras ameaças que possam comprometer o funcionamento seguro e confiável do sistema.

2.6 Desenvolvimento de IDEs com Qt e Python

O Qt[8] é um framework multiplataforma para o desenvolvimento de interfaces gráficas de usuário (GUI) que oferece uma ampla gama de componentes e ferramentas. [Blanchette and Summerfield, 2008]. Quando combinado com a linguagem Python[9], através dos bindings da biblioteca PySide6[10], permite o desenvolvimento rápido de aplicações com interfaces modernas e funcionais [Summerfield, 2007].

Python é uma linguagem de programação interpretada, de alto nível e com uma sintaxe clara e concisa, o que facilita a manutenção e a legibilidade do código [Lutz, 2013]. A combinação de Qt e Python é particularmente poderosa no desenvolvimento de IDEs, pois une a eficiência e a riqueza de recursos do Qt com a simplicidade e versatilidade do Python.

Além disso, a utilização de Python com Qt através do PySide6 proporciona aos desenvolvedores acesso a uma vasta biblioteca de módulos e pacotes que agilizam o processo de desenvolvimento. O PySide6 é a versão mais recente dos bindings oficiais do Qt para Python, garantindo compatibilidade com as funcionalidades modernas do Qt 6. Isso possibilita a criação de interfaces de usuário responsivas e adaptativas, integrando-se perfeitamente com outros componentes do sistema e facilitando a implementação de padrões de design como MVC (Model-View-Controller) [Frank Buschmann and Schmidt, 2007].

Ademais, o desenvolvimento de IDEs com Qt e Python beneficia-se da comunidade ativa e extensa documentação disponível para ambas as tecnologias. A natureza de código aberto do Python e a disponibilidade do Qt sob licenças open-source permitem a colaboração e o compartilhamento de soluções entre desenvolvedores, promovendo a inovação e a melhoria contínua das ferramentas desenvolvidas. Estudos têm demonstrado que o uso de Python no desenvolvimento de aplicações de interface gráfica pode reduzir significativamente o tempo de desenvolvimento e os custos associados, sem comprometer a performance ou a qualidade do software final [Lutz, 2013].

2.7 Gerenciamento de Dispositivos USB e Exportação

O gerenciamento de dispositivos USB em sistemas Linux é realizado através do subsistema USB do kernel, que fornece interfaces para detecção, configuração e comunicação com dispositivos conectados [Love, 2010]. Ferramentas como *udev*[11] e *usbutils*[12] permitem o monitoramento e a manipulação desses dispositivos em espaço de usuário.

[8]Disponível em `https://www.qt.io/`
[9]Disponível em `https://www.python.org/`
[10]Disponível em `https://pypi.org/project/PySide6`
[11]Disponível em `https://git.kernel.org/pub/scm/linux/hotplug/udev.git`
[12]Disponível em `http://www.linux-usb.org`

A exportação de dispositivos USB para outros sistemas pode ser realizada utilizando o protocolo USBIP [Community, 2024], que permite o compartilhamento de dispositivos USB através de redes TCPIP [Sunahara, 2005]. Essa tecnologia é útil em cenários onde é necessário acessar dispositivos USB conectados a um sistema remoto, como programadores de microcontroladores ou interfaces de depuração.

Adicionalmente, a utilização do USBIP em conjunto com tecnologias de virtualização e contêineres permite a criação de ambientes isolados nos quais dispositivos USB podem ser compartilhados entre diferentes máquinas virtuais ou contêineres, aumentando a flexibilidade e eficiência dos recursos computacionais [Sunahara, 2005]. No entanto, é fundamental considerar os aspectos de segurança associados à exportação de dispositivos USB pela rede, uma vez que a comunicação pode estar sujeita a interceptações ou acessos não autorizados. A implementação de medidas de segurança, como criptografia de dados ou utilização de redes virtuais privadas (VPNs), é recomendada para proteger a integridade das informações transmitidas e assegurar operações remotas confiáveis.

2.8 Interfaces de Terminal com Textualize

As interfaces de usuário baseadas em terminal continuam sendo relevantes, especialmente em ambientes com recursos limitados ou onde a simplicidade e eficiência são prioridades. Bibliotecas como *Textualize*[13] (e sua predecessora *Rich*) permitem a criação de interfaces de terminal sofisticadas em Python, incluindo elementos como cores, estilos, tabelas e layouts atualizados [McGugan, 2021].

Essas ferramentas facilitam o desenvolvimento de aplicações de linha de comando com interfaces amigáveis, melhorando a experiência do usuário sem a necessidade de uma interface gráfica completa. No contexto do servidor da IDE proposta, uma interface de terminal aprimorada permite monitorar e controlar o sistema de forma eficiente.

Adicionalmente, o uso do *Textualize* contribui para a redução da sobrecarga de recursos computacionais, uma vez que as interfaces de terminal geralmente consomem menos memória e processamento em comparação com interfaces gráficas tradicionais. Isso é particularmente vantajoso em ambientes onde o servidor opera em hardware com capacidades limitadas, como dispositivos embarcados ou sistemas remotos. Além disso, a biblioteca oferece suporte para a criação de interfaces responsivas e adaptativas, facilitando a interação do usuário através de conexões de rede com diferentes larguras de banda e latências. Dessa forma, a adoção do *Textualize* não apenas otimiza o desempenho do sistema, mas também amplia a acessibilidade e usabilidade da aplicação em diversos cenários operacionais.

[13] Disponível em `https://www.textualize.io`

Capítulo 3

Metodologia

Este capítulo descreve a metodologia no desenvolvimento da IDE para facilitação da criação de cadernos de teste para jigas de PCBs. A abordagem metodológica abrange a arquitetura do sistema, tanto no lado do cliente (IDE) quanto do lado do servidor, detalhando os padrões arquiteturais adotados, os componentes-chave, os fluxos de interação e as tecnologias utilizadas. Além disso, são discutidos os protocolos de comunicação e os mecanismos de segurança implementados.

3.1 Arquitetura da IDE

3.1.1 Padrão Arquitetural: Event-Driven e MVC

A IDE foi desenvolvida utilizando uma arquitetura mista que combina os padrões **Event-Driven** (orientada a eventos) e **Model-View-Controller** (MVC). O padrão padrão MVC é amplamente utilizado no desenvolvimento de interfaces gráficas por separar a aplicação em três componentes principais: modelo, visão e controlador [Gamma et al., 1994]. Já a arquitetura orientada a eventos é adequada para aplicações que requerem alta responsividade e interação assíncrona, onde as ações dos usuários ou eventos do sistema desencadeiam respostas específicas [Harel, 1987].

A combinação desses padrões permite uma estrutura modular e escalável, facilitando a manutenção e a expansão das funcionalidades da IDE. Os Widgets representam as Views no MVC, responsáveis pela interface gráfica e interação com o usuário. A lógica do aplicativo é implementada em Manipuladores, que desempenham o papel dos Controllers, enquanto os Models gerenciam os dados e estados da aplicação, a Figura 3.1 mostra de forma visual a arquitetura da IDE.

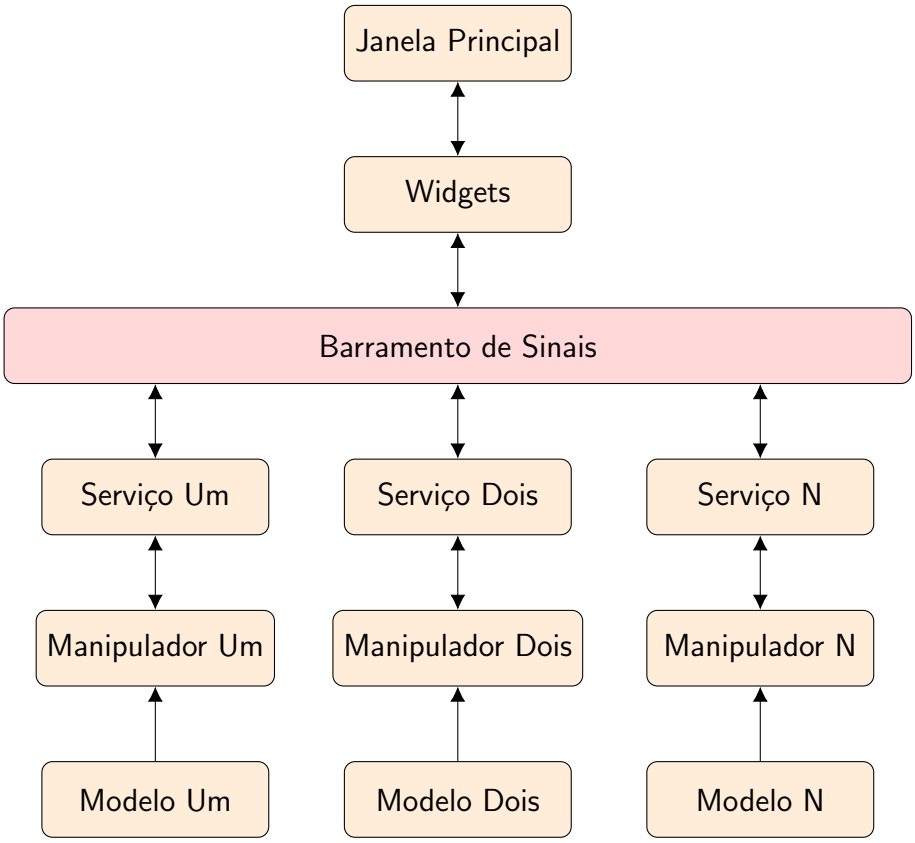


Figura 3.1: Representação da arquitetura de software da IDE proposta.

Fonte: Elaborado pelo autor, (2024)

3.1.2 Componentes da Interface de Usuário (Widgets)

Os **Widgets** são os componentes visuais fundamentais da interface gráfica da aplicação, desenvolvidos utilizando o framework **QT**[Blanchette and Summerfield, 2008]. Cada Widget é responsável por uma parte específica da interface do usuário, como botões, caixas de diálogo, tabelas e outros elementos interativos. Esses componentes são projetados para proporcionar uma experiência de usuário intuitiva e responsiva, facilitando a interação entre o usuário e o sistema. A modularidade dos Widgets permite a construção de interfaces complexas a partir de componentes menores e reutilizáveis, promovendo a consistência visual e a manutenção eficiente do código-fonte.

É importante destacar que os Widgets não devem incorporar lógica de negócios; sua função principal é capturar as ações do usuário e apresentar informações de forma clara e eficiente. Eles atuam como intermediários entre o usuário e a lógica subjacente da aplicação, delegando o processamento das ações para outros componentes responsáveis pela lógica de negócios, como os manipuladores ou handlers. Essa separação de responsabilidades está alinhada com os princípios do padrão arquitetural Model-View-Controler (MVC), garantindo uma maior organização e escalabilidade do sistema

[Gamma et al., 1994]. A Figura 3.2 ilustra de forma visual a arquitetura interna de um Widget, evidenciando sua interação com outros componentes e realçando sua função na estrutura geral da aplicação.

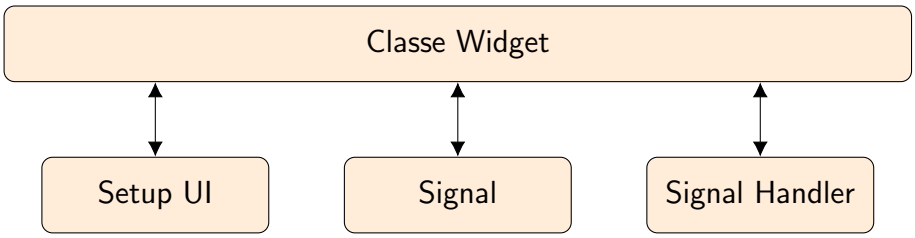


Figura 3.2: Representação da arquitetura de software de um Widget.

Fonte: Elaborado pelo autor, (2024)

3.1.3 Barramento de Eventos (Signals do Qt)

A comunicação entre os widgets e outros componentes da aplicação é realizada através de um barramento de eventos, implementado utilizando o mecanismo de Signals e Slots do Qt [Blanchette and Summerfield, 2008]. Os sinais são emitidos em resposta a eventos do usuário ou do sistema e podem ser conectados a slots que executam ações específicas.

Esse barramento atua como um sistema de publish-subscribe, onde os widgets ou serviços podem se inscrever em tópicos de interesse e reagir aos eventos correspondentes. Essa abordagem promove o desacoplamento entre os componentes, permitindo maior flexibilidade e modularidade [Frank Buschmann and Schmidt, 2007]. A Figura 3.3 mostra de forma visual a arquitetura do barramento de eventos.

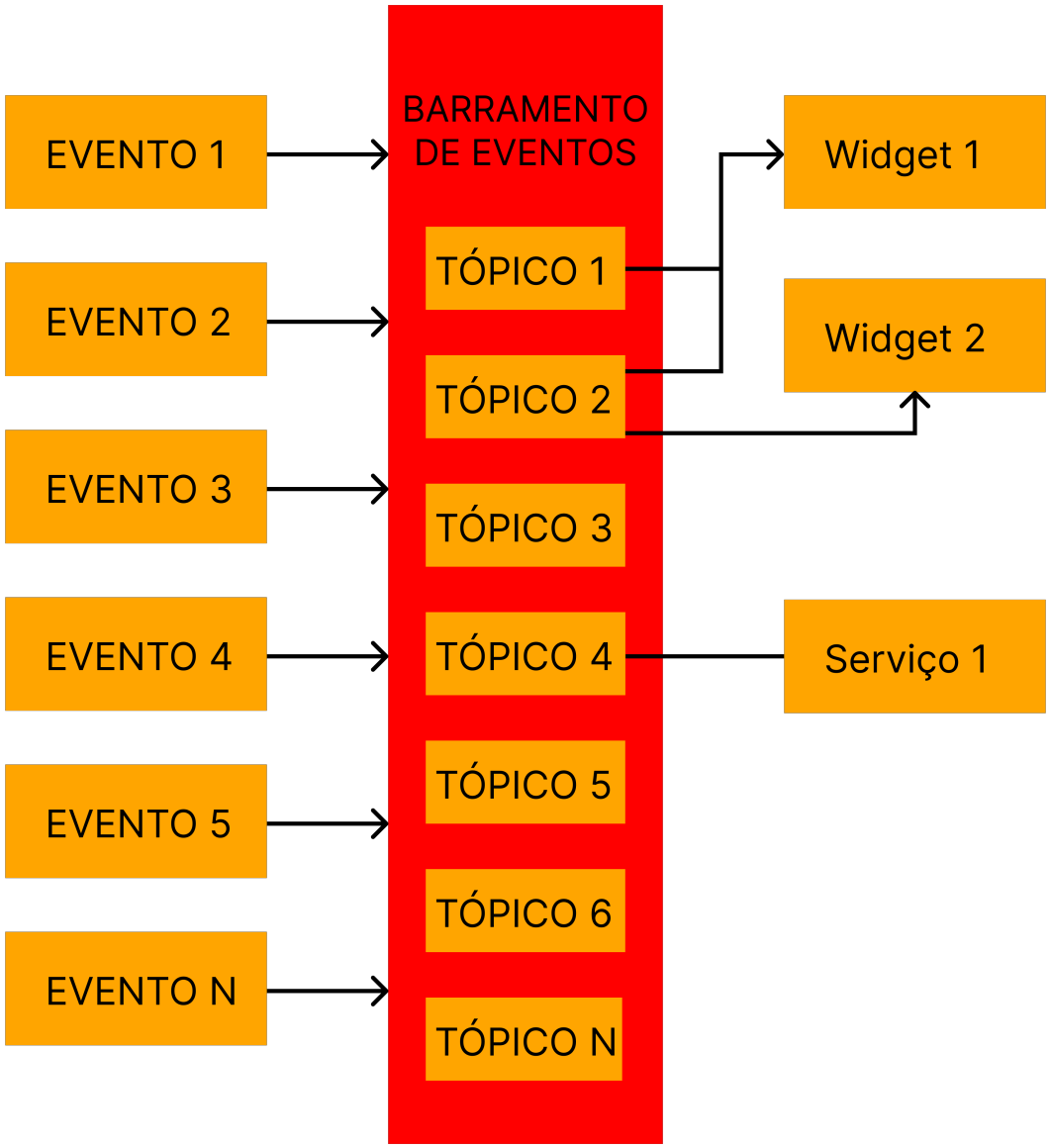


Figura 3.3: Representação da arquitetura de software do barramento de eventos.

Fonte: Elaborado pelo autor, (2024)

3.1.4 Máquina de Estados Central e Manipuladores

Uma máquina de estados central gerencia a lógica de fluxo da aplicação, recebendo os eventos emitidos pelos widgets e tomando as decisões apropriadas. Essa máquina de estados processa os sinais recebidos, determina a ação necessária e emite novos sinais, garantindo que a aplicação responda corretamente às interações do usuário [Harel, 1987].

Os handlers implementam a lógica de negócios da aplicação. Eles atuam como intermediários entre os widgets e os models, processando os dados e executando as operações necessárias. Por exemplo, quando um usuário solicita a conexão com o servidor, o handler correspondente gerencia o processo de autenticação e estabelece a comunicação via socket [Gamma et al., 1994].

Adicionalmente, a integração entre a máquina de estados central e os handlers é fundamental para a manutenção da consistência e integridade da aplicação. A máquina de estados orquestra a transição entre diferentes estados da aplicação com base nos eventos recebidos, enquanto os handlers executam as operações específicas associadas a cada estado ou evento. Essa colaboração permite a implementação de fluxos de trabalho complexos de forma organizada e modular. A utilização de padrões de projeto como o State e o Strategy facilita a extensão e a modificação do comportamento da aplicação sem impactar negativamente sua estrutura geral[Gamma et al., 1994].

3.2 Fluxo de Interação na IDE

3.2.1 Exemplo de Fluxo: Conexão com o Servidor

Um exemplo representativo do fluxo de interação na IDE ocorre quando o usuário deseja conectar-se a um servidor:

1. **Ação do Usuário**: O usuário clica no botão de conexão com o servidor (widget).
2. **Caixa de Diálogo**: Uma caixa de diálogo é apresentada, solicitando as informações necessárias para a conexão (widget).
3. **Emissão de Evento**: Ao confirmar, um *signal* é emitido para o barramento de eventos com o tópico "conexão com servidor".
4. **Processamento pelo Barramento**: A máquina de estados central recebe o *signal* e encaminha para os serviços inscritos no tópico.
5. **Ação do Serviço**: O serviço de socket recebe o *signal*, tenta estabelecer a conexão e retorna o resultado.
6. **Atualização da Interface**: O resultado (sucesso ou falha) é emitido como um novo *signal*, permitindo que os widgets atualizem a interface conforme necessário.

3.2.2 Sistema Publisher-Subscriber

O sistema de **publisher-subscriber** permite que diferentes componentes da aplicação se comuniquem de forma assíncrona e desacoplada [Eugster et al., 2003]. Widgets e serviços podem publicar eventos em tópicos específicos e se inscrever para receber eventos de seu interesse. Isso facilita a escalabilidade e a manutenção do sistema, pois novos componentes podem ser adicionados sem impactar significativamente a estrutura existente.

A comunicação entre Widgets e serviços é mediada pelo barramento de sinais. Widgets emitem sinais em resposta a ações do usuário, e os serviços processam esses sinais para

executar operações lógicas ou de comunicação. Essa arquitetura promove a separação de responsabilidades e melhora a testabilidade da aplicação.

O desacoplamento advindo da utilização do padrão publisher-subscriber permite que seus componentes evoluam de forma independente. Essa característica é especialmente vantajosa em sistemas complexos, onde a modularidade e a flexibilidade são essenciais para acomodar mudanças nos requisitos ou integrar novas funcionalidades. O emprego de tópicos específicos para a publicação e assinatura de eventos possibilita a implementação de mecanismos de filtragem e priorização, aprimorando a eficiência da comunicação e o desempenho geral da aplicação [Gero Muhl, 2006]. Dessa forma, o sistema publisher-subscriber não apenas facilita a escalabilidade e manutenção, mas também promove uma arquitetura mais robusta e adaptável. A Figura 3.4 ilustra de forma visual a arquitetura interna do sistema publisher-subscriber.

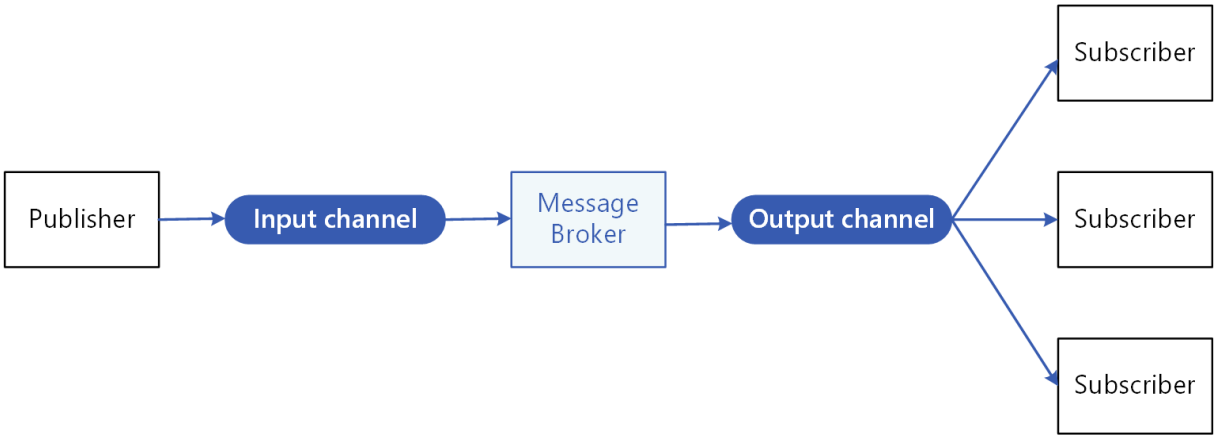


Figura 3.4: Ilustração da arquitetura do sistema publisher-subscriber.[1]

3.3 Arquitetura do Servidor

O servidor foi implementado em Python, utilizando a biblioteca Textualize, que permite a criação de interfaces de terminal ricas e interativas. Essa escolha possibilita um consumo mínimo de recursos e facilita a execução em dispositivos embarcados com capacidades limitadas.

O servidor possui um serviço de sockets que escuta por conexões de clientes. A comunicação é feita com sockets seguros (SSL), o que garante que os dados transmitidos estejam criptografados e protegidos contra a interceptação ou adulteração por entidades externas à comunicação [Stallings, 2016]. A implementação utiliza a biblioteca nativa ssl do Python configurada para utilizar protocolos e cifras seguros.

[1]Fonte: Site da Microsoft. Disponível em: `https://learn.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber` Acesso em 01 de dez. 2024

Para cada novo cliente que se conecta, o servidor criará uma nova thread dedicada ao atendimento dessa conexão. Essa abordagem multithreaded permite que múltiplos clientes sejam atendidos simultaneamente, sem bloqueios ou degradação significativa de desempenho [Butenhof, 1997].

A comunicação entre o cliente e o servidor é realizada através de um protocolo definido, onde os comandos são encapsulados em classes com um **header** especificando o comando a ser executado e um **payload** contendo os argumentos necessários para a execução do comando, podendo o **payload** ser vazio caso não necessário para a execução do comando.

3.4 Comandos e Protocolos de Comunicação

Os comandos enviados entre a IDE e o servidor seguem uma estrutura padronizada:

- **Header**: Indica o tipo de comando ou ação a ser executada.
- **Payload**: Contém os dados ou argumentos necessários para a execução do comando, podendo ser vazio.

Essa estrutura facilita a serialização e a desserialização dos comandos, bem como o processamento dos mesmos, além de facilitar a manutenção do sistema e permitir a extensão futura com novos tipos de comandos sem quebra de compatibilidade. A Tabela 3.1 apresenta os comandos implementados para a comunicação entre a IDE e o servidor:

Tabela 3.1: Lista de Comandos Válidos

Comando	**Descrição**
Listar dispositivos USB do servidor	Solicita a lista de dispositivos USB conectados ao servidor.
Reservar dispositivo USB do servidor ao cliente	Reserva o uso de um dispositivo USB do servidor ao cliente.
Liberar dispositivo USB do servidor ao cliente	Cancelar a reserva de uso de um dispositivo USB do servidor ao cliente.
Conectar dispositivo USB do servidor ao cliente	Estabelece uma conexão do dispositivo USB do servidor diretamente no cliente.
Desconectar dispositivo USB do servidor ao cliente	Encerra a conexão do dispositivo USB com o cliente.
Listar os GPIOs do servidor	Obtém a lista de GPIOs disponíveis no servidor.
Ler um GPIO do servidor	Realiza a leitura do estado de um GPIO específico.
Escrever em um GPIO do servidor	Define o estado de um GPIO (alto ou baixo).
Habilitar detecção de mudança de borda em um GPIO do servidor	Configura um GPIO para detectar mudanças de estado.
Desabilitar detecção de mudança de borda em um GPIO do servidor	Desativa a detecção de mudanças em um GPIO.

Fonte: Elaborado pelo autor, (2024)

3.5 Testes e Validação

Para garantir o correto funcionamento da IDE e do servidor, foram realizados testes manuais de integração, visando avaliar o funcionamento conjunto de seus componentes.

O modelo de validação foi projetado no formato de um teste prático entre um circuito de simples elaboração em uma placa de prototipagem, um servidor Linux embarcado, e um computador executando a IDE de testes. O servidor Linux foi conectado através de seus GPIOs a pontos de teste na placa de prototipagem, a fim de simular um teste ICT. A Tabela 3.2 contêm a lista dos componentes utilizados neste teste. As Figuras 3.5 e 3.6 ilustram o ambiente de testes antes e depois de sua montagem. Foi feita a utilização dos GPIOs 27, 22 e 23 do servidor, em conjunto com um pino de alimentação de 3.3 Volts e um pino de ground, os GPIOs 27 e 22 serão conectados a dois leds, por intermédio de dois resistores de 330 Ω e um switch será conectado ao GPIO 23. Desta forma estaremos observando alterações de níveis lógicos configurando os GPIOs 27 e 22, como saída e o

GPIO 23 como entrada, através da IDE desenvolvida.

Tabela 3.2: Tabela de Comandos Válidos

Componente	**Quantidade**
Raspberry Pi 3 Model B	1 Unidade.
Diodo emissor de luz	2 Unidades.
Switch	1 Unidade.
Resistores de 330 Ω	2 Unidades.
Jumpers	9 Unidades

Fonte: Elaborado pelo autor, (2024)

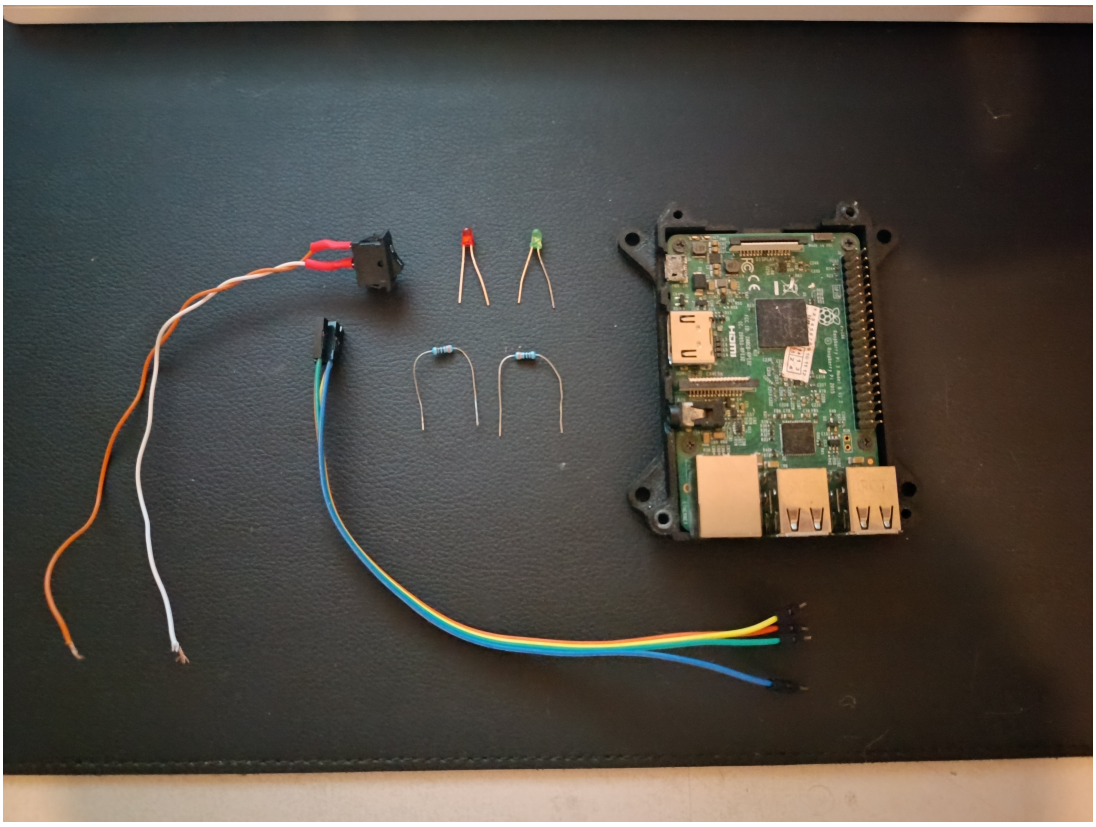


Figura 3.5: Foto dos componentes utilizados durantes os testes de validação, antes da montagem.

Fonte: Elaborado pelo autor, (2024)

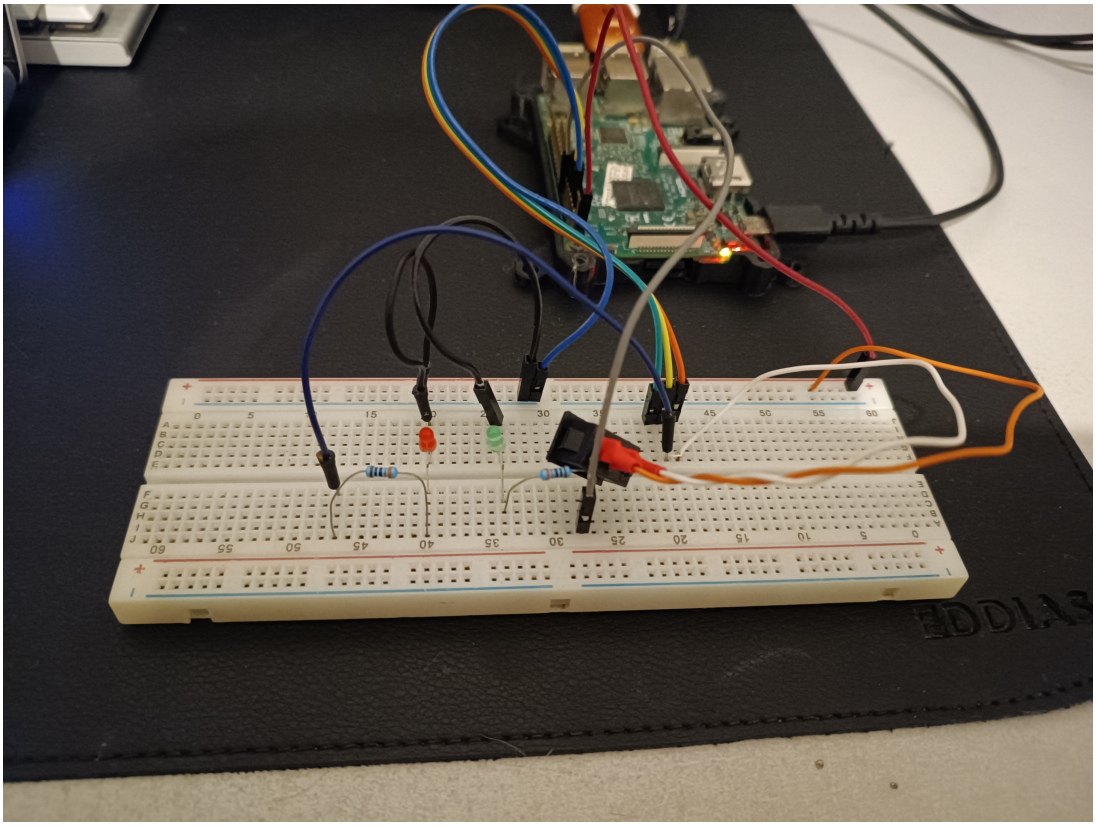


Figura 3.6: Montagem do ambiente de testes de integração da IDE com o servidor desenvolvido.

Fonte: Elaborado pelo autor, (2024)

Capítulo 4

Desenvolvimento e Resultados

Neste capítulo serão apresentadas as interfaces desenvolvidas, os desafios encontrados, as contribuições do projeto e os pontos de melhoria para trabalhos futuros em cima deste projeto.

4.1 Interface Desenvolvida

A Figura 4.5 mostra a interface gráfica desenvolvida em sua tela inicial, após a abertura do programa. Nessa tela é possível visualizar um menu no canto superior do programa, possuindo 3 botões no canto esquerdo, e uma caixa de seleção e dois botões no canto superior direito. O primeiro botão do canto esquerdo, quando pressionado, abre uma caixa de diálogo contendo um formulário para o usuário entrar com as informações do servidor a qual ele quer se conectar, sendo estas informações: endereço IP, porta e apelido do servidor. Os botões à direita deste botão são os botões de salvar e recuperar o estado do programa, respectivamente; eles têm a função de permitir que o usuário salve de forma manual o estado atual do programa em um arquivo nomeável de extensão '.jig'. Este arquivo possui informações em si referente aos servidores já adicionados ao programa, bem como os cadernos de jiga elaborados durante a sessão atual do usuário. A caixa de seleção no canto direito permite que o usuário selecione o servidor no qual quer se conectar, caso haja mais de um adicionado, sendo o botão à sua direita o botão de conexão ao servidor selecionado. Devido ao uso de chaves privadas para realizar a conexão segura entre servidor e cliente, uma caixa de diálogo irá abrir quando pressionado e o usuário terá que digitar a senha de sua chave privada para poder se conectar ao servidor. O último botão à direita é um indicador do estado de conexão entre servidor e cliente, sendo transparente quando não houver nenhuma conexão, verde quando houver uma conexão ativa e amarelo quando houver uma perda de conexão estabelecida.

No central direito, se encontra a barra de navegação, onde o usuário poderá trocar de tela ao pressionar qualquer um de seus botões, do botão superior ao inferior, temos: botão de tela de acesso a USB remoto, botão de tela de depuração de GPIO remoto, botão de

tela de elaboração de cadernos de teste, e por fim, o botão de gerenciamento de chaves de autenticação remota.

No canto inferior central, se encontra a janela de depuração, onde os logs do programa são escritos conforme a necessidade, e com três graus de coloração de texto: branco para informações, amarelo para eventos que requerem atenção e vermelho para erros.

No canto central, temos as telas principais de cada funcionalidade do sistema, que serão apresentadas a seguir.

No Vídeo [de França Moura, 2024], temos uma demonstração de uso da IDE, com o passo a passo para se realizar a conexão ao servidor conectado a uma jiga de testes, simulado através de uma protoboard, contendo dois leds, conectados aos GPIOS 27 e 22, respectivamente, com o intuito de demonstrar mudanças de nível lógico nestes GPIOs configurados como saídas, e um botão ligado a uma fonte de alimentação e ao GPIO 23, este configurado como entrada, para se observar uma mudança de nível lógico ao se acionar o botão.

4.1.1 Tela de Acesso a Dispositivos USB Remotos

A Figura 4.5 mostra a tela de acesso a dispositivos USB remotos, nesta tela temos uma tabela com a descrição de cada dispositivo USB disponível no servidor conectado, sendo possível solicitar, através de dois botões no canto direito, a reserva do dispositivo USB selecionado para o cliente e a conexão entre o dispositivo reservado e o cliente que está conectado. A tabela possui informações referentes ao fabricante do dispositivo, o nome do dispositivo, o estado de uso do dispositivo e o endereço de barramento do dispositivo, sendo assim o usuário poderá visualizar todos os dispositivos disponíveis no servidor da jiga de teste podendo solicitar a conexão com um ou mais dispositivos. Na Figura **??** podemos ver uma lista de dispositivos remotos, estando um deles conectado à máquina do cliente através do protocolo USBIP, na Figura 4.2 podemos ver que a conexão foi realizada entre uma máquina rodando o sistema operacional Windows 11 e o servidor da jiga que está rodando uma distribuição Linux.

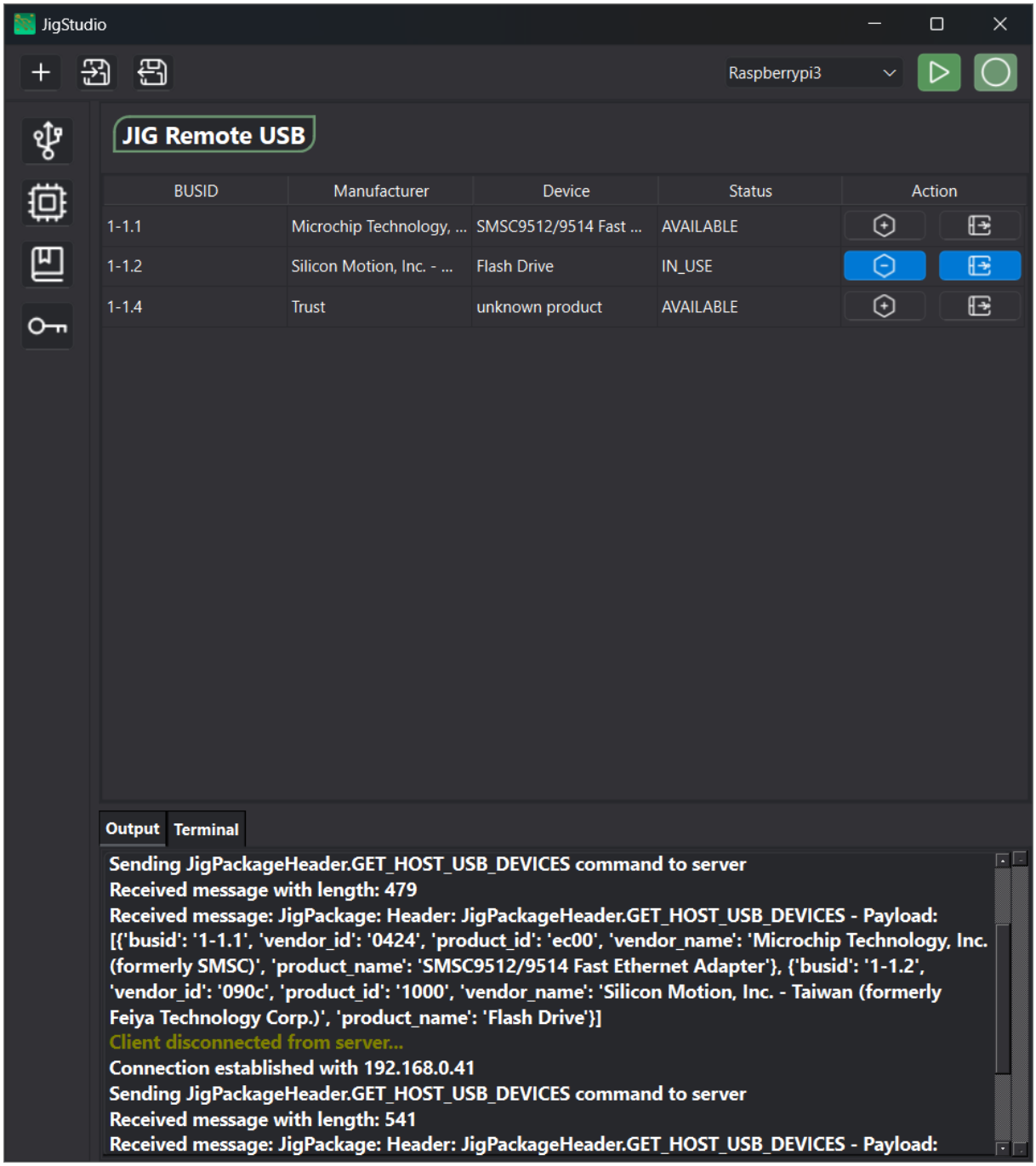


Figura 4.1: Teste de conexão remota de um dispositivo USB conectado conectado ao servidor da jiga.

Fonte: Elaborado pelo autor, (2024)

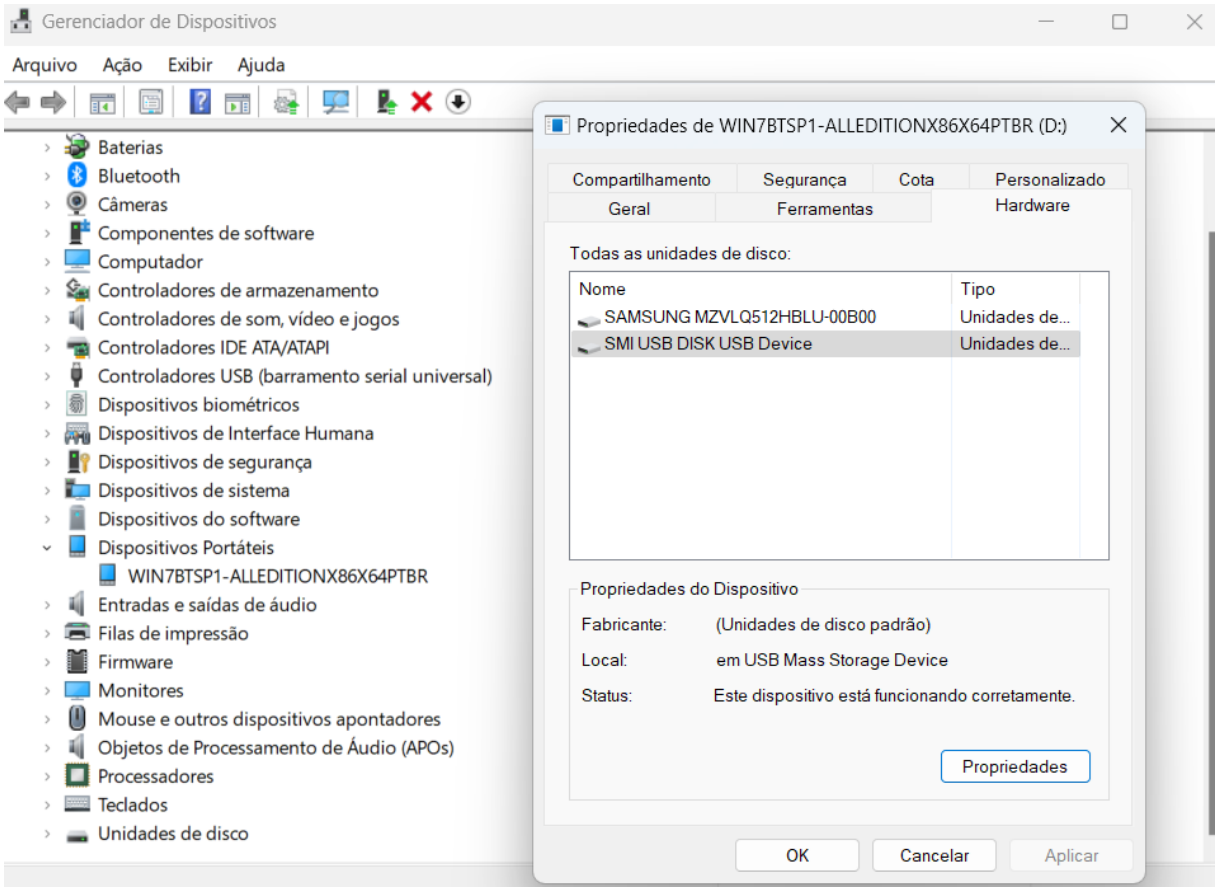


Figura 4.2: Gerenciador de dispositivos do Windows 11, com um "Dispositivo Portátil"compartilhado do servidor da jiga.

Fonte: Elaborado pelo autor, (2024)

4.1.2 Tela de depuração e manuseio de GPIOs

A Figura 4.6 mostra a tela de depuração de GPIOs, onde o usuário pode realizar uma varredura pelo servidor em que está conectado para listar todos os GPIOs disponíveis em seu hardware. O Linux permite que os GPIOs sejam divididos em chips, conforme podemos observar na Figura 4.6. Para cada linha em chip, é possível realizar uma leitura, escrita ou habilitar a detecção de mudança de borda da mesma, sendo possível observar o resultado em tempo real na tabela de GPIOs. Na Figura 4.3 e 4.4 podemos ver o resultado de um teste de manipulação de GPIOs do servidor.

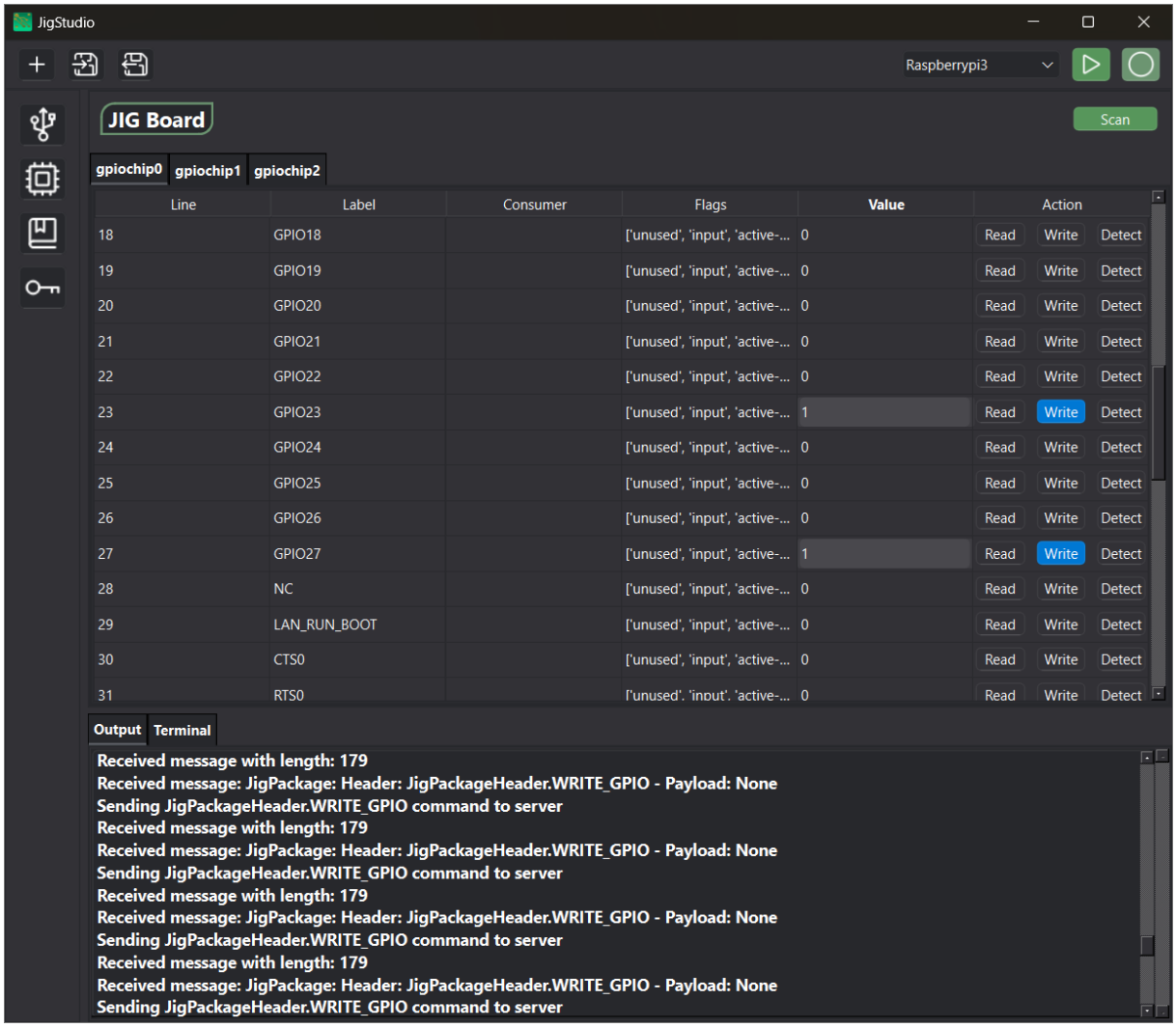


Figura 4.3: Teste de manipulação de GPIOs do servidor da jiga

Fonte: Elaborado pelo autor, (2024)

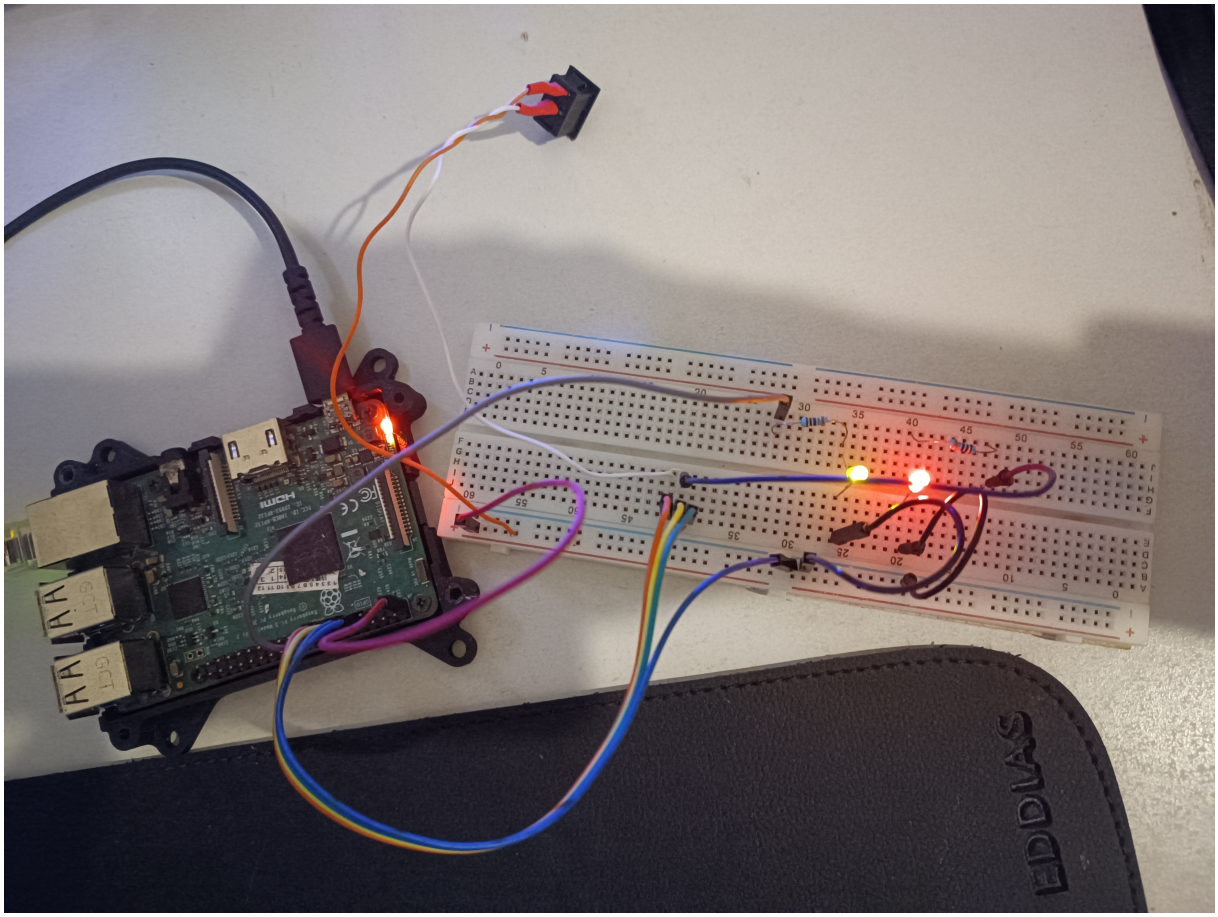


Figura 4.4: Resulta do teste de manipulação de GPIOs do servidor da jiga

Fonte: Elaborado pelo autor, (2024)

4.1.3 Tela de criação de cadernos de teste

A Figura 4.7 mostra a tela de criação de cadernos de teste, onde o usuário pode realizar a criação de diversos testes por caderno, um teste consiste em realizar ações de escrita, leitura ou detecção de mudanças de nível lógico de um GPIO em um ponto de teste, podendo em caso de leitura realizar um desvio condicional, para o caso do nível lógico baixo ou alto, por fim um teste é considerado finalizado com sucesso se o comportamento do sistema reagir conforme o esperado.

As instruções de teste são dispostas no formato de uma lista com hierarquia de árvore e a adição de instruções é feita através de menu de ferramentas no canto direito da interface, através de um botão com formato de quebra cabeças, ao pressionar o botão, o usuário poderá preencher um formulário configurando o tipo de instrução que deve ser executado, em que teste e caderno de teste será inserido e sua posição na hierarquia dos comandos atuais, também é configurado o resultado esperado em caso de sucesso para essa ação.

No canto superior direito da tela, é possível selecionar o caderno de testes atual, criar um novo caderno de testes vazio e executar o caderno de testes selecionado atualmente

por completo.

4.1.4 Tela do servidor

A Figura 4.8 mostra a tela do servidor que é intencionalmente bem simples, e possui apenas um menu lateral para mudar entre duas telas, a de execução e depuração do servidor, e a tela de configurações, e para encerrar o programa.

Na tela do servidor, é possível selecionar uma chave de autenticação através de um seletor, bem como inserir a senha desta chave, tendo apenas um botão no canto direito para realizar a execução do servidor e permitir a conexão de clientes remotos. Esta tela possui uma caixa de texto de depuração, onde todos os eventos que acontecem no servidor são registrados para fim de depuração.

Figura 4.5: Tela de acesso a dispositivos USB remotos

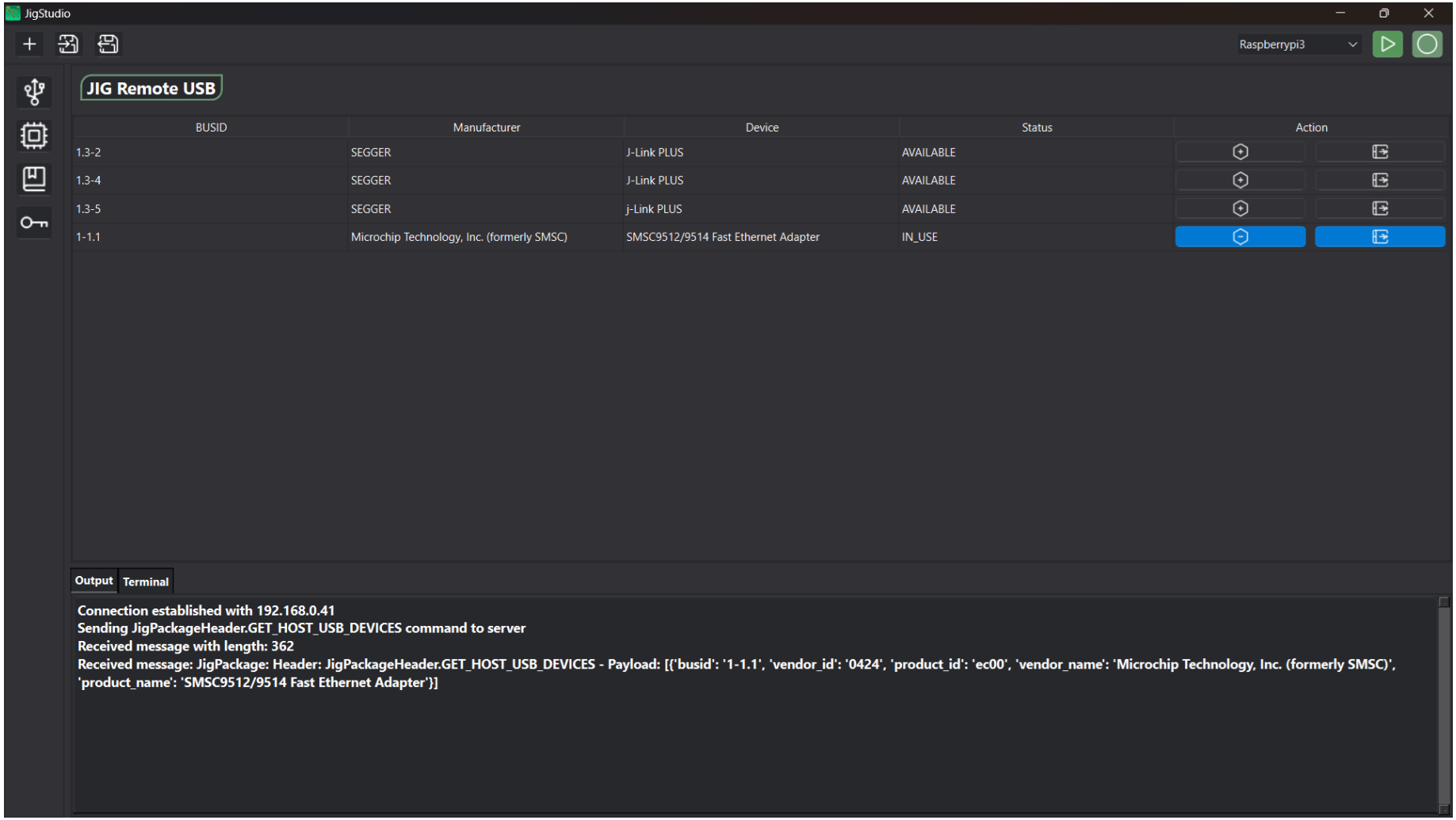
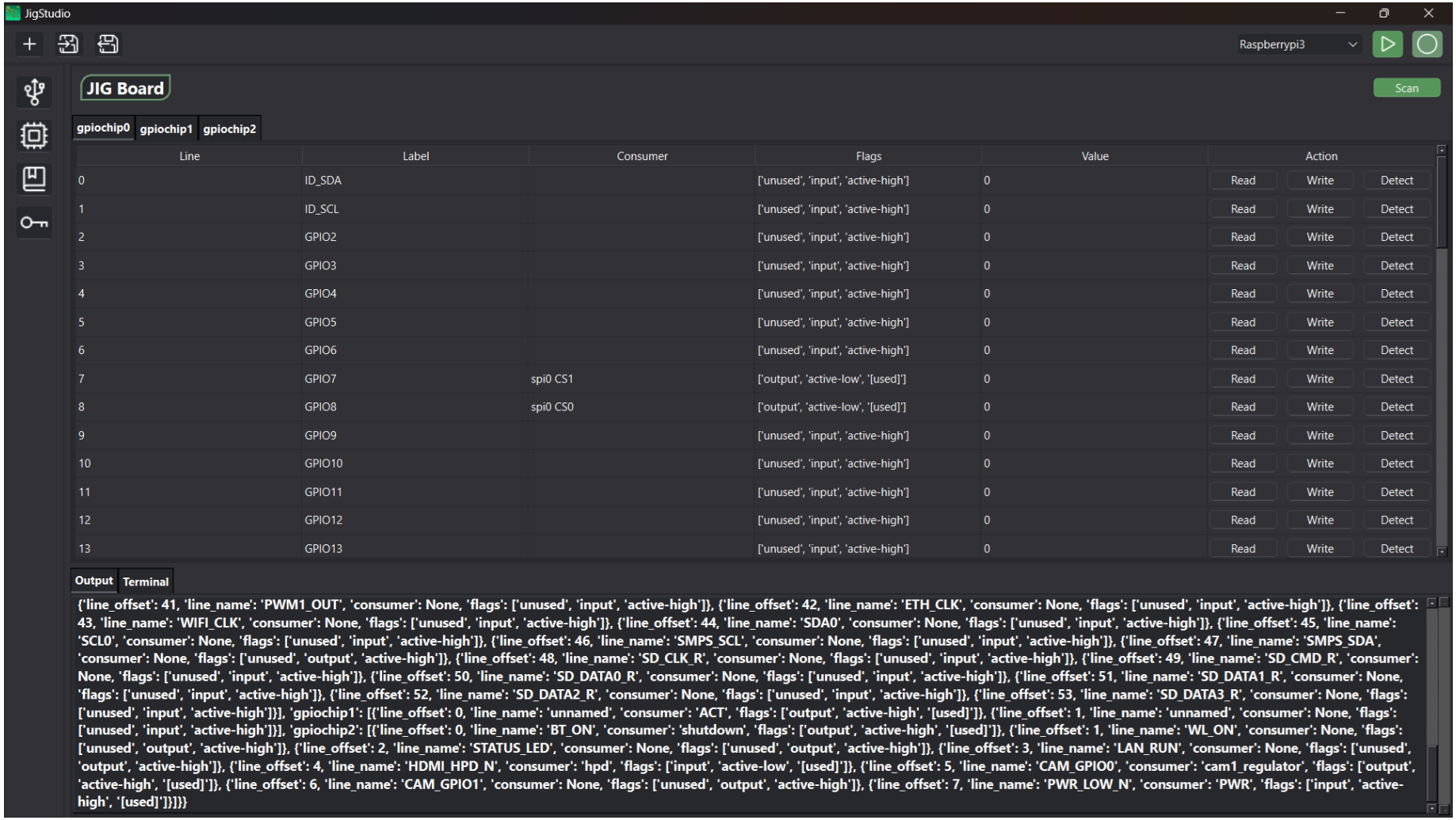


Figura 4.6: Tela de depuração e manuseio de GPIOs



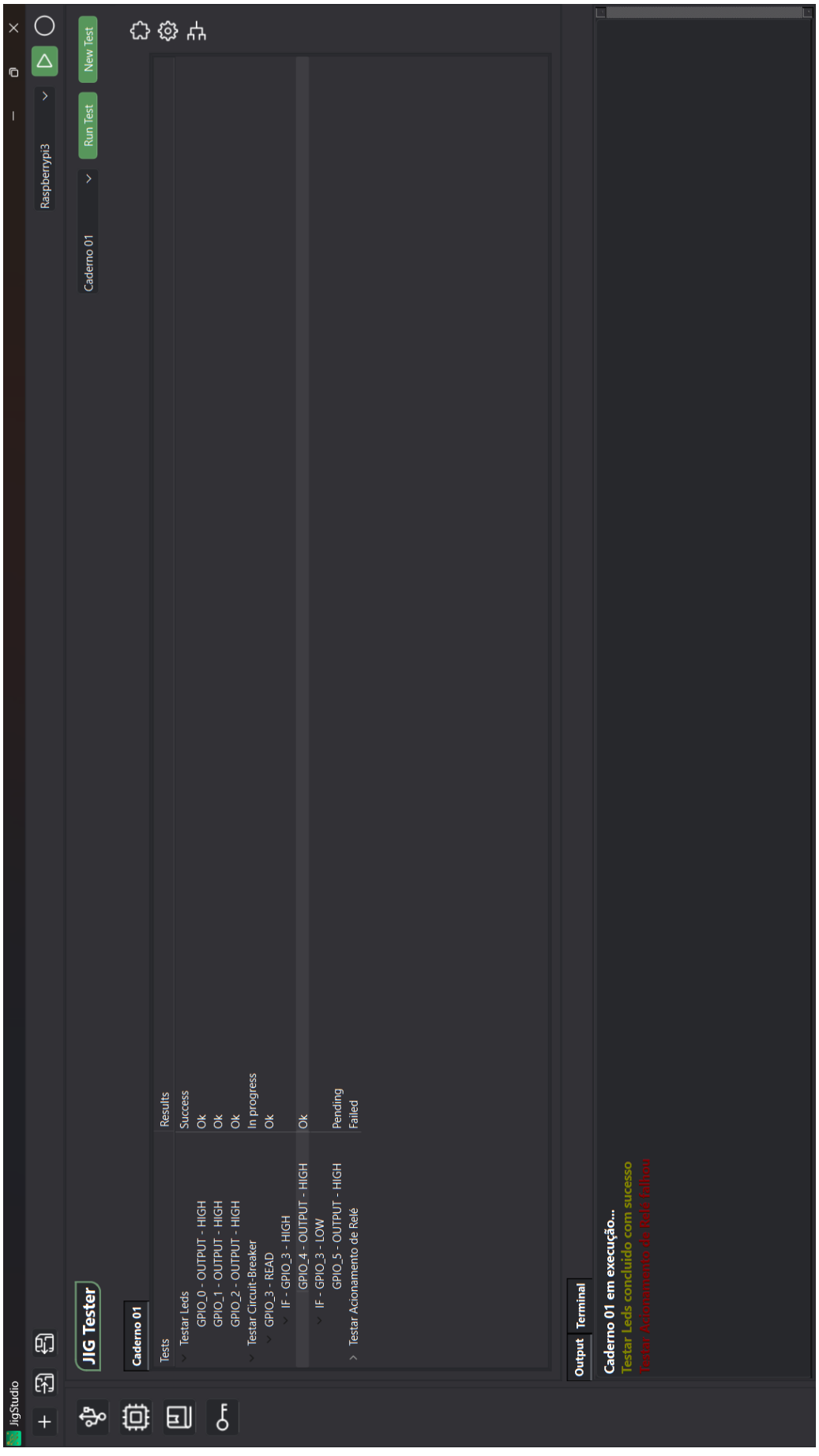
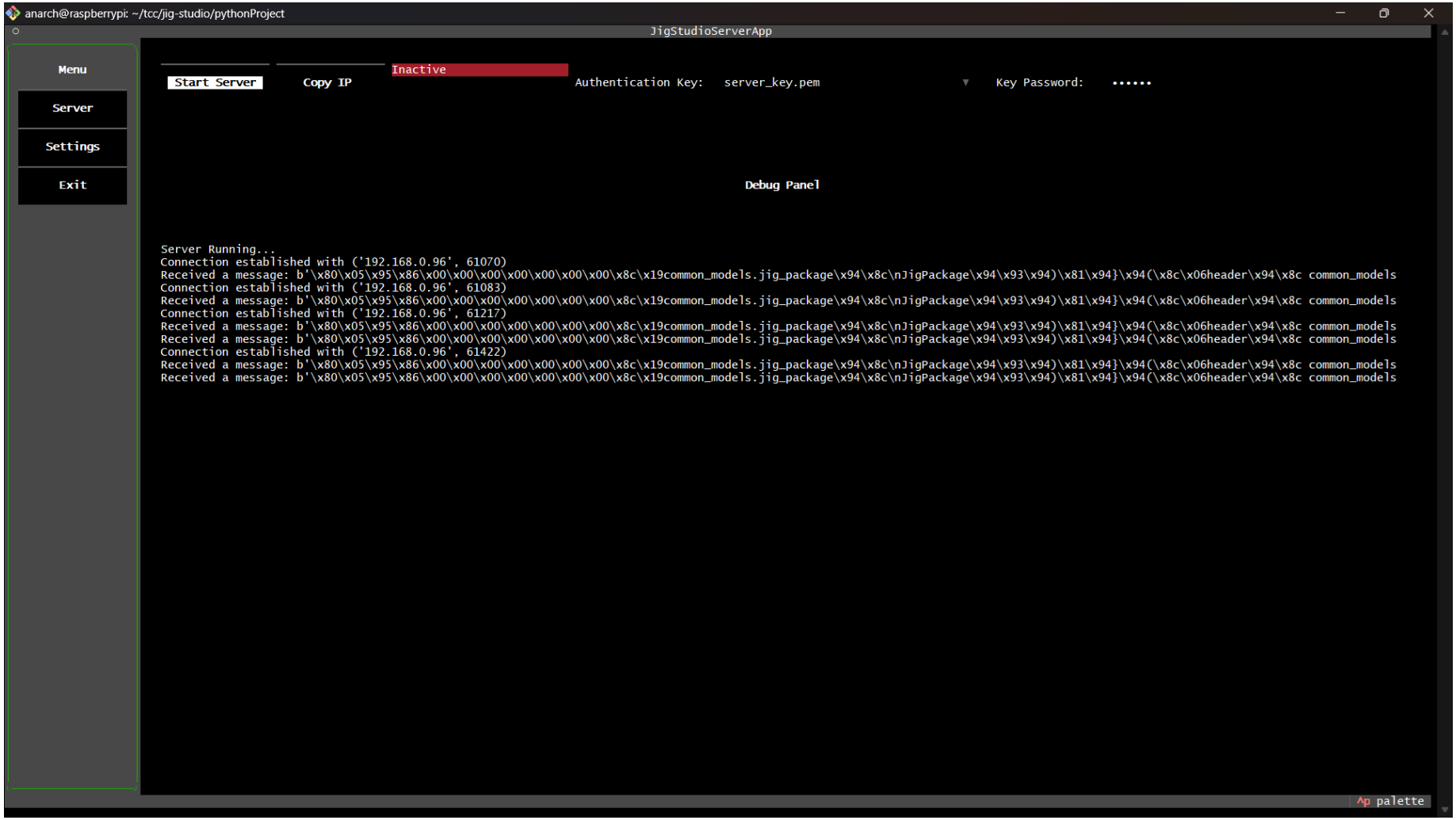


Figura 4.7: Tela de criação de cadernos de teste

Figura 4.8: Tela principal do servidor



Conclusão

Este trabalho apresentou o desenvolvimento de uma IDE para acesso remoto a gravadores de PCBs, bem como um depurador e criador de cadernos de jigas de teste. Formado por três eixos principais: IDE, servidor e PCB. A PCB é o objeto a ser testado, o servidor o meio pelo qual se obterá acesso físico a PCB, e por fim a IDE que permitirá a um usuário criar e depurar testes a serem executados na PCB.

Para se obter estes resultados foram necessários o desenvolvimento de dois softwares, um para a IDE, e outro para o servidor, com o uso de Python e uma arquitetura modular de código, o que simplifica a organização e manutenção desses sistemas.

Certamente existem diversos pontos de melhoria a serem tratados nestes softwares, porém, visto que se trata de um produto mínimo viável, já foi possível realizar a verificação da viabilidade da arquitetura desenvolvida. A IDE demonstrou um bom desempenho ao não consumir muitos recursos de sistema e não demonstrou gargalos causados pela conexão a um servidor, apenas perdendo conexão com o mesmo em momento de baixa na largura de banda de dados da rede em que estava conectada. O servidor também obteve um bom desempenho, visto que estava sendo executado em um hardware de capacidades de recursos reduzidas, com a ressalva de que a depender da quantidade de clientes simultâneos desejados, haverá um requerimento de hardware mais elevado.

A arquitetura proposta se mostrou viável para a resolução do problema. Demonstrando um potencial comercial alto, visto a simplicidade e modularidade do sistema. O que permitirá a expansão de suas funcionalidades de forma escalável. Adicionadas as melhorias citadas na seção anterior, a solução poderá se tornar responsável por ainda mais seções de uma linha de teste, se tornando uma solução completa de testes.

4.2 Desafios e Soluções

Diversos desafios técnicos surgiram no decorrer do desenvolvimento deste trabalho, sendo os principais:

1. **Definição da arquitetura de software**: Devido a natureza específica de um software voltado a interação gráfica com o usuário, diversos fatores foram impactantes para a definição da arquitetura do software, sendo o principal deles a definição da

forma como os elementos gráficos(widgets) se comunicariam entre si, de forma a não gerar eventos bloqueantes entre o cliente e o servidor durante sua execução. A solução desenvolvida para esse problema, foi o desenvolvimento de uma arquitetura híbrida entre MVC e Event-Driven, modularizando todos os widgets e tornando a comunicação entre eles e o servidor assíncrona.

2. **Problemas de sincronismo**: Ao percorrer do desenvolvimento e testes iniciais do software, por diversas vezes ocorreram erros de falta de sincronicidade entre elementos do servidor e do cliente, gerando uma série de efeitos indesejados e bugs de difícil depuração. Para evitar este problema foi necessário separar logicamente serviços e funções síncronas das assíncronas.

4.3 Contribuições do Projeto

Destaca-se como contribuições decorrentes do desenvolvimento deste trabalho:

1. **Software modular**: A arquitetura do software desenvolvido pode ser utilizado em contextos diferentes do deste trabalho, devido ao desacoplamento entre os elementos de interface e a lógica de negócios, bem como o sistema de eventos implementado, sendo possível servir de base para outros trabalhos.

2. **Solução generalista para jigas de teste**: Devido a adição de uma camada de abstração entre o elemento físico da jiga de testes e seu elemento lógico, a solução presente facilita a elaboração de cadernos de testes para qualquer que seja a PCB, desde que possua pontos de teste acessíveis a jiga física.

3. **Servidor e cliente eficientes**: Devido a natureza da arquitetura, os frameworks escolhidos e a forma como o sistema foi implementado, tanto o programa principal cliente, como o programa servidor, demonstraram um bom desempenho e consumo de recursos de sistema, sendo assim, os requisitos de hardware necessários para executar essa solução foram minimizados com o maior custo benefício estimado possível.

4.4 Trabalhos Futuros

Conforme o que foi desenvolvido, é possível expor caminhos possíveis de expansão para este trabalho, tais como:

1. **Adição de comandos mais complexos ao caderno de jiga**: Adicionar comandos de escrita e leitura em portas seriais, UART I2C, SPI, etc.

2. **Criação de uma plataforma de hardware**: Criar uma plataforma de hardware exclusiva para o servidor realizar as manipulações dos pinos de testes de PCBs

3. **Monitoramento em tempo real**: Adicionar a possibilidade de leitura em tempo real de todos os pontos de testes de uma PCB, de forma escalável.

4. **Adicionar inspeção visual com IA**: Adicionar a utilização de modelos de IA, juntamente a sensores, para realizar a inspeção visual de uma PCB, para apontar possíveis falhas antes do testes lógico.

5. **Criar um instalador**: Criar um instalador de pacotes e realizar uma "release"do software para facilitar seu uso.

6. **Execução de caderno de teste no servidor**: Adicionar ao servidor a capacidade de receber um arquivo contendo o caderno de teste que deve ser executado, visando um ambiente de teste em massa.

7. **Adicionar zonas de temperatura**: Adicionar sensores de temperatura e a possibilidade de configurar zonas de temperatura através da IDE, para que o programa consiga avisar e até mesmo desligar o dispositivo em caso de detecção de níveis acima do permitido de temperatura em uma dada zona na PCB.

8. **Geração de relatórios**: Adicionar a geração de relatórios contendo os resultados dos cadernos de teste executados de forma detalhada.

Bibliografia

[Blanchard and Fabrycky, 2010] Blanchard, B. S. and Fabrycky, W. J. (2010). *Systems Engineering and Analysis*. Pearson, 5 edition.

[Blanchette and Summerfield, 2008] Blanchette, J. and Summerfield, M. (2008). *C++ GUI Programming with Qt 4*. Prentice Hall, Upper Saddle River.

[Buckroyd, 2013] Buckroyd, A. (2013). *In-Circuit Testing*. Butterworth-Heinemann.

[Butenhof, 1997] Butenhof, D. R. (1997). *Programming with POSIX Threads*. Addison-Wesley.

[Cheung, 2010] Cheung, J. T. D. S. P. Y. (2010). *Boundary-Scan Interconnect Diagnosis*. Springer.

[Community, 2024] Community, T. K. D. (2024). Usb/ip protocol. `https://docs.kernel.org/usb/usbip_protocol.html`.

[de França Moura, 2024] de França Moura, A. V. P. (2024). Demonstração de manuseio da ide desenvolvida. Vídeo. Disponível em: `https://drive.google.com/file/d/1T3e4pp_1hiGNBWlQGLw-YXKk-IIHk7Qj/view?usp=sharing`. Acesso em: 01 dez. 2024.

[Drew et al., 2016] Drew, D., Newcomb, J. L., McGrath, W., Maksimovic, F., Mellis, D., and Hartmann, B. (2016). The toastboard: Ubiquitous instrumentation and automated checking of breadboarded circuits. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST '16, page 677–686. ACM.

[Eugster et al., 2003] Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131.

[Frank Buschmann and Schmidt, 2007] Frank Buschmann, K. H. and Schmidt, D. C. (2007). *Pattern-Oriented Software Architecture, Volume 5: On Patterns and Pattern Languages*. John Wiley & Sons.

[Gamma et al., 1994] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

[Geier, 2015] Geier, M. J. (2015). *How to Diagnose and Fix Everything Electronic*. McGraw-Hill Education, New York, 2 edition.

[Gero Muhl, 2006] Gero Muhl, Ludger Fiege, P. P. (2006). *Distributed Event-Based Systems*. Springer.

[Golaszewski, 2022] Golaszewski, B. (2022). Libgpiod v2: New major release with a ton of new features. Vídeo. Disponível em: `https://www.youtube.com/watch?v=6fxcDDLII6Y`. Acesso em: 30 nov. 2024.

[Groover, 2015] Groover, M. P. (2015). *Automation, Production Systems, and Computer-Integrated Manufacturing*. Pearson, Boston, 4 edition.

[Harel, 1987] Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274.

[Johnson, 2003] Johnson, H. (2003). *High-Speed Signal Propagation: Advanced Black Magic*. Pearson.

[Liu et al., 2016] Liu, A., Zou, C., Lin, T., Li, J., Tan, C. K., Feng, Z. J., Geiger, D., Liu, S., Wen, J. P., Xiao, J., Liu, L., and Krastev, E. (2016). X-ray inspection methods for controlling pcba potting process — 2dx and partial angle computer tomography. In *2016 Pan Pacific Microelectronics Symposium (Pan Pacific)*, page 1–5. IEEE.

[Love, 2010] Love, R. (2010). *Linux Kernel Development*. Addison-Wesley, Indianapolis, 3 edition.

[Lutz, 2013] Lutz, M. (2013). *Learning Python*. O'Reilly Media, Sebastopol, 5 edition.

[McGugan, 2021] McGugan, W. (2021). Rich: Python library for rich text and beautiful formatting in the terminal. *Python Software Foundation*.

[Moganti and Ercal, 1995] Moganti, M. and Ercal, F. (1995). Automatic pcb inspection systems. *IEEE Potentials*, 14(3):6–10.

[Nayak et al., 2017] Nayak, J. P. R., Anitha, K., Parameshachari, B. D., Banu, R., and Rashmi, P. (2017). Pcb fault detection using image processing. *IOP Conference Series: Materials Science and Engineering*, 225:012244.

[Rescorla, 2000] Rescorla, E. (2000). *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley.

[Singh et al., 2024] Singh, K., Kharche, S., Chauhan, A., and Salvi, P. (2024). Pcb defect detection methods: A review of existing methods and potential enhancements. *Journal of Engineering Science & Technology Review*, 17(1).

[Stallings, 2016] Stallings, W. (2016). *Cryptography and Network Security: Principles and Practice*. Pearson, Boston, 7 edition.

[Streif, 2016] Streif, R. (2016). *Embedded Linux Systems with the Yocto Project*. Pearson, Boston.

[Summerfield, 2007] Summerfield, M. (2007). *Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming*. Prentice Hall, Upper Saddle River.

[Sunahara, 2005] Sunahara, T. H. E. K. H. (2005). USB/IP - a peripheral bus extension for device sharing over IP network. *Proceedings of the 2005 USENIX Annual Technical Conference*.

[Taha et al., 2014] Taha, E. M., Emary, E., and Moustafa, K. (2014). Automatic optical inspection for pcb manufacturing: A survey. *International Journal of Scientific and Engineering Research*, 5(7):1095–1102.

[Tanenbaum and Wetherall, 2010] Tanenbaum, A. S. and Wetherall, D. J. (2010). *Computer Networks*. Pearson, Boston, 5 edition.

[Todorov and Asparuhova, 2023] Todorov, A. and Asparuhova, K. (2023). Methods for automated pcb testing - requiarements, rules and implementation. In *2023 XXXII International Scientific Conference Electronics (ET)*, page 1–4. IEEE.

[Velasco, 2023] Velasco, J. (2023). Designing a bed of nails tester.

[Vempati et al., 2008] Vempati, S. R., Tay, A. A. O., Kripesh, V., and Yoon, S. W. (2008). Bed of nails—100-μm-pitch wafer-level interconnections process. *IEEE Transactions on Electronics Packaging Manufacturing*, 31(4):333–340.

[Verma, 2002] Verma, A. (2002). Optimizing test strategies during pcb design for boards with limited ict access. In *27th Annual IEEE/SEMI International Electronics Manufacturing Technology Symposium*, IEMT-02, page 364–371. IEEE.

[Wolfram Donat, 2021] Wolfram Donat, Matt Richardson, S. W. (2021). *Getting Started with Raspberry Pi*. Make Community, LLC, Prague.