DÉBORA AMORIM SILVA

Implementação de Métodos de Testes para Sistema Embutido de Alarme e Proteção Contra Incêndios.

UNIVERSIDADE DO ALGARVE

Instituto Superior de Engenharia

2022

DÉBORA AMORIM SILVA

Implementação de Métodos de Testes para Sistema Embutido de Alarme e Proteção Contra Incêndios.

Mestrado em Engenharia Eletrotécnica e de Computadores
Especialidade de Sistemas de Energias e Controlo

Trabalho efetuado sob a orientação de:
Professor Doutor Jorge Filipe Leal Costa Semião.

UAlg
UNIVERSIDADE DO ALGARVE
UNIVERSIDADE DO ALGARVE

Instituto Superior de Engenharia

2022

Implementação de Métodos de Testes para Sistema Embutido de Alarme e Proteção Contra Incêndios.

DECLARAÇÃO DE AUTORIA DE TRABALHO

Declaro ser o autor deste trabalho, que é original e inédito. Autores e trabalhos consultados estão devidamente citados no texto e constam da listagem de referências incluídas.

_____

À minha **mãe**,

“Que os anjos te digam o quanto

Eu te amo”

Resumo

Os procedimentos de teste em sistemas têm sido objeto de muitos estudos nas últimas décadas, por proporcionarem mais segurança e confiabilidade aos sistemas, para além de diminuírem custos com problemas futuros. O controlo para aplicações de sistemas embutidos está a tornar-se cada vez mais complexo, devido à crescente implementação de novas e diferentes funções de utilização. No entanto, a maior parte do investimento é dedicado no desenvolvimento do projeto de *hardware* e *software*, sendo poucos os esforços para o controlo de qualidade através de testes, o que gera falhas que podem comprometer todo o projeto.

Quando se trata de Controlo e Segurança contra incêndios, é essencial que o sistema não falhe, principalmente em momentos de alarme crítico. Deste modo, o estudo e desenvolvimento de técnicas para testes no sistema embutido de proteção e alarme contra incêndios é uma parte de grande importância no projeto.

O presente documento enquadra-se na temática de testes em sistemas embutidos e consiste no estudo e desenvolvimento de testes para sistemas de controlo e segurança contra incêndios. Este relatório de estágio tem como principal objetivo relatar os principais métodos de testes executados aos painéis de controlo de segurança contra incêndios da empresa *Global Fire Equipment*, para além de mostrar novos métodos de testes aplicados ao sistema do painel.

PALAVRAS-CHAVE: autoteste em sistemas embutidos, sistema embutido, segurança contra incêndio, teste em software.

Abstract

Test procedures in systems have been the subject of many studies in recent decades for providing more security and reliability to the systems, and also to reduce costs with possible future errors. The control system for embedded applications is becoming more and more complex, due to the increasing implementation of different usage functions. However, most of the investment is dedicated to the development of the *hardware* and *software* project, and few efforts are made to improve control quality through testing, which generates failures that can compromise the entire project.

When it comes to Fire Control and Safety, it is essential that the system does not fail, especially in times of critical alarm. Thus, the study and development of techniques for testing the embedded fire protection and alarm system is a very important part of the project.

This document is part of the theme of tests in embedded systems and consists of the study and development of tests for control and fire safety systems. This internship report has as main objective to report the main test methods performed to the fire safety control panels of the company Global Fire Equipment, and also to show new test methods applied to the panel system.

KEYWORDS: self-test in embedded systems, embedded system, fire safety, software testing.

Índice

Lista de Figuras

Capítulo 1

1 Introdução

O aumento do desempenho dos processadores e da redução dos custos das memórias, possibilitou o crescimento e o avanço dos sistemas embutidos, que desde então passaram a executar diversas funcionalidades. Observou-se também que houve um aumento significativo na complexidade das funções executadas através dos mesmos [1]. No quotidiano podem-se observar muitos exemplos deste tipo de dispositivos, como sistemas de proteção contra incêndios, aparelhos telefónicos, máquinas fotográficas, tocador de música, entre outros. Tais dispositivos podem aceder a internet, entre outras funcionalidades. São equipamentos que se têm tornado cada vez mais complexos, além de possuírem um *software* complexo. Ou seja, o *hardware* embutido pode ter em sua composição, vários processadores, memórias e partes reconfiguráveis, bem como o *software* embutido que possui milhares de linhas com uma grande restrição de memória [1]. As pessoas que integram a equipa de desenvolvimento de um sistema embutido são especialistas no desenvolvimento de *software* e *hardware* e em conjunto desenvolvem de acordo com as necessidades do projeto.

O departamento de tecnologia da empresa *Global Fire Equipment* (GFE-TEC) trabalha no desenvolvimento de sistemas embutidos para alarme e proteção contra

incêndios. Estes sistemas devem possuir grande fiabilidade, pois trata-se do alarme para proteção de bens e, principalmente, de vidas quando são expostas a situações de incêndio. Percebendo-se que é praticamente impossível evitar a 100% a deflagração de incêndios em construções, uma das melhores maneiras de minimizar perdas é através da deteção do incêndio, juntamente com o alarme para os ocupantes, alerta para os bombeiros e acionamento dos sistemas e equipamentos de segurança, o mais precocemente possível. Desta forma, não só a possível evacuação de ocupantes pode ser feita sem perigo, mas também as equipas de intervenção poderão atuar em melhores condições de segurança e reduzir assim a possibilidade de o fogo alastrar a outros compartimentos ou edifícios vizinhos. Em situações menos graves, uma deteção precoce pode também permitir a atuação rápida das pessoas presentes no edifício, evitando dessa forma que se forme um fogo de proporção significativa, uma vez que ao limitar os fogos que se iniciam através de explosões, passam a ser mais fáceis de apagar as pequenas chamas já formadas.

Entretanto, devido à grande complexidade das instalações, como por exemplo se há exposição a fumo devido ao grande número de fumantes no ambiente, ou então porque a instalação pode encontrar-se próximo a indústrias que produzem uma elevada quantidade de fumo, é suposto que as centrais de controlo contra incêndios tenham a capacidade de detetar e distinguir quando realmente se trata de um incêndio, ou quando simplesmente for apenas devido a fatores presentes no ambiente. Então é devido à existência de tantas variáveis, que o projeto da central de controlo e proteção contra incêndios precisa ter uma série de funcionalidades, para que o usuário tenha a opção de selecionar e projetar o cenário que melhor se adapte as necessidades da área a ser protegida.

À medida que se inserem mais funções e disponibilidades para execução de projetos, aumenta-se também a probabilidade de ocorrerem erros nos sistemas, os totalmente indesejados “bugs”. A alta confiabilidade no sistema é fundamental nos sistemas de proteção. Tempo de inatividade ou falhas de conexão podem ser fatais no momento de operação ativa do sistema, além de gerar um grande custo para a empresa, sendo que este pode ser monetário ou de valor sobre a qualidade dos equipamentos comercializados pela empresa. Na busca de evitar que tais erros aconteçam, seja a nível de *hardware* ou de *software*, funcional ou estrutural, *online* ou *offline*, os equipamentos devem ser devidamente testados para minimizar ao máximo erros de falsos alarmes

(alarmes que ocorrem devido a interrupções normais do ambiente), falha de deteção, falhas de comunicação, controlo, monitoramento ou muitos outros que possam acontecer.

A confiabilidade de sistemas embutidos, como é o caso do sistema de alarme e proteção contra incêndios, depende muito do *software* de controlo, principalmente em função do crescimento da proporção desta parte do sistema e do aperfeiçoamento constante do *hardware* [3]. O grande desafio do teste de *software* embutido são as particularidades associadas a esses sistemas. As aplicações de tal sistema são fortemente afetadas pela proximidade com eventos não determinísticos que ocorrem no dia a dia, o que faz com que um simples procedimento de *test case* tenha que prever eventos de tempo real com entradas inesperadas, falhas de *hardware*, de entre outros. A imensa preocupação com a confiabilidade e segurança em relação à melhor forma de se realizar os testes é porque trata-se de um sistema que é utilizado em uma área em que há risco à vida humana.

A revolução digital tem gerado continuamente novos conceitos, como por exemplo a computação em nuvem, plataformas, big data, cidades inteligentes, aprendizagem de máquina, inteligência artificial e assim por diante, que foram empregados para descrever tendências recentes em computação e comunicação, que impulsionam a automação da sociedade cada vez mais. O último termo a acrescentar é o "gémeo" digital (*digital twins*). Um gémeo digital é uma imagem espelhada de um processo físico que é articulado paralelamente ao processo em questão, normalmente ao corresponder exatamente ao funcionamento do processo que ocorre em tempo real, que foi utilizado pela primeira vez no início dos anos 2000 por Michael Grieves [2].

Sendo o modelo uma imagem espelhada completa, que é a definição assumida de um gémeo digital, então pode-se argumentar que o gêmeo digital não está separado do sistema, mas na verdade é o próprio sistema. Nesse sentido, todos os sistemas físicos podem ter um equivalente que converge e se funde com o sistema em questão. Nesse sentido, um verdadeiro gémeo digital funcionando em tempo real não é diferente do próprio sistema e isso representa a questão de como o gémeo digital pode ser usado para aprender sobre o sistema e usado para explorar, simular e testar novos projetos se for o próprio sistema [4]. Logo, em relação ao futuro, um *Digital Twin* será capaz de executar análises ao sistema com mais precisão e em menor tempo.

1.1 Motivação

A melhoria e segurança no desempenho das funções do sistema de controlo de alarme e proteção contra incêndios é fundamental para manter a credibilidade e confiabilidade da empresa. Apesar de alguns participantes do processo de desenvolvimento afirmarem que é melhor impedir a ocorrência da falha a procurá-las para então corrigi-las, a realidade é que atualmente não é possível se produzir um sistema livre de falhas [5]. Assim, percebe-se que realizar testes é essencial para o desenvolvimento do sistema.

No *Software*, o objectivo do teste é fornecer informações de como prosseguir (ou evoluir) no processo de desenvolvimento. Através das falhas observadas, de acordo com as exigências do sistema, pode-se aplicar melhores decisões e alocar de forma mais eficiente os recursos disponíveis de acordo com o projeto proposto [5].

Então, com o objetivo de obter uma maior confiabilidade ao sistema embutido de proteção contra incêndios, será implementado neste trabalho um Autoteste para *software* do sistema de proteção contra incêndios, Chameleon. Para implementação e desenvolvimento do Autoteste são usadas as ferramentas de testes que permitem aplicar as técnicas e observar os critérios de testes de *software*, no entanto para certificar esse *software* é necessário executá-los no ambiente de trabalho normal.

Além do Autoteste auxiliar da deteção de problemas presentes no sistema, a cobertura de falhas, obtidas com o reuso, pode ser ampliada com a introdução de rotinas (algoritmos) de teste de *hardware* quando implementados no nível de *software* e depois a ser embutido juntamente com o Autoteste constituído para *software*, o que pode constituir a integração de técnicas de teste de *software* e de *hardware*.

1.2 Objetivos e planificação do Trabalho

1.2.1 Objetivos

Este trabalho apresenta os seguintes objetivos:

- Apresentar algumas técnicas para testes em Sistemas de *Hardware*/*Software* (online, offline, funcional, estrutural);
- Apresentar o funcionamento do sistema endereçável de alarme e proteção contra incêndios Chameleon a nível de *Hardware*/*Software*.
- Descrever os testes offline realizados no *firmware* durante o período de estágio na Global Fire Equipment no departamento técnico;
- Implementar um Autoteste para o Sistema de Alarme e proteção contra incêndios para o *Software* do sistema Chameleon que tenha como objetivo detetar de forma automática problemas que podem ocorrer com alta frequência e gravidade significativa nos painéis, como, por exemplo, Autoteste de comunicação entre painéis, confirmação do desabilitar de dispositivos, funcionamento dos sensores de deteção e alarme;
- Resultados e Discussões - como diagnósticos, vantagens e cobertura de falhas com o autoteste aplicado.

1.2.2 Planificação do Trabalho

Este trabalho é dividido em 5 Capítulos, como mostra o fluxograma da Figura 1-1. O primeiro capítulo é a apresentação do relatório, com seus objetivos e planificação do trabalho. No Capítulo 2 são apresentadas as definições dos testes realizados neste trabalho. No Capítulo 3 tem-se a descrição da composição e funcionamento de um sistema de alarme e proteção contra incêndios. No Capítulo 4 foi relatado quais os testes realizados no sistema de painéis da rede Chameleon, além da implementação de novos testes e autotestes. E por fim, no Capítulo 5 estão as considerações finais.

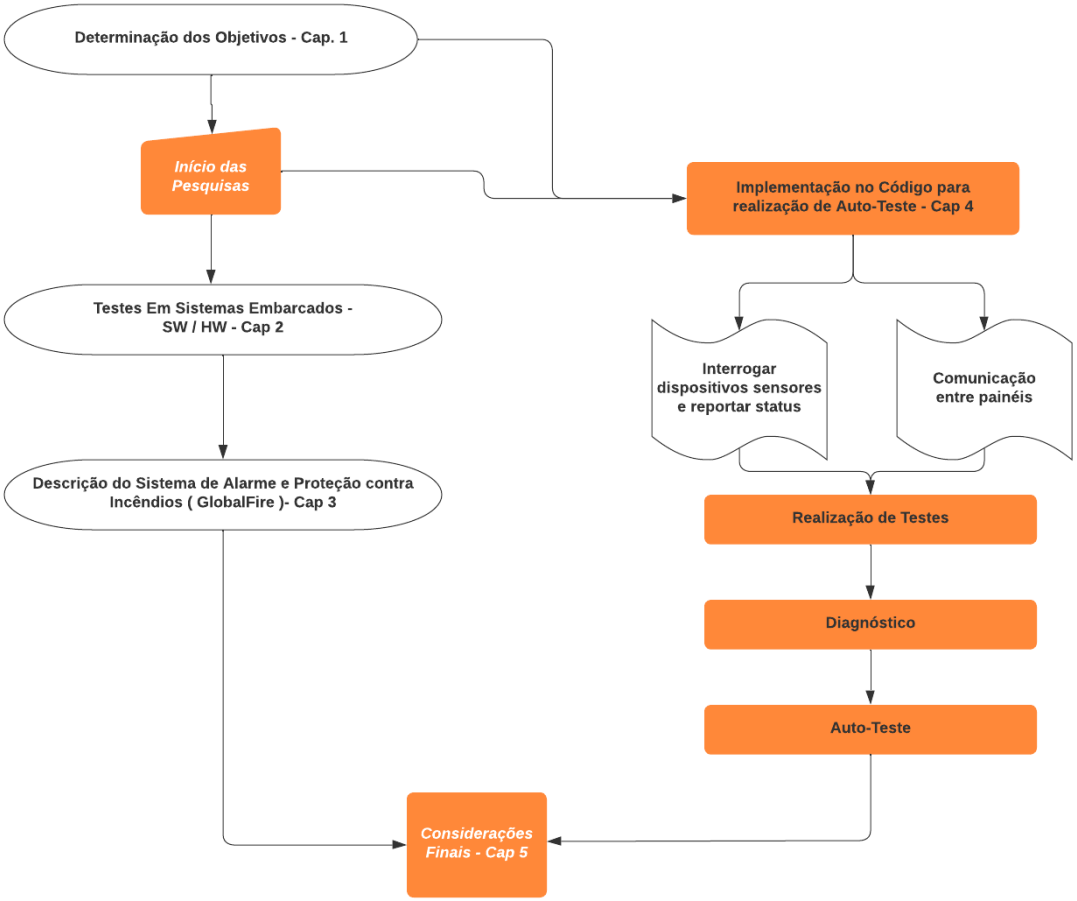


Figura 1-1 –Fluxograma com a planificação do trabalho.

1.2.3 Breve descrição da empresa

A Global Fire Equipment (GFE) é um fabricante europeu de sistemas de segurança contra incêndios de alta qualidade fundada em 1994 na Dinamarca. Atualmente o centro de fabricação localiza-se na cidade de São Braz de Alportel e o escritório de Engenharia em Pesquisa e Desenvolvimento (GFE-TEC) fica situado na cidade de Olhão, ambas na Região do Algarve.

A Global Fire desenvolve e fabrica uma vasta gama de equipamentos do sistema de proteção contra incêndios, entre eles encontra-se o Sistema Chameleon que será abordado neste relatório.

Capítulo **2**

2 Técnicas para Testes em Sistemas de *Hadware/Software*

Com o progresso contínuo da sociedade e o rápido desenvolvimento da informática e tecnologia, a aplicação de computadores e *software* na economia nacional e na vida social está a tornar-se cada vez mais extensa e aprofundada.

Como o cérebro do computador, o *software* desempenha um papel importante nele. A falha de *software* pode causar enormes perdas econômicas e até mesmo colocar em risco a vida humana. Por exemplo, a falha de lançamento do veículo lançador Ariane 5 em 1996 foi causada por falhas de *software* [6].

Todas as etapas do desenvolvimento de *software* requerem envolvimento humano. Como nem o trabalho humano nem a comunicação são perfeitos, os erros são inevitáveis. Ao mesmo tempo, com a melhoria contínua da complexidade dos objetos controlados por meio do computador e o aprimoramento contínuo das funções do *software*, a escala do *software* também está a aumentar. Por exemplo, o código para o sistema operacional Windows NT tem cerca de 32 milhões de linhas [7]. Isso torna os erros mais prováveis. Os erros que as pessoas cometem na fase de projeto do *software* são a principal causa de falha de *software*. A complexidade de *software* é uma fonte extremamente importante de

defeitos. O teste de *software* é um meio importante para garantir a qualidade e a confiabilidade do código implementado.

Atualmente, a prática de engenharia de *software* de muitos projetos é baseada em análise de projetos estruturados. No estágio inicial de desenvolvimento, além da análise e projeto de requisitos, revisão técnica e gerenciamento de engenharia são usados como meio de garantia de qualidade. A depender do tamanho do projeto, é provável que introduza bugs e se expanda para estágios de desenvolvimento subsequentes [6]. Por outro lado, o teste de *software* pode, de facto, encontrar muitos defeitos ocultos no *software*. Por exemplo, no projeto SHOLIS desenvolvido pela Universidade de York para a Marinha Britânica, embora o método formal tenha sido usado para descrever e provar a especificação do *software*, e o método de correção do programa tenha sido usado para eliminar muitos defeitos no estágio inicial de desenvolvimento de *software*, o teste de unidade ainda encontrou falhas em peças individuais do *software*. Com o aprofundamento da compreensão das pessoas sobre a importância do teste de *software*, a proporção de teste de *software* em todo o ciclo de desenvolvimento de *software* está a aumentar.

Agora, algumas instituições de desenvolvimento de *software* colocam mais de 40% do esforço de pesquisa e desenvolvimento em testes de *software*; para alguns *software*s críticos, o custo do teste é de 3 a 5 vezes o custo total de todos os outros estágios de engenharia de *software* [6]. Embora as pessoas também usem métodos formais para descrever e provar especificações de *software* no processo de desenvolvimento de *software* [8], além de métodos como prova de correção do programa e verificação de modelo [9] para garantir a qualidade do *software*, esses métodos ainda têm certas limitações, pois não atingiram o estágio de uso prático generalizado. Portanto, o código do programa, em última análise, reflete a qualidade do *software*. Seja através da metodologia de desenvolvimento de *software* ou pelos benefícios do próprio teste de *software*, o teste de *software* ainda será um meio importante de garantir a qualidade do *software* por muito tempo no futuro.

A tecnologia de teste de *software* existente é geralmente dividida em teste estático e teste dinâmico. O teste estático é o processo de encontrar possíveis defeitos no código do programa ou avaliar o código do programa sem executar o código do programa. O teste estático inclui revisões de código, orientações de código, inspeções de desktop, que

são realizadas principalmente por humanos, e análise estática, que é automatizada principalmente por ferramentas de *software*. Se entendido em um sentido amplo, o teste estático também inclui análise de requisitos de *software* e revisão técnica na fase de projeto [6]. Teste dinâmico é um método de teste de software usado para testar o comportamento dinâmico do código de software. O principal objetivo do teste dinâmico é testar o comportamento do software com variáveis dinâmicas ou variáveis que não são constantes e encontrar áreas fracas no ambiente de tempo de execução do software. O código deve ser executado para testar o comportamento dinâmico. É preciso dois V's para o processo de *Testing* que é verificação e validação. Desse modo, a Verificação é chamada de teste Estático e o outro "V", a Validação é conhecida como teste Dinâmico.

2.1 Defeitos, Erros e Falhas

Um defeito em um sistema eletrônico é a diferença não intencional entre o *hardware* implementado e seu projeto pretendido. Alguns defeitos típicos em chips VLSI (*Very Large Scale Integration)* são [10]:

1. Defeitos do processo – vias de contato ausentes, capacidades parasitas, quebra de óxido, etc.
2. Defeitos de material - defeitos a granel (rachaduras, imperfeições de cristal), impurezas de superfície, etc.
3. Defeitos de idade - rutura dielétrica, eletromigração, etc.
4. Defeitos da embalagem - degradação do contato, vazamentos de vedação, etc.

Os defeitos ocorrem durante a fabricação ou durante o uso dos dispositivos. A repetida ocorrência do mesmo defeito indica a necessidade de melhorias no processo de fabricação ou no projeto do dispositivo. Os procedimentos para diagnosticar defeitos e encontrar suas causas são conhecidas como análises de modo de falha (FMA).

Como o processo de fabricação de placas de circuito impresso (PCBs) é diferente dos chips VLSI, seus defeitos também são diferentes. Um sinal de saída errado produzido por um sistema defeituoso é chamado de erro. Um erro é o resultado de algum "defeito". Uma representação de um "defeito" na função abstrata nível é chamado de falha. A

diferença entre um defeito e uma falha é bastante subtil, ou seja, um defeito não gera, necessariamente, o fim da capacidade de desempenhar uma função, mas pode levar à falha que é o fim da capacidade de desempenhar uma função em específico. Assim, eles são as imperfeições no *hardware* e na função, respetivamente [12].

2.2 Verificação, Validação e Teste

Na Figura 2-1 tem-se a demonstração da atividade de teste, onde é composta pelo processo de verificação, primeiramente, e posteriormente pela validação:

- A atividade de verificação tem características vinculadas ao sistema embutido, sendo que este pode fazer uso de simulações;
- A validação é um conjunto de passos onde o foco e parâmetro é o código do *software* e *hardware* embutido, a revelar falhas de especificação não sanadas pela verificação e erros de codificação.
- O teste se refere as atividades de examinar o comportamento do sistema embutido. A partir de um conjunto de valores de entrada, observa-se o comportamento resposta do sistema através de suas funcionalidades, o que gera resultados e os mesmo são comparados com os resultados esperados para a aplicação [11].

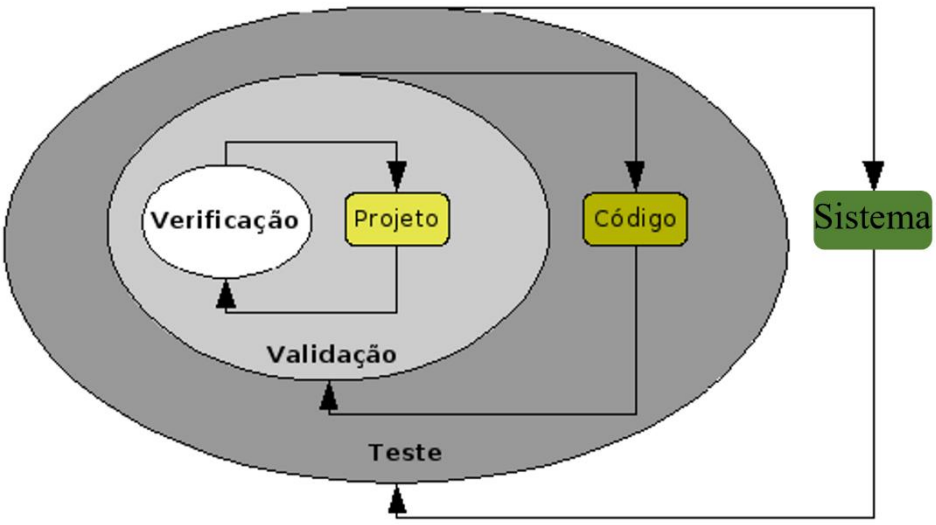


Figura 2-1 – Verificação, validação e teste em teste de sistemas embutidos [11].

2.3 Teste do Sistema

Um computador é um sistema cujos componentes são chips, placas, drive de disco, teclado e *software*s. Um sistema pode ser, e geralmente é, hierárquico. Assim, a unidade de disco pode ser um subsistema que está incorporado no sistema de computador. A ideia básica é que um sistema executa uma função muito maior que nenhum de seus componentes podem fazer sozinhos.

Os sistemas são projetados e construídos por particionamento. Assim, um sistema de computador pode ser particionado em uma caixa de *hardware* e um programa de *software*. Cada um destes pode ser ainda mais particionado. Por exemplo, a caixa de *hardware* seria particionada em várias placas e componentes de E/S. Cada placa será dividida em vários chips VLSI. Enquanto o design do sistema segue uma hierarquia *top-down*, ou seja, começamos o processo de particionamento de uma função global do sistema, o sistema é realmente construído por um processo de baixo para cima [13]. Os componentes de nível mais baixo na hierarquia do sistema, ou seja, chips VLSI, são fabricados e, claro, testados primeiro. Esses são então montados em placas e outros subsistemas, que também são testados.

2.4 Problema de Teste do Sistema Definido

Os testes de sistema são usados em vários estágios da vida útil de um sistema. Esses testes podem ser classificados em duas grandes categorias:

1. Teste funcional ao sistema: Este teste verifica a integridade do sistema a testar especificações funcionais e relacionadas ao desempenho. Isso é basicamente uma aprovação/reprovação tipo de teste, com capacidade diagnóstica limitada, se houver. Em geral, pode ser várias versões de testes funcionais, alguns mais abrangentes do que outros. Um exemplo típico de um teste funcional abrangente é o teste de aceitação de um sistema de computação. Este seria o teste realizado quando o sistema é entregue ao seu usuário. Ele pode testar todas ou a maioria das

funções do sistema e verificar parâmetros de desempenho. Um teste menos abrangente pode ser executado toda vez que o sistema está alimentado. Isso testaria a disponibilidade de todos os subsistemas. Os testes funcionais ao sistema são usados como a etapa final na fabricação e como a primeira etapa na operação ou manutenção do sistema. Se uma falha for indicada por esses testes, o próximo nível de teste envolve o diagnóstico de falhas.

2. Teste de diagnóstico: Um teste de diagnóstico é projetado para isolamento ou diagnóstico de falhas. O diagnóstico refere-se à identificação de uma peça defeituosa. Um teste diagnóstico é caracterizado por sua resolução diagnóstica, definida como a capacidade de se aproximar à culpa. Quando uma falha é indicada por um tipo de teste de aprovação/reprovação em um sistema que está operando em campo, é aplicado um teste de diagnóstico. Para um reparo de campo eficaz, este teste deve ter uma resolução de diagnóstico da menor unidade substituível (LRU)[14]. Para um sistema de computador, o LRU pode ser uma parte como uma placa de memória, disco rígido ou teclado. O teste diagnóstico deve identificar a peça defeituosa para que possa ser substituída. A parte defeituosa é então enviada para a oficina, onde outro teste de diagnóstico com um diagnóstico superior resolução será usada. Este teste identificará a unidade substituível da loja com defeito (SRU)[14], que pode ser um processador ou um chip de memória na placa defeituosa.

2.4.1 Teste funcional versus teste estrutural

O uso de testes funcionais é frequentemente considerado necessário para verificação de design. O objetivo do teste de *hardware* (também conhecido como teste de fabricação) é descobrir qualquer falha causada por defeitos de fabricação ou erros. Uma suposição básica feita é que o projeto que está a ser fabricado está correto.

Em 1959, Eldred derivou testes que observariam o estado dos sinais internos nas saídas primárias de um grande sistema digital. Tais testes são chamados de estruturais porque eles dependem da estrutura específica (tipos de porta, interconexões, netlist) do circuito. Uma das maiores vantagens do teste estrutural é que ele nos permite desenvolver

algoritmos. O foco para esses algoritmos são os modelos de falhas. A maioria dos algoritmos de geração de teste e avaliação de teste (simulação de falha) são com base em modelos de falhas selecionadas [12].

A modelagem de falhas está intimamente relacionada à modelagem do circuito. Na hierarquia do projeto, o nível se refere ao grau de abstração. Assim, o nível comportamental (às vezes chamado de alto nível) tem menos detalhes de implementação e falhas modelos neste nível podem não ter correlação óbvia com defeitos de fabricação. Alto nível modelos de falhas desempenham um papel maior na verificação do projeto baseado em simulação, do que em testes. As exceções são os modelos de falhas funcionais de memórias de semicondutores. Como a função da memória é simples, é possível um teste funcional ser exaustivo e é normalmente usado na prática.

O nível de transferência de registradores (RTL) ou nível lógico consiste em uma netlist de portas e as falhas travadas neste nível são os modelos de falha mais populares em testes digitais. Outros modelos de falhas neste nível são falhas de ponte e falhas de atraso.

Transistor e outros níveis inferiores incluem tipos de falhas que também são conhecidas como falhas dependentes de tecnologia. No nível do componente falhas são modeladas principalmente em testes de circuitos analógicos.

Finalmente, existem modelos de falhas que podem não se encaixar em nenhuma das hierarquias de design. Um exemplo típico é o defeito de corrente quiescente. A utilidade desses modelos decorre do fato de que eles podem representar alguns defeitos físicos não representados por nenhum outro modelo. Por isso, às vezes, esses modelos têm sido chamados de “realistas”. Na verdade, existem muitos modelos de falhas que aparecem na literatura [12].

2.4.2 Teste de Diagnóstico

O teste de diagnóstico é aplicado após a falha de um sistema. O ambiente de reparação do sistema determina o nível ou as unidades a serem identificadas. Por

exemplo, em manutenção de campo, a unidade substituível mais baixa (LRU) seria uma placa, disco com defeito ou um subsistema de E/S. A cardinalidade do conjunto suspeito de LRUs que o teste identifica é definido como sua resolução diagnóstica. Um teste ideal terá a resolução de diagnóstico de 1. Tal teste será capaz de identificar exatamente o defeito unidade e permitirá o reparo mais eficiente. Em uma oficina, pode ser necessário reparar a placa defeituosa e as LRUs seriam chips ou componentes discretos no quadro. Os conceitos de dicionário de falhas e árvore de diagnóstico são relevantes para qualquer tipo de diagnóstico [12].

2.4.3 Dicionário de Faltas

A aplicação de um teste simplesmente nos diz se o sistema sob teste é ou não defeituoso. Devemos analisar ainda mais o resultado do teste para determinar a natureza da falha, para que ela possa ser corrigida. Um dicionário de falhas contém o conjunto de sintomas de teste associado a cada falha modelada [12].

Na aplicação de um teste diagnóstico, também é possível que a síndrome do teste observado não corresponde a nenhuma entrada do dicionário. Em seguida, contamos com heurísticas para completar o dicionário. Por exemplo, suponha que devido a um processo de fabricação que causou um erro, a porta OR no circuito da Figura 2-2 foi substituída por uma porta NOR. A saída invertida falhará em todos os quatro testes, produzindo uma síndrome 1111. Logo, um procedimento de diagnóstico identificará a entrada do dicionário no menor *Hamming* de distância da síndrome de teste observada como o suspeito mais provável sendo c_1, d_1, e_1 e e_0.

Este diagnóstico indica um problema em torno do portão OR, mas não fornece uma ideia clara da falha real [12].

2.4.4 Árvore de Diagnóstico

Segundo [12] uma desvantagem da abordagem do dicionário de falhas é que todos os testes devem ser aplicados antes que uma inferência possa ser feita. Além disso, para grandes circuitos, o volume de armazenamento de dados pode ser grande e a tarefa de combinar a síndrome do teste também pode levar tempo. Um procedimento alternativo,

conhecido como árvore de diagnóstico ou árvore de falhas, é frequentemente encontrado para ser mais eficiente.

Neste procedimento, os testes são aplicados um de cada vez. Após a aplicação de um teste, obtém-se um diagnóstico parcial. O teste a ser aplicado em seguida é escolhido com base no resultado do teste anterior.

Uma árvore de diagnóstico para o circuito da Figura 2-2 é mostrada na Figura 2-3. O teste mostrado na raiz da árvore, é aplicado primeiro. Se mostrar uma falha, o diagnóstico seguirá o ramo inferior rotulado. Neste ponto, o conjunto de falha suspeito contém o teste que é aplicado em seguida. O teste T_4 mostrado na raiz da árvore mostrada na Figura 2-3, é aplicado primeiro. Se apresentar falha, o diagnóstico seguirá o ramo inferior $t_4 = 1$. Neste ponto, o conjunto de falha suspeita contém a_0, b_0, d_0 e e_0. O teste T_2 é aplicado em seguida. Uma falha ($t_2 = 1$.) irá diagnosticar a falha como e_0. Em caso de outras falhas, o diagnóstico seguirá outros caminhos na árvore. Cada nó folha fornece o mesmo tipo de diagnóstico dado pelo dicionário. No entanto, o processo pode terminar antes que todos os testes sejam aplicados, como aconteceu com a falha e_0. A profundidade máxima da árvore de diagnóstico é limitada pelo número total de testes, mas pode ser menos do que isso também.

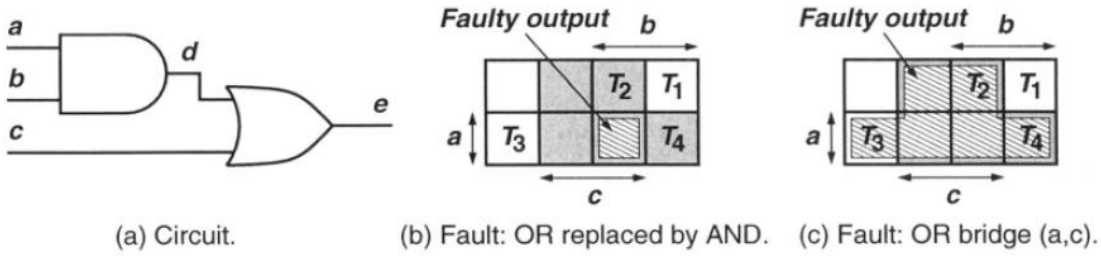


Figura 2-2 - Um exemplo de diagnóstico [12].

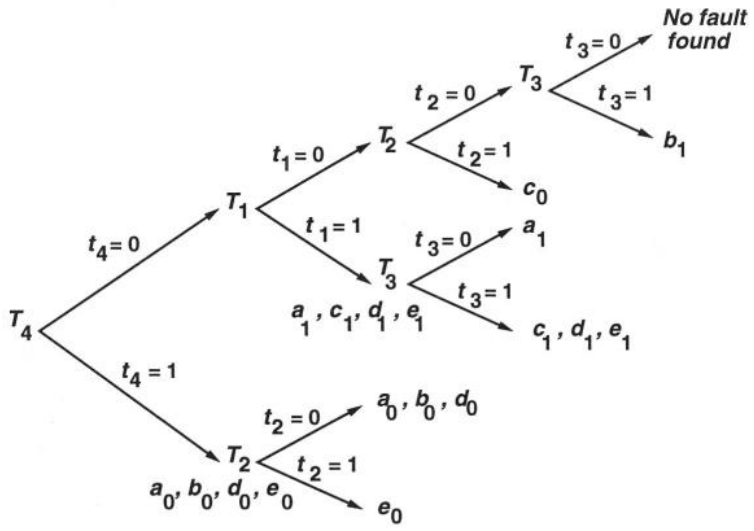


Figura 2-3 - Uma árvore de diagnóstico para o circuito da Figura 2-2 [12].

Capítulo **3**

3 Sistemas de Deteção de Incêndios

Os sistemas de detecção de incêndios são denominados de convencionais ou endereçáveis. Estes diferenciam-se quanto à capacidade de suportar dispositivos, de identificar o local exacto da detecção e de determinar os níveis de fumo, calor ou gás (dependendo do método de detecção) captados no local onde existe o foco de incêndio. Na Figura 3-1 pode-se observar a arquitectura simplificada de um sistema de detecção de incêndios.

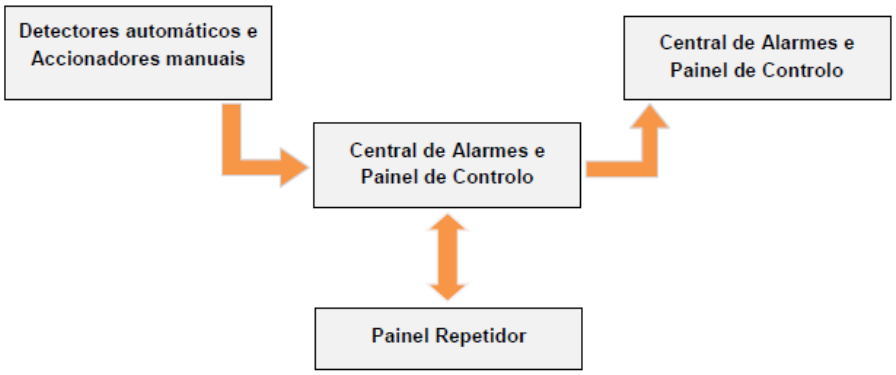


Figura 3-1 - Arquitectura elementar de um sistema de detecção de incêndios [29].

3.1 - Sistema Convencional

Num sistema convencional (Figura 3-2), os diferentes dispositivos de entrada são ligados em paralelo num circuito fechado. Em regra, cada circuito constitui uma zona de alarme que também pode ser denominada de loop.

Ao actuar, um detector introduz uma resistência no circuito, o que leva à passagem de uma corrente específica, e consequentemente o painel a entrar em alarme.

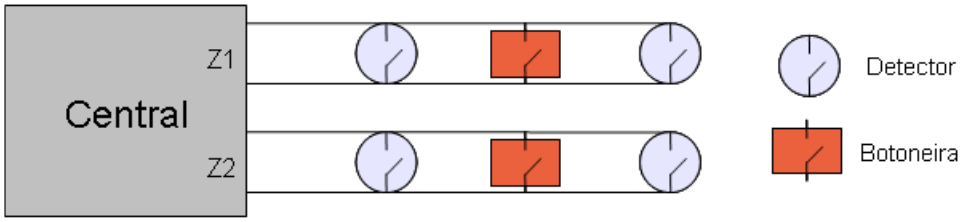


Figura 3-2 - Configuração básica de um sistema convencional [29].

A indicação da zona que houver deteção de fogo é dada por LED's ou por um por um painel LCD que serve de interface com o utilizador, e onde também se encontram botões para comandar e programar a central. A grande "desvantagem" desse tipo de painel em comparação com os endereçáveis é porque não é possível conhecer qual ou quais as áreas que se encontram em fogo somente através do acesso ao painel. Seria necessário procurar o detector em alarme (se um LED vermelho pisca rapidamente ou permanece aceso quando o detector está em alarme), o que é totalmente inviável em uma situação real de alarme.

3.2 - Sistema Endereçável

Num sistema endereçável, cada dispositivo ligado ao painel de controlo tem um endereço digital, configurado no dispositivo (por *hardware*) ou na central (por *software*). Isso permite a identificação precisa do detector que se encontra activo.

Neste tipo de sistemas podem ser ligados um elevado número de dispositivos (detectores, sirenes e módulos diversos) ao loop, (sendo as diferentes zonas criadas logicamente, escolhendo-se a partir da central quais os detectores pertencentes a uma

determinada zona. O estado dos dispositivos é continuamente monitorizado pela central (mestre) que comunica com todos os dispositivos do loop (escravos) sequencialmente, a interrogá-los sobre a sua condição.

A existência de um painel repetidor, instalado numa entrada segura do edifício, permite visualizar o estado do sistema bem como realizar o controlo do mesmo. A sua função é reproduzir as informações do painel existente no gabinete da central, de forma a que se possa avaliar correctamente a situação do sistema e a progressão do incêndio mesmo que este se localize longe da central.

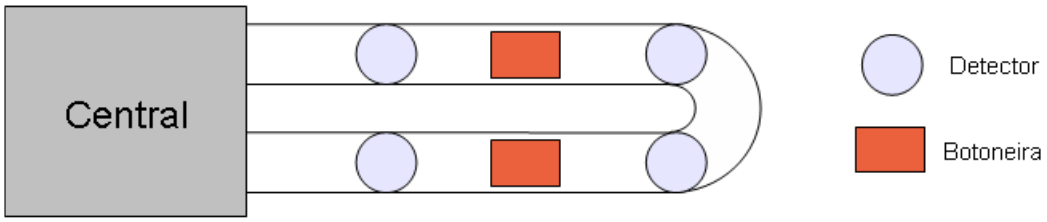


Figura 3-3 - Configuração básica de um sistema endereçável [29].

Neste tipo de sistemas é possível haver uma quantificação dos níveis de fumo e/ou calor nos detectores, permitindo desta forma a existência de níveis de pré-alarme, isto é, permite que a central saiba quando um detector está próximo de uma condição de alarme. Tipicamente, esse nível é de 80 % do nível de alarme.

Cada dispositivo (detector, estação de acionamento... etc.) tem um número exclusivo atribuído para chamar o endereço para relatar alarmes e problemas. Dispositivos endereçáveis transmitem uma mensagem eletrónica de volta à central de controlo que representa o seu estado (Normal, Alarme, Fogo) quando consultado pela Unidade de Controlo.

3.2.1 . Configuração de um sistema automático de deteção de incêndios (endereçável)

Um sistema de deteção automática de incêndios (SADI - Figura 3-4) é uma instalação técnica capaz de registar um início de incêndio, sem a intervenção humana,

podendo vigiar permanentemente zonas inacessíveis à deteção humana, transmitir as informações correspondentes a uma central de sinalização e comando (CDI – central de deteção de incêndios), dar o alarme automaticamente, de forma local e restrita, ou geral, ou à distância (alerta) e accionar todos os comandos (imediatos ou temporizados) necessários à segurança contra incêndio dos ocupantes e do edifício onde está instalado: fechar portas, accionar dispositivos de evacuação de fumos, parar elevadores, comandar sistemas automáticos de extinção, desligar energia eléctrica, etc [31].

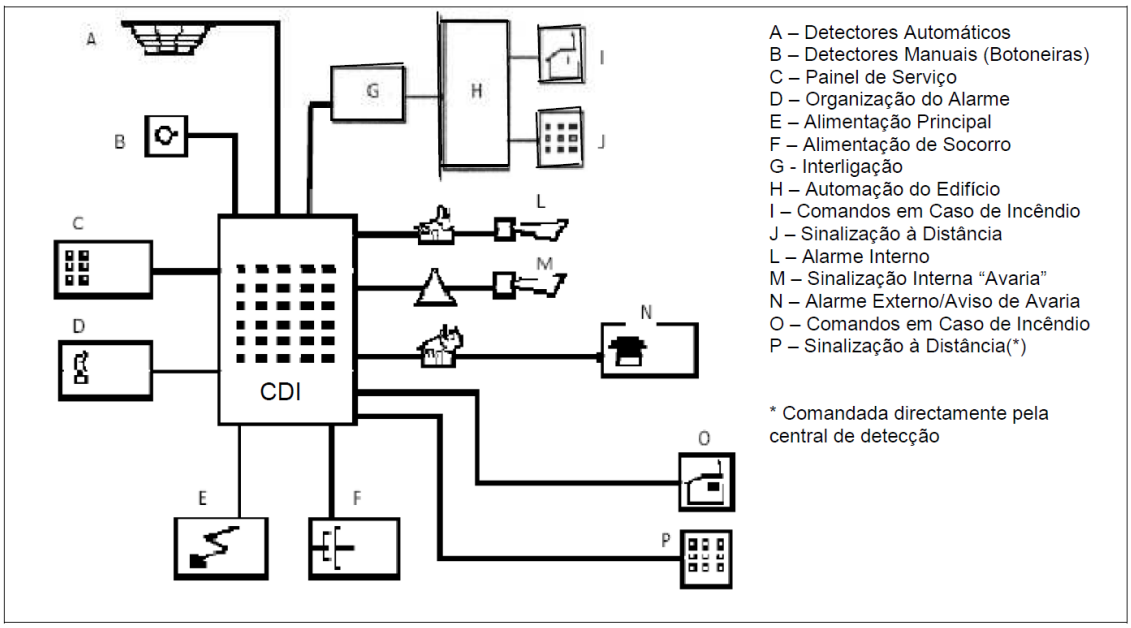


Figura 3-4 - – Configuração tipo de um SADI [33].

A CDI deve ter duas temporizações programáveis, a de presença que corresponde à aceitação do alarme por parte do operador e a de reconhecimento que corresponde à confirmação local do alarme.

Na figura seguinte (Figura 3-5) apresenta-se um possível fluxograma da organização de um sistema automático de detecção de incêndios, onde se considera que qualquer alarme originado em um botão de alarme é sempre verdadeiro, enquanto um alarme originado por um detector automático precisa de verificação, caso o sistema se encontre no modo dia.

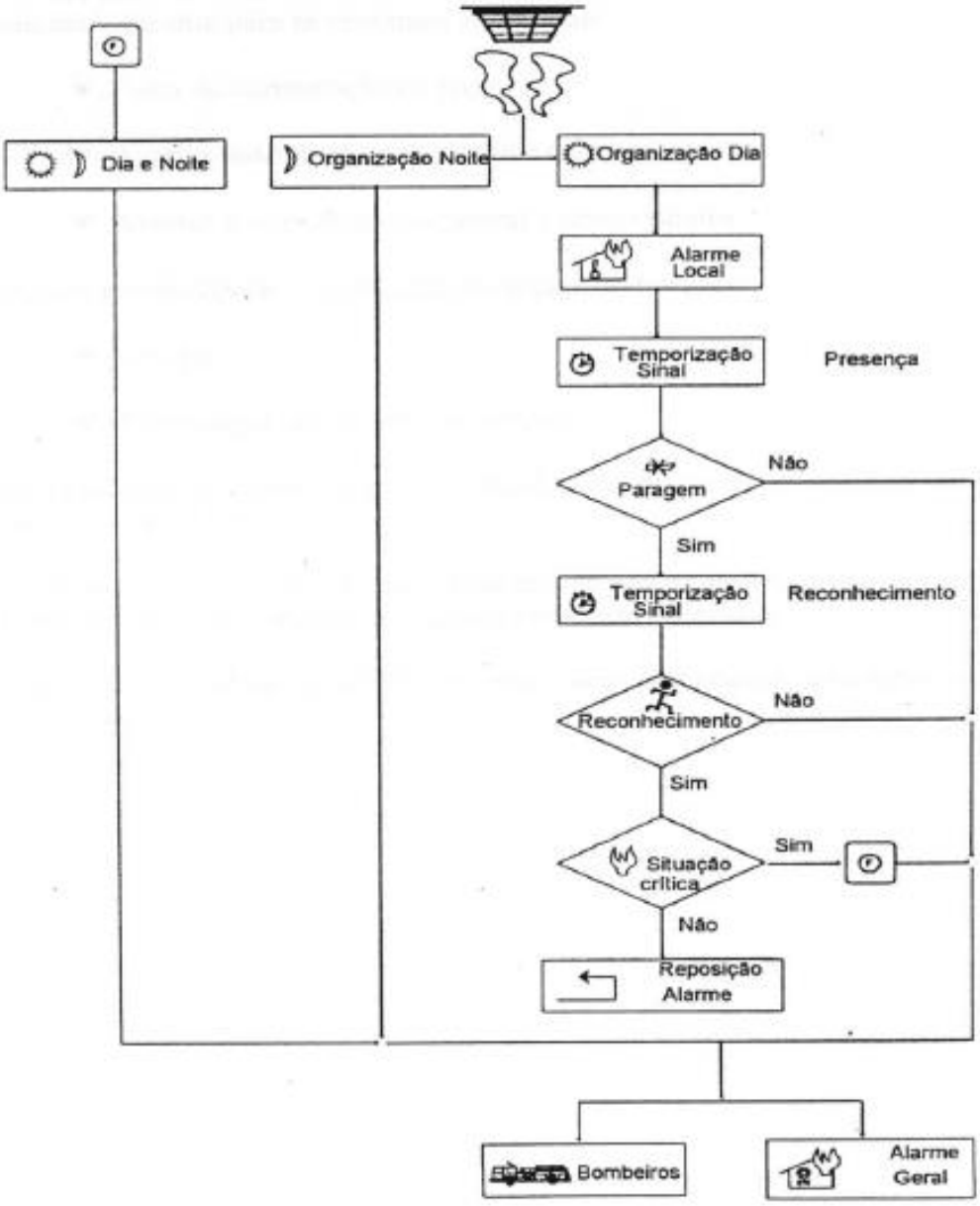


Figura 3-5 – Fluxograma da organização de alarme [33]

Tais sistemas endereçáveis têm muitos recursos e funções [30]: Detector endereçável, botoneiras de emergência, sinais que são identificados individualmente pelo painel de controlo:

- Os detectores endereçáveis devem ser conectados através de duas conexões de tal forma que a conexão do ambos os fios formam um laço, a fim de fornecer integridade ao circuito. Com essa redundância, caso o sistema sofra uma interrupção na alimentação por um dos lados, o outro irá alimentar mantendo todas conexões ativas;

- O detector endereçável é endereçado através de *software* de endereçamento ou dip switch;
- A técnica de comunicação multiplex permite cada detector sinalizar independentemente seu status de volta para o painel de controlo;
- O sistema endereçável também pode lidar com o dispositivo de saída no circuito de zona, onde o endereço pode ser uma instrução de comando para um dispositivo de saída (ligado/desligado), por exemplo: Módulo de sirene, comutação Relés liga/desliga;
- Módulo de interface é um dispositivo que é usado para interfacear o sistema de alarme convencional e o sistema endereçável.

Os sensores fornecem um sinal de saída analógico que representa o valor do fenômeno dos sentidos. A saída de um detector endereçável analógico é variável e é a representação proporcional do sentido efeito de fogo, fumo e chama. A transição desta saída é geralmente na forma de corrente para o painel de controlo que informa ao painel a que condição está a sala a ser monitorada. Para que um sistema endereçável analógico acione o alarme, o valor analógico de saída do detector deve estar na condição de alarme (acima do limite de alarme) para um período igual ao tempo necessário para completar três seqüências de endereços sucessivas.

O sistema endereçável tem algumas vantagens, além da grande possibilidade de monitorizar o sistema mais detalhadamente, possui instalação fácil e precisão. Mas historicamente, os sistemas endereçáveis tendiam a ser complicados de configurar e custavam entre 50% a 100% mais do que um convencional equivalente, portanto, poderíamos tomar isso como uma desvantagem em comparação com um sistema convencional [9].

3.3 Rede Chameleon

A Rede Chameleon é um sistema formado por vários painéis que atuam de forma independente, mas que comunicam entre si e compartilham a configuração “causa-efeito”, como um sistema global.

Cada vez que um evento ocorre em um painel, o painel encaminha esse evento por todas as portas de comunicação da rede, transmitindo esse evento na rede. Quando um evento é recebido em qualquer porta de comunicação, o painel regista o evento em seu "Registro de Eventos" local e executa uma causa/efeito definida para esse evento.

3.3.1 Painéis da Rede Chameleon

Os painéis que fazem parte da Rede Chameleon são: OCTO+; GEKKO; G-ONE; NODE+; e Chameleon REP.

As redes devem sempre ser projetadas em uma topologia de *loop* fechado. Conexões redundantes devem ser utilizadas, garantindo um mínimo de duas opções de caminho em caso de falha de comunicação, como pode-se observar na Figura 3-6.

No caso de conectores não utilizados, podem ser utilizadas derivações de rede e topologias cabeadas "mesh". Mesmo assim, o exemplo de *loop* fechado é o mais comumente usado e recomendado.

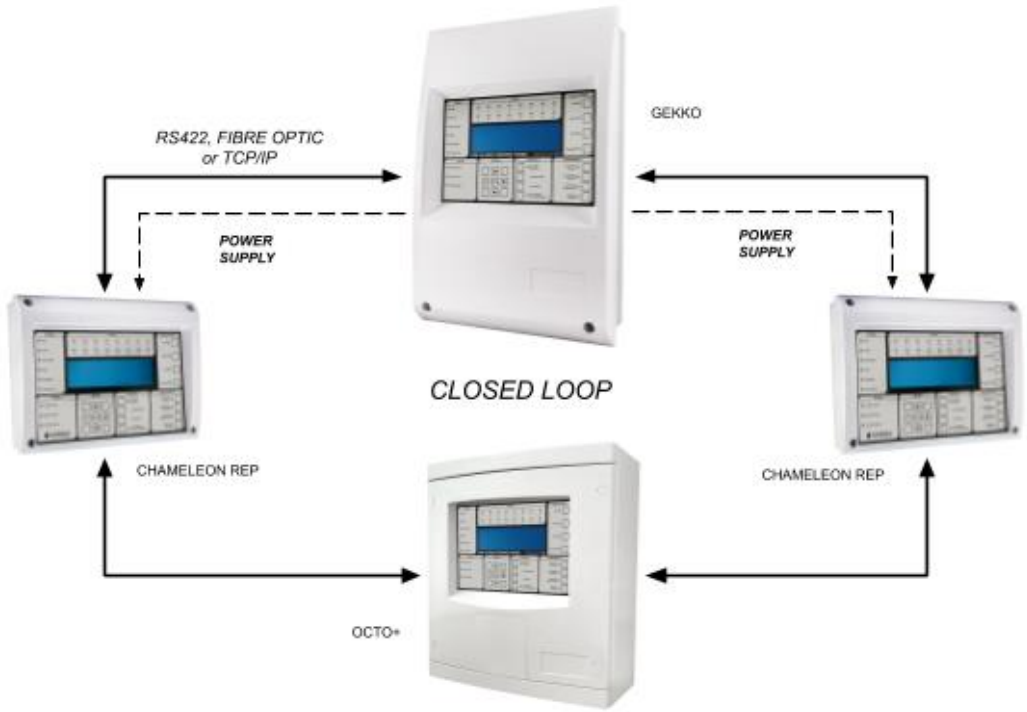


Figura 3-6 - Exemplo de rede de loop fechado - Rede de quatro painéis com OCTO+, GEKKO e CHAMELEON REP em uma rede de loop fechado.

3.3.2 Características do *Hardware*

Na Figura 3-7 abaixo pode-se ver o esquemático simplificado referente ao *hardware* do painel endereçável Gekko, que pertence a rede de comunicação Chameleon, onde o mesmo será o foco de estudo neste relatório.

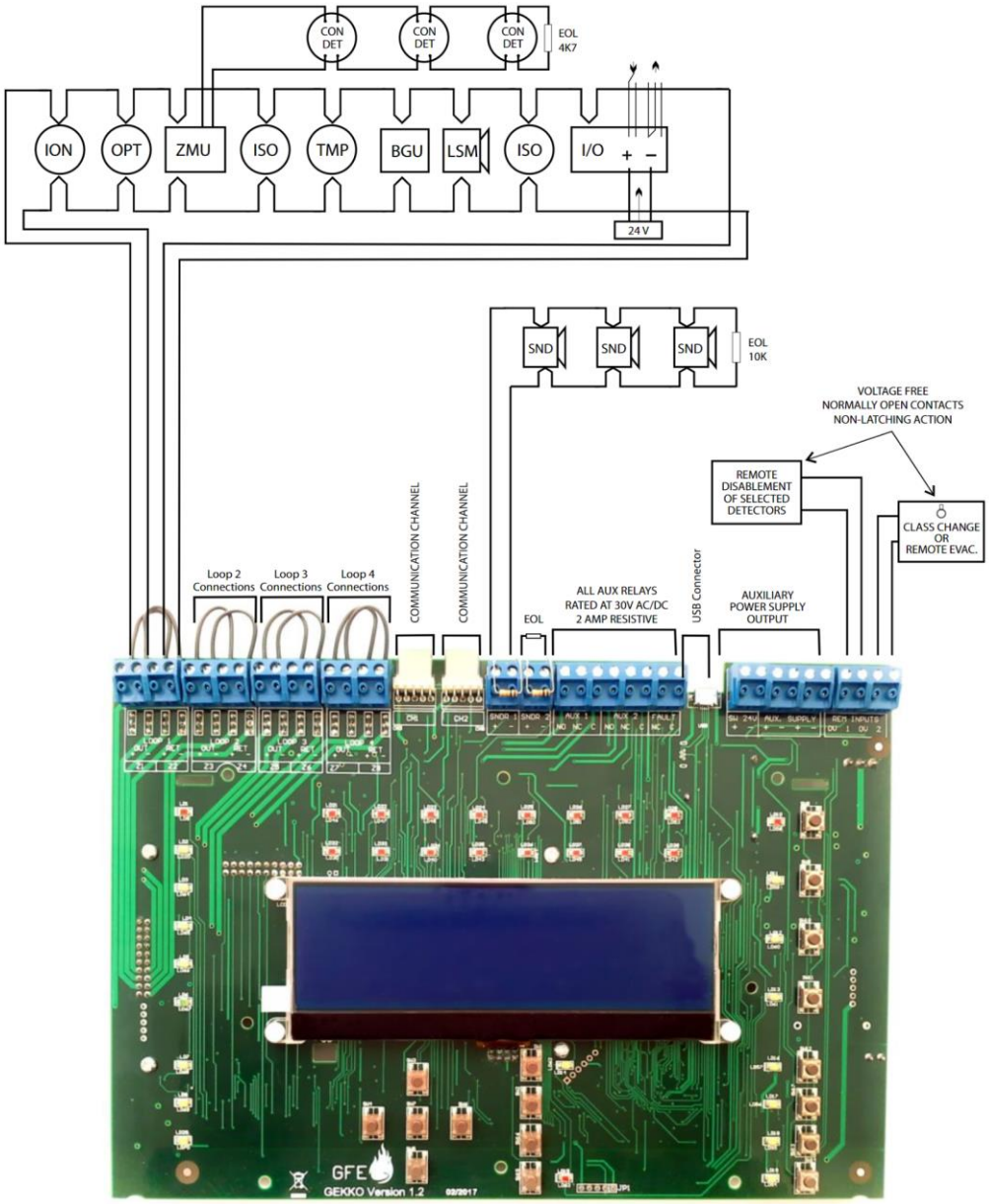
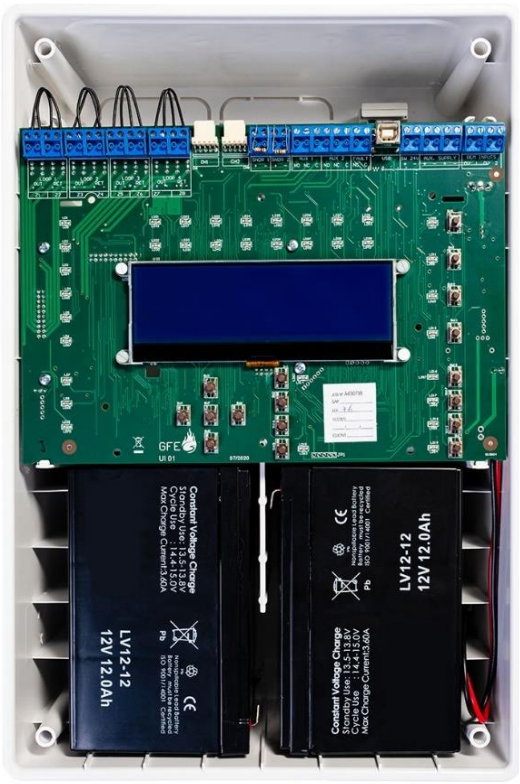


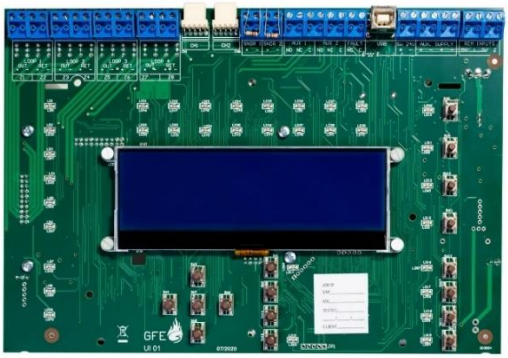
Figura 3-7 – Típico Esquemático do painel Gekko

As características do painel presente na Figura 3-8, Figura 3-9 e Figura 3-10 são:

- O painel pode ter de 1 a 4 laços;

- Suporta conexão com repetidores via RS485, fibra óptica ou TCP/IP;
- Pode comportar até 250 dispositivos por loop (sendo 125 são endereçados individualmente);
- Em relação as sirenes, pode-se ter até 96 sirenes de base de corrente ultrabaixa VULCAN 2 (endereçáveis) (limite de 32 endereços), além de 32 endereços de sirene programáveis individualmente;
- Possui 2 relés de saída de incêndio (comutação) e 1 relé de falha (NF - abre em falha), 2 saídas de alarme convencionais (programáveis individualmente);
- Ambos os loops de detecção possuem redundância de ligação para ser possível o monitoramento quanto à integridade. São 384 zonas totalmente programáveis;
- Tem 512 grupos de sirenes totalmente programáveis e 512 grupos de E/S;
- O registro de eventos chega a 10000 entradas;
- Disponível apenas com protocolo GFE – Específico GlobalFire Equipment;
- Tem um visor LCD gráfico retroiluminado de 240x64 pixels;
- Programação por teclado integrado ou *software* Loader PC;
- Suporte a vários idiomas (menu selecionável);
- Indicação de zona de incêndio LED de 16 zonas integrada;





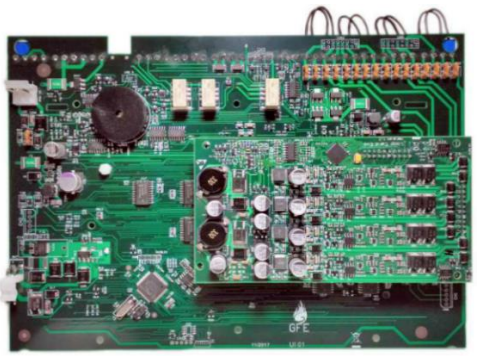


Figura 3-8 - Vista geral a placa referente ao Hardware do painel GEKKO

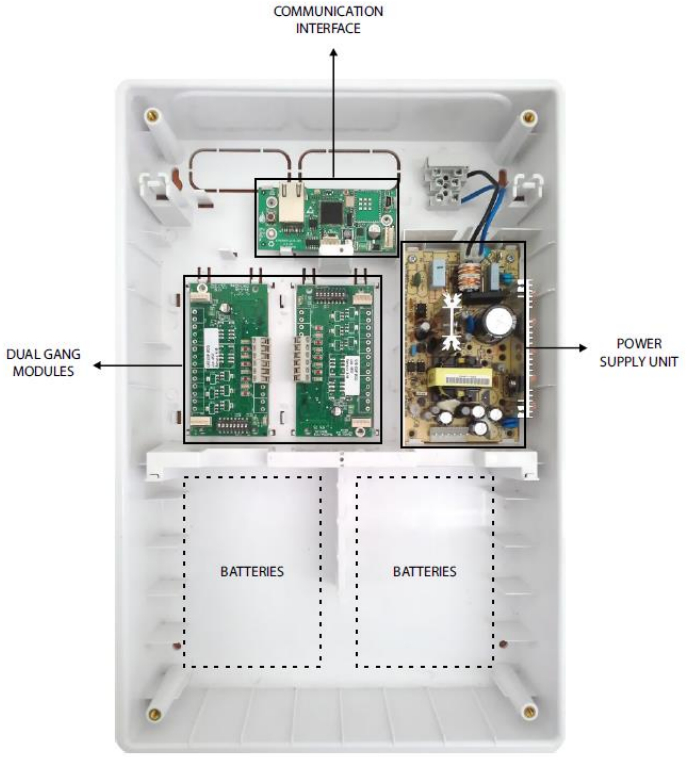


Figura 3-9 - Identificação de alguns componentes presentes na caixa da Gekko

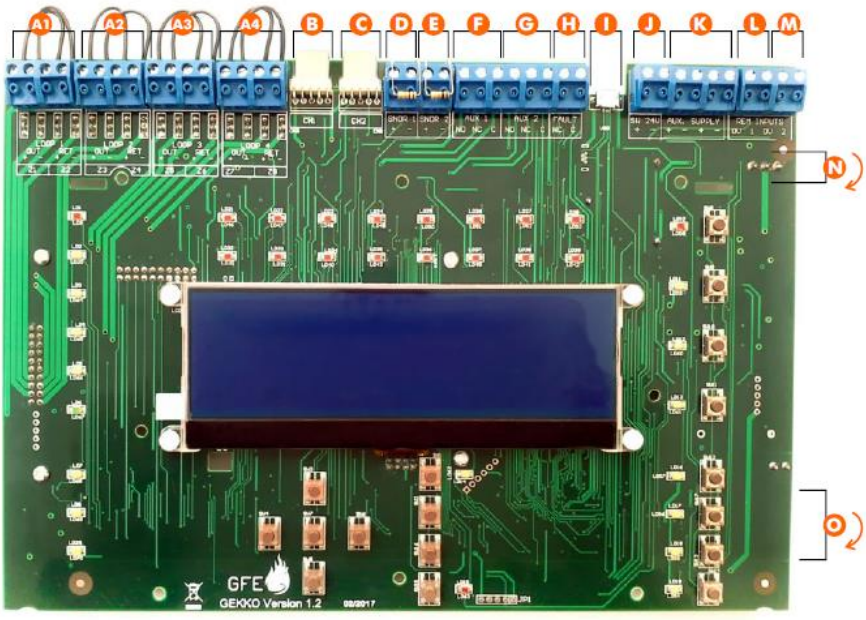


Figura 3-10 – Painel principal – Definição das Conexões

A. Conexões de loop. A1 corresponde ao Loop 1, A2 ao Loop 2, A3 ao Loop 3 e A4 ao Loop 4;

B. Canal de comunicação;

C. Canal de comunicação;

D. Circuito de sirene convencional 1;

E. Circuito de sirene convencional 2;

F. Saída de relé de comutação auxiliar 1 (ativada por qualquer incêndio presente no sistema, desabilitada pelo botão frontal);

G. Saída de relé auxiliar de comutação 2 (Ativada por qualquer incêndio presente no sistema, desabilitada pelo botão frontal);

H. Contato de relé NF de falha (Ativado por qualquer falha presente no sistema, abre em falha);

I. Conector USB;

J. Alimentação auxiliar de 24V comutada (desliga por 15 segundos a cada evento de reset);

K. Saída de fonte de alimentação auxiliar de 24V para alimentar dispositivos externos. Potência máxima de 300mA limitada e monitorada;

L. Desativação remota de detectores selecionados;

M. Evacuação Remota ou Mudança de Classe;

N. Entrada de energia do sistema;

O. Conexão de bateria 24V.

3.3.3 Laços analógicos de deteção

Os Laços Analógicos fornecem a ligação a todos os dispositivos analógicos endereçáveis e sirenes alimentadas pelo laço. O Laço (anel) tem que estar completo para que o painel monitorize a sua integridade relativamente a um curto-circuito e circuitos abertos (Figura 3-11).

Os dispositivos que podem ser ligados no laço analógico incluem: detetores de fumo e térmicos, módulos de zona (ZMU), módulos de E/S, sirenes de laço e botoneiras manuais.

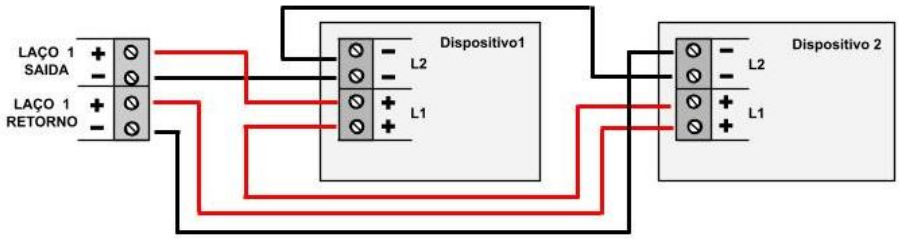


Figura 3-11 – Conexão de dispositivos ao laço analógico

Podem ser instaladas num laço analógico um máximo de 32 botoneiras manuais. Se este valor for excedido, o tempo de resposta para certos tipos de botoneiras poderá ser superior.

Para manter a integridade dos caminhos de transmissão, recomenda-se a utilização de isoladores de curto-circuito no Laço Analógico. Esta aplicação está diretamente relacionada com o layout físico da instalação e deve ser aplicada em conformidade de forma a limitar as consequências das avarias no Laço Analógico.

As consequências aceites das avarias que utilizam isoladores de curto-circuito são especificadas nas diretrizes nacionais de planeamento, concepção e instalação de sistemas de deteção de incêndios (códigos de prática), etc. e podem ser diferentes em determinados países.

Recomenda-se que nunca exceda 32 dispositivos de deteção no mesmo troço de cabo ou zona sem a utilização de isoladores de curto-circuito. O que significa que, em caso de curto-circuito, não devem ser afetados mais de 32 dispositivos de deteção.

3.3.4 Sirenes convencionais

Sirenes convencionais é o termo indicado para descrever as sirenes de sinalização acústica (ou sirenes polarizadas) ligadas diretamente ao painel. As sirenes analógicas são diferentes e estão ligadas no Laço Analógico.

No painel existem dois circuitos de sirenes convencionais. Em cada circuito podem ser ligadas mais de uma sirene. A saída de corrente máxima é de 500mA @ 28,5V DC nominal, para ambos os circuitos.

Estas saídas são monitorizadas relativamente a curto-circuitos e circuitos abertos. Se algum circuito de sirenes convencionais não for utilizado, deve ligar uma resistência de 10kohm nos terminais dessa saída.

A carga de corrente dos laços de deteção, circuitos de sirenes e saídas auxiliares de alimentação não deve exceder o limite máximo de corrente do painel.

3.3.5 Relés de fogo auxiliares e relé de avaria

São fornecidas na placa GEKKO duas saídas de relés auxiliares de fogo, estão identificadas como AUX1 e AUX2. Estas saídas são ativadas quando é detectado um fogo. Sob a presença de qualquer condição de fogo, estes 2 relés serão atracados. Ambos os conjuntos de contatos mudam de estado. A corrente máxima atual do contacto para cada conjunto de relé é de 2 Amp @ 30V DC resistiva / 0.5 Amp @ 120V AC resistiva.

Também é fornecida uma saída de relé auxiliar de avaria. Esta saída permanecerá fechada enquanto não houver avarias no sistema. Sob qualquer condição de avaria, o relé será desatracado e o contacto do relé será aberto. O relé de avaria é normalmente fechado, e abrirá em qualquer avaria no sistema. As classificações de contacto são: 2 Amp @ 30V DC resistiva / 0.5 Amp @ 120V AC resistiva.

3.3.6 Detalhes das baterias

As baterias estão ligadas à placa do circuito GEKKO (Figura 3-12). Esta ligação não só alimenta o painel em caso de avaria da alimentação primária como também carrega e mantém as baterias totalmente carregadas. Para ligar as baterias deve-se verificar a tensão nos terminais das baterias, esse valor deve ser de 27,5V DC +/–0,5V.

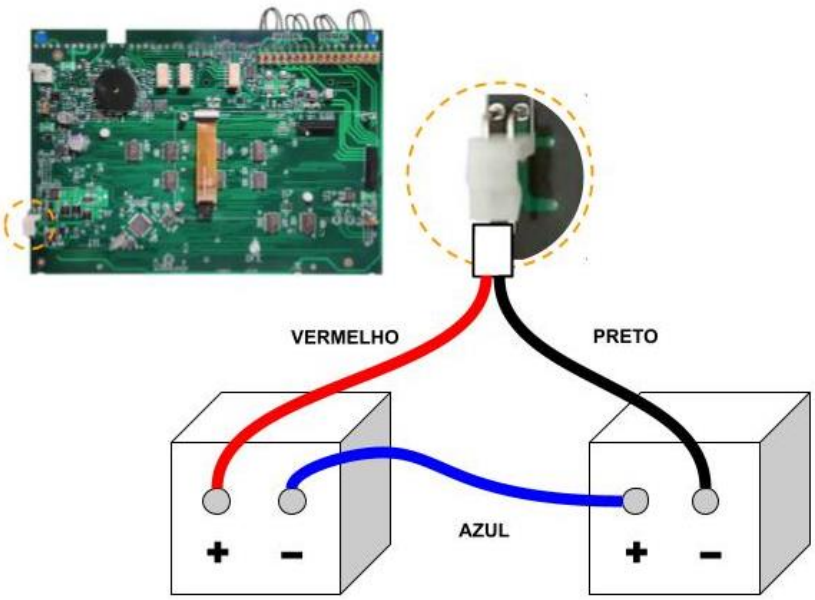


Figura 3-12 – Conexão das baterias ao hardware (GEKKO)

3.3.7 Características do *Software*

3.3.7.1 Bootloader

O *bootloader* é usado para atualizar o *firmware* em um dispositivo de destino sem a necessidade de um programador ou depurador externo.

Os painéis de controlo da evolução possuem um aplicativo *bootloader* permanentemente armazenado em uma área controlada do flash interno do microcontrolador principal. O programa *bootloader* não pode ser atualizado em campo, ou seja, ele é programado apenas na fábrica usando um programador adequado.

O código do *bootloader* começa a ser executado em um dispositivo Reset. Se não houver condições para entrar no modo de atualização de *firmware*, o *bootloader* inicia a execução do código do aplicativo.

As condições que levarão o *bootloader* a entrar em modo de atualização é a condição de trigger que pode ser o pressionamento de um switch durante a reinicialização do dispositivo, ou se não houver aplicação válida na área de aplicação do flash. Quando

em modo de atualização, a comunicação entre o aplicativo do PC e o *bootloader* deve seguir o protocolo de atualização.

3.3.7.2 Arquivo de configuração

Para instruir o aplicativo do PC sobre o que carregar no *hardware*, através do *bootloader*, existe um arquivo designado 'Arquivo de configuração' que possui uma estrutura semelhante a xml contendo todas as informações / dados necessários para concluir o processo.

O arquivo de configuração é criado automaticamente, pelo IDE (*Integrated Development Environment*), após o processo de compilação, por meio de um script dedicado.

3.3.8 GFE Connector

Chameleon Connector é o *software* que completa a gama de painéis Chameleon da GFE. Este *software* permite que o usuário configure todas as configurações de causa/efeito e carregue-as no painel analógico. O usuário também pode fazer download das configurações do painel e atualize as configurações conforme necessário. Na Figura 3-13 – Aplicação Chameleon Connector Figura 3-13 tem-se a aparência da aplicação Chameleon Connector.

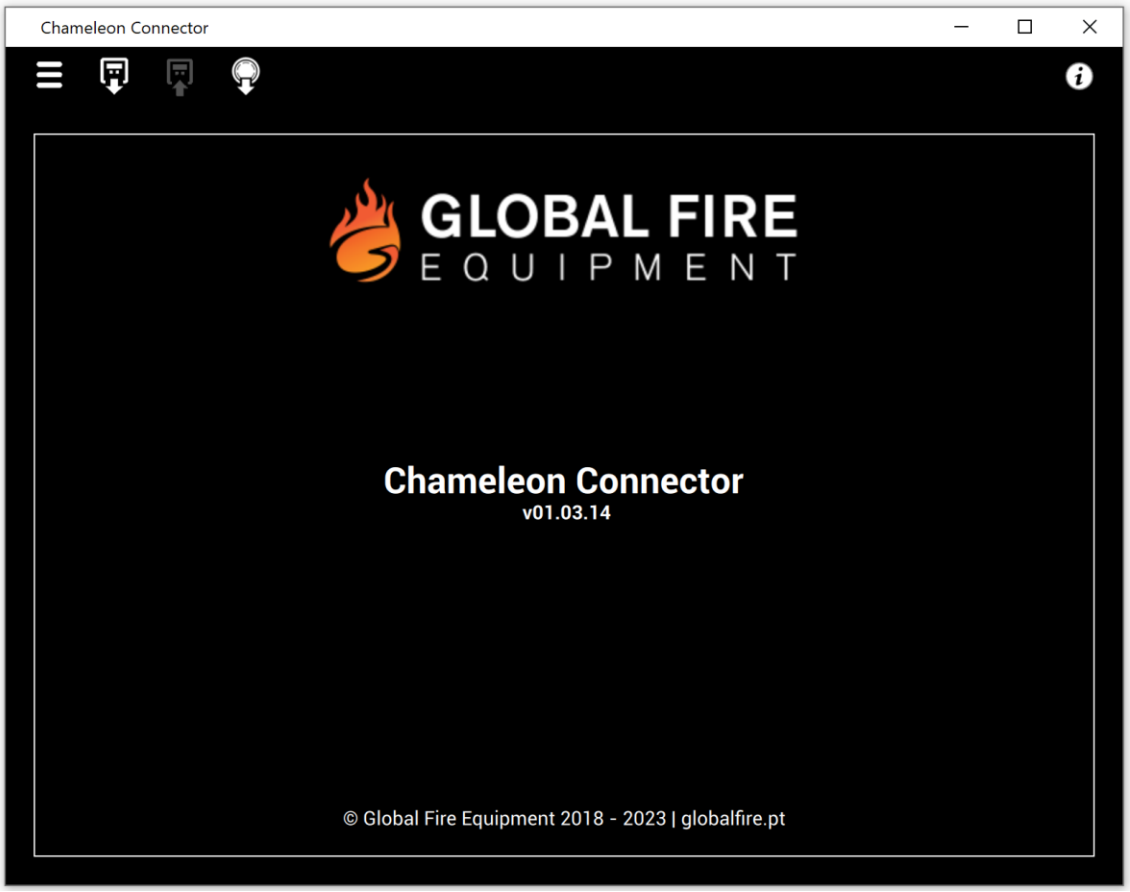


Figura 3-13 – Aplicação Chameleon Connector

O Connector é de extrema importância para o funcionamento do sistema, pois é ele que possibilita o usuário (Cliente) realizar a atualização de *firmware* do painel de controlo. Além de ser utilizado também para programação de projetos, configurações e informações que devam ser enviadas para o painel. E também ler as configurações e dispositivos que já estejam conectados ao painel.

Realizar testes no sistema também envolve a programação e utilização do Connector.

Na

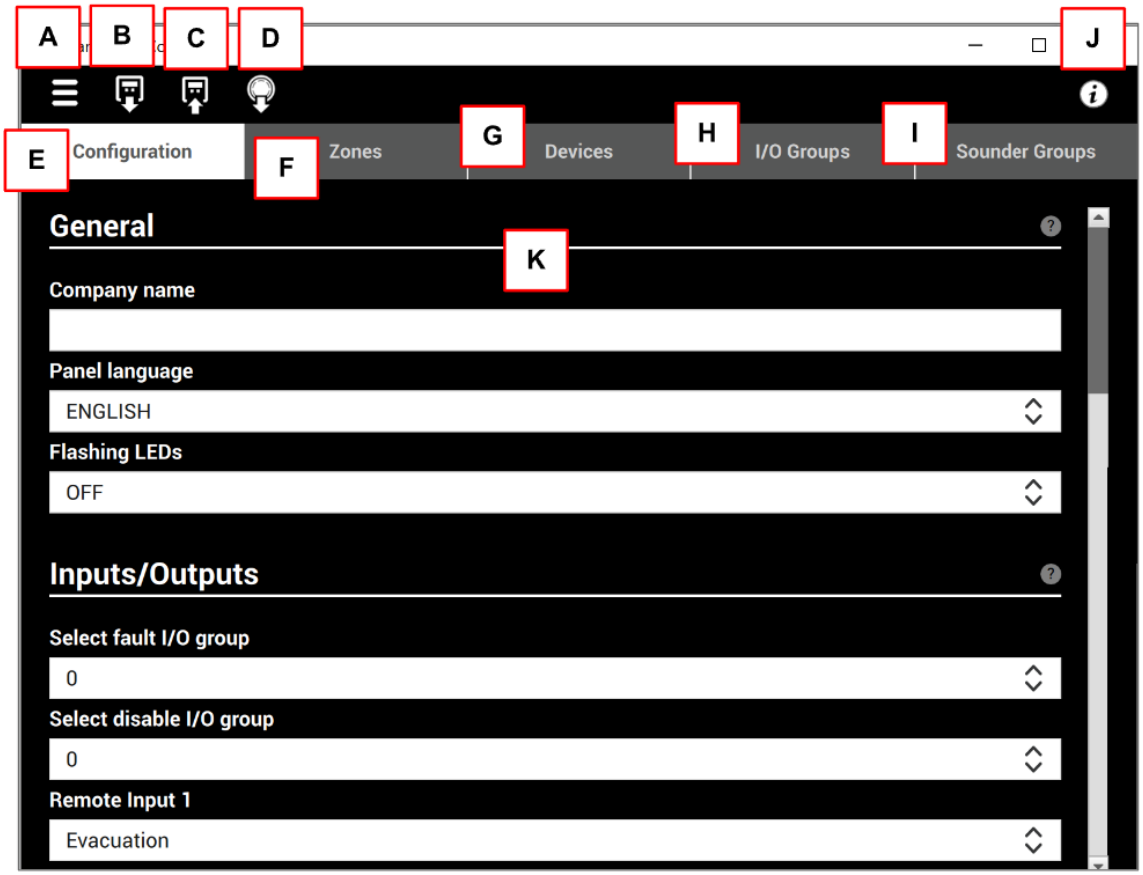


Figura 3-14 tem-se a indicação através das letras de “A” até “K” referente aos Menus de configuração utilizados no Chameleon Connector. Na Tabela 1 tem-se a descrição dos mesmos.

A janela do aplicativo é dividida em 3 partes:

- Barras de menu e de tarefas, onde podem ser executadas ações de upload e download, bem como criar novas configurações ou carregar arquivos de configuração ou firmware.
- Barra de abas, onde o usuário pode navegar entre as diferentes seções de configuração selecionando a aba desejada.
- Área de conteúdo, onde podem ser editados os dados de cada aba [32].

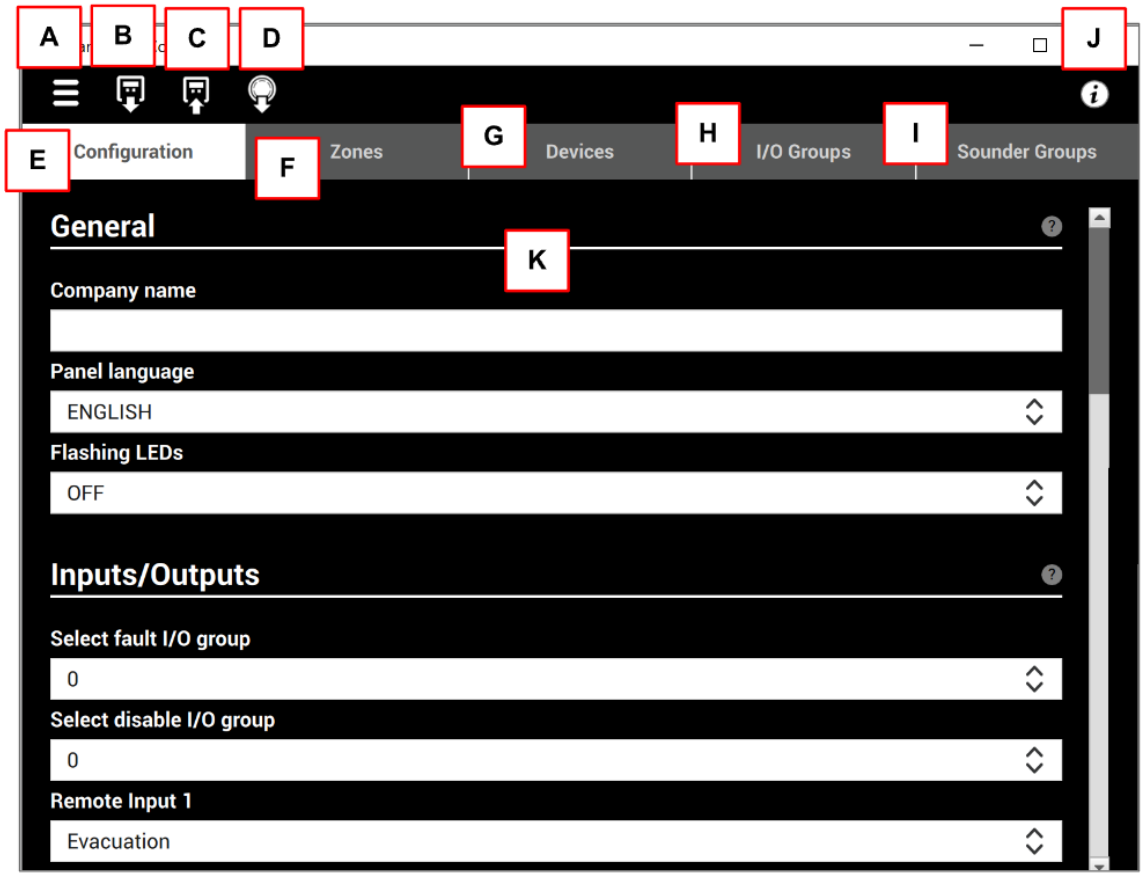


Figura 3-14 – Menu e Configurações do Chameleon Connector [32]

A	***MENU OPTIONS***	Selecionar o idioma, Novos arquivos, Carregue arquivos ou obtenha o registro do painel
B	***GET CONFIG FROM PANEL***	Baixar as configurações de causa e efeito do painel
C	***SEND CONFIG TO PANEL***	Carregar configurações de causa e efeito para o painel
D	***GET DEVICES DATA***	Obter informações dos dispositivos: tipo, valor e condição
E	***CONFIGURATION***	Configuração Geral
F	***ZONE***	Selecionar Zonas
G	***DEVICES***	Comportamento dos Dispositivos
H	***I/O GROUPS***	Configuração de Grupos de I/O's
I	***SOUNDER GROUPS***	Configuração de Grupo de sirenes
J	***INFORMATION***	Informação do CHAMELEON CONNECTOR
K	***CONTENT AREA***	Área de configuração de dados

Tabela 1 – Descrição dos Menu's de configuração do Chameleon Connector [32]

Através do Chameleon Connector é possível atualizar o *Firmware* dos painéis Chameleon. E a atualização é feita da seguinte forma:

- Deve-se verificar se o cano USB está conectado corretamente entre o computador e painel;
- Os cabos de rede ou comunicação devem ser todos desconectados (desligar o painel antes de remover os cabos);
- O painel deve estar desbloqueado no nível de acesso do instalador (ou superior);
- O painel deve estar no modo de instalação (menu 8-4-1)

Capítulo **4**

4 Implementação de Testes para Sistemas Embutidos de Alarme e Proteção contra incêndios

Os testes abordados nesse trabalho são implementados para serem realizados pelo usuário final. O sector de Pesquisa e Desenvolvimento responsável pela implementação destes testes não está localizado na fábrica, onde são produzidos os dispositivos. Após o desenvolvimento e implementação no código dos testes é gerada uma nova versão de *firmware* responsável pela programação dos painéis de controlo. Esta versão, programa os painéis na fase final de fabricação, para em seguida ir para o usuário final, onde este irá realizar os testes quando desejar ou for necessário. Os testes que são realizados na fábrica são em sua maioria testes estruturais e offline.

4.1 Procedimentos de Testes Funcionais Primários ao Sistema

4.1.1 Teste básicos de inicialização do Sistema (Teste Estrutural)

1. Certificar de que o cartão de laços esteja no lugar certo;
2. Conectar o sistema do painel;

3. Certificar de que o Painel esteja no Modo de Instalação (LED SYSTEM ON a piscar). Caso contrário, entrar no modo de programação e selecionar a função 8-4-1 Active/Installation Mode e colocar o painel no modo de instalação;
4. Pressionar Reinicialização do Sistema;

4.1.2 Teste de Comunicação entre Painéis (Teste Funcional e Online)

Observar e confirmar se os Painéis estão a comunicar na rede (verificar se todos os painéis estão a mostrar informações idênticas). Verificar se se pode ver no menu "8-5-2 Known Panels" toda a rede. Repetir este procedimento em todos os painéis do sistema.

4.1.3 Teste de verificação da presença de dispositivos no laço (Teste Funcional e Online)

1. Digitar o código mestre ▲▲►▼▲;
2. Confirmar se o protocolo do painel está selecionado corretamente para permitir total compatibilidade com os dispositivos instalados;
3. Selecionar a função "0-1-2 Erase All Panels (Factory Reset!!)";
4. Sair do modo de programação;
5. Pressionar REINICIAR SISTEMA;
6. Aguardar 90 segundos para que o sistema saiba automaticamente quais dispositivos estão presentes e relate quaisquer falhas;
7. Uma REINICIALIZAÇÃO DO SISTEMA ou REINICIALIZAÇÃO MESTRE resulta em: Um período de desligamento do laço analógico de 8 segundos [reset]; Uma carga de laço analógico de 15 segundos; e o início da sondagem de laço, nesta ordem.
8. Revisar as falhas (usar a tecla FAULT QUEUE REVIEW se houver mais de uma). Anotar as mensagens e, em seguida, desligar a alimentação e retificar as falhas;
9. Ligar o sistema, deixar inicializar e entrar no modo de programação;
10. Selecionar a função 7-1 Device Count, Type e Value;

11. Usar ▲ ou ▼ para selecionar o dispositivo, confirmar se todos os dispositivos estão presentes. Se o cartão de laços instalado tiver mais de 1 laço, usar ► para percorrer os laços e verificar a presença e o funcionamento adequado de todos os dispositivos instalados neste laço. Isso também pode ser feito via Connector, clicando no botão “obter dados do dispositivo”;
12. Depois que todas as falhas forem eliminadas e o sistema estiver no Modo de Instalação por 120 segundos, o sistema poderá ser colocado no Modo Ativo;
13. Observar que não há um final claro para o modo de instalação porque o sistema está constantemente a procurar e aprender. No entanto, se o sistema for colocado no Modo Ativo e o Modo de Instalação não tiver tido tempo de identificar todos os componentes do sistema, o painel mostrará relatórios de erros relacionados a dispositivos inesperados.
14. Se os dispositivos forem removidos, substituídos ou adicionados, o Modo de instalação deve ser selecionado para que o sistema possa aprender uma nova configuração. Se você não fizer isso, o sistema relatará uma falha ou dispositivos ausentes.

4.1.4 Configuração Geral de Teste

Como rotina de teste é realizada uma configuração padrão no painel GEKKO e realiza-se os testes seguintes:

• Configura-se o um Painel GEKKO com endereço 2 no menu 8-5-1, e conecta-se os seguintes dispositivos no Laço de número 2:

- ÓPTICO DETECTOR L2D1; ÓPTICO DETECTOR L2D2; ÓPTICO DETECTOR L2D3; PONTO DE CHAMADA L2D4; PONTO DE CHAMADA L2D5; PONTO DE CHAMADA L2D6; SIRENE L2D94; SIRENE L2D95; SIRENE L2D96; E/S L2D9; E/S L2D10; E/S L2D1.

• Atribuir Grupos de E/S às Zonas (3-3)

- Zona : 1 , 1st stage#1 : 1 , 2nd stage: 2, Disable:3
- Zona : 2 , 1st stage#1 : 1 , 2nd stage: 2, Disable:3
- Zona : 3 , 1st stage#1 : 1 , 2nd stage: 2, Disable:3

• Configurar grupos de E/S (5-1)

- G1E1 P2L2D9; G2E1 P2L2D10; G3E1 P2L2D1;

• Atribuir Zona ao Dispositivo (3-4)

- P02L2D1 : Zona 1; P02L2D2: Zona 2; P02L2D3 : Zona 3; P02L2D4 : Zona 1; P02L2D5 : Zona 2; P02L2D6: Zona 3; P02L2D94 : Zona 1; P02L2D95 : Zona 2; P02L2D96 : Zona 3;

• Configurar grupos de sirenes (4-2)

- G1P2 L2 Endereço 94- P; G2P2 L2 Endereço 95-C e 96-C; G3P2 L2 Endereço 95-P e 96-C;

• Atribuir grupos de sirenes a zonas (3-2)

- Z1 Grupo de sirenes do 1º estágio: 1 Grupo de sirenes do 2º estágio:1
- Z2 Grupo de sirenes do 1º estágio: 2 Grupo de sirenes do 2º estágio:1
- Z3 Grupo de sirenes do 1º estágio: 3 Grupo de sirenes do 2º estágio:1

• Definir P2L2D10 - unidade de E/S ativa na evacuação (5-5)
• Definir P2L2D11 - Substituir atraso de E/S (5-6)
• Definir P2L2D3 - Evacuação imediata (5-6)
• Definir P2L2D4 - Inicia o temporizador de evacuação (8-1-5)
• Definir P2L2D5 - A ativação anula os atrasos (6-1-6)
• Definir P2L2D6 - Desativação seletiva (6-1-3)
• Definir "Modo de sirene" - Programado (4-1)
• Definir atrasos à noite “OFF” (8-1-3)
• Definir hora às 12:00 (8-1-1)
• Definir atraso de evacuação 00m 45s - Modo de dispositivo (8-1-4)
• Defina o atraso de E/S em 1m 00s (5-7)
• Definir atraso da sirene 00m 15s Modo global - Qualquer dispositivo (4-6)
• Ative o botão "Atrasos Ativos"

4.1.5 Testes de Rotina (Testes Funcionais e Offline)

Os testes funcionais são divididos de acordo com a ação realizada e a análise da resposta do sistema. Nos subtópicos de *4.1.5.1* a *4.1.5.5* pode-se observar alguns dos testes funcionais offline realizados no sistema Chameleon.

4.1.5.1 Simulação 1 (teste "Atrasos e Evacuação Imediata"):

Ação: Ativar P2L2D1

Resposta: Após o painel reportar o alarme;

1. Após 15 segundos, o endereço da sirene 94 é ativado em P;

2. Após 1 min, o Grupo 1 de E/S (I/O L2D9) será ativado.

Pressionar Reinicialização do Sistema.

Ação: 1. Ative P2L2D1; 2. Em seguida, ative P2L2D3;

Resposta: Após o painel reportar o alarme;

1. A Evacuação Imediata será ativada e isso acionará todas as Sirenes (atrasos predominantes);

2. e também ativar I/O L2D10 porque possui unidade I/O ativa na evacuação;

3. Após 1 min, o Grupo 1 de E/S (I/O L2D9) será ativado.

Pressionar Reinicialização do Sistema.

4.1.5.2 Simulação 2 (teste "Override I/O Delay"):

Ação: Em 6-1-2, desabilitar o dispositivo P2L2D2 de outro painel diferente do painel de número 02.

Resposta: A E/S L2D11 será ativada logo após (sem atraso).

Pressionar Reinicialização do Sistema.

4.1.5.3 Simulação 3 (teste "Inicia o temporizador de evacuação"):

Ação: Ative P2L2D2, em seguida ative P2L2D4

Resposta: 1. Ao ativar P2L2D2 15 segundos após a ativação do Grupo 2 de Sirenes (Sirene 95-C e 96-C);

2. e I/O L2D10 será ativado após 1min.

3. Com a ativação de P2L2D4 imediatamente após 15 segundos, o Grupo de Sirenes 1 (Sirene 94-P) também será ativado;

4. e o I/O L2D9 será ativado após 1min.

5. O P2L2D4 ativará o “Inicia o temporizador de evacuação” para que após 45 s todas as sirenes estejam em C.

6. E se 1 min não passou do I/O L2D10, ele será ativado em 45s.

Pressionar Reinicialização do Sistema

4.1.5.4 Simulação 4 (teste “Atrasos de Ativação Anulam”):

Ação: Ative P2L2D2 e, em seguida ative imediatamente P2L2D5

Responda: 1. Com a ativação do P2L2D2 todos os atrasos passaram a contar; 2. porém com a ativação logo após P2L2D5, fará com que os atrasos de Override.

3. Assim na 1ª etapa o tempo do Grupo de Sirene 2 (Sirene 95-C e 96-C) contará para ativar; 4. com a ativação de P2L2D5 no 2º estágio, o Grupo 2 de Sirenes será ativado imediatamente e o Grupo de Sirenes 1 (94-P e 96-C); 5. além de ativar imediatamente as I/O's L2D9 e L2D10.

Pressionar Reinicialização do Sistema

4.1.5.5 Simulação 5 (teste “Desativação seletiva”):

Ação: Ative o botão DETECTORES SELECIONADOS

Responda: 1. A E/S L2D11 será ativada logo em seguida, pois o dispositivo P2L2D6 foi desabilitado (Se o botão “DETECTORES SELECIONADOS” estiver desativado);

2. A E/S L2D11 também deve ser desativada posteriormente.

Pressionat Reinicialização do Sistema.

4.1.6 Teste dos comandos Odyssey no Chameleon

Para realizar testes nos comandos de um dos protocolos de comunicação do sistema Chameleon, utiliza-se o *Software* RealTerm como ferramenta. O Odyssey é um protocolo de comunicação criado pela empresa que possibilita a comunicação entre os painéis e usuário por meio de um computador. Através do RealTerm são enviados códigos dos comandos em formatação Decimal, e obtém-se a resposta em formatação hexadecimal como mostra a Figura 4-1, Figura 4-2 e Figura 4-3.

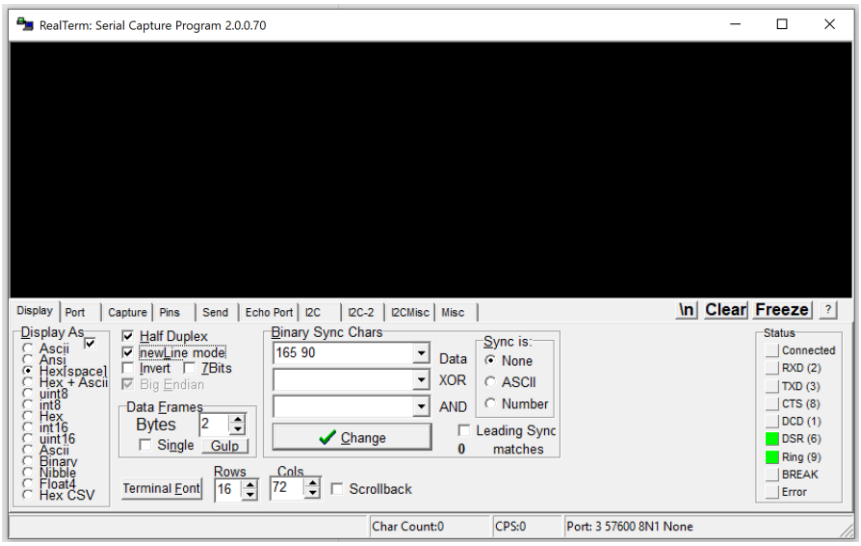


Figura 4-1 – Menu configuração do RealTerm: Serial Capture Program

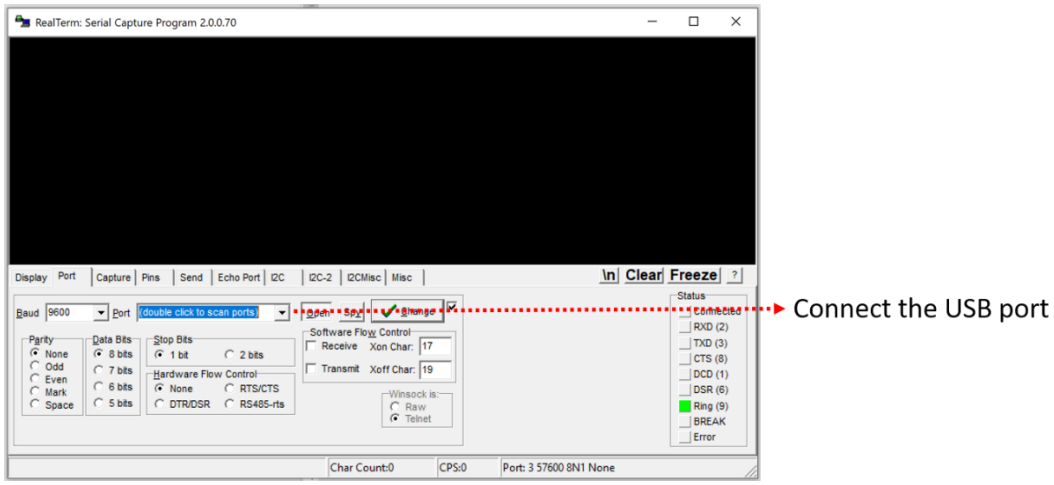


Figura 4-2 – Menu configuração conexão da porta (USB)

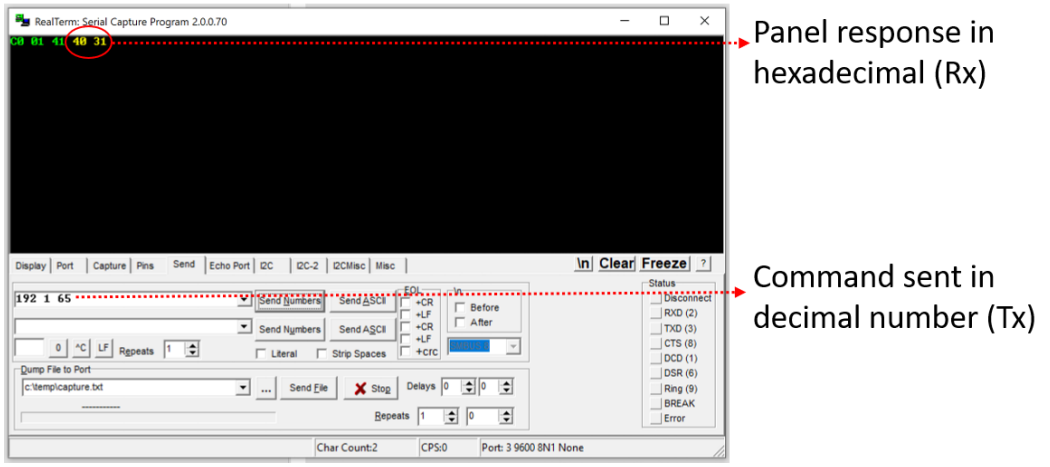


Figura 4-3 – Envio e Recebimento de comandos para GEKKO.

Com o envio de comandos certifica-se se desempenham a função solicitada e se o painel responde de forma coerente ao esperado.

4.2 Teste de Comunicação entre Painéis conectados em rede com Interface RS422 (Teste Funcional com painel conectado em rede)

Este teste visa certificar que a comunicação entre os painéis está a acontecer completamente. De forma a verificar se as configurações presentes em todos os painéis são iguais ou se há a necessidade de realizar o compartilhamento de configuração entre os painéis na rede através do *Broadcast* Menu 8-5-4. Na Figura 4-4 pode-se observar os meios de comunicação existentes.

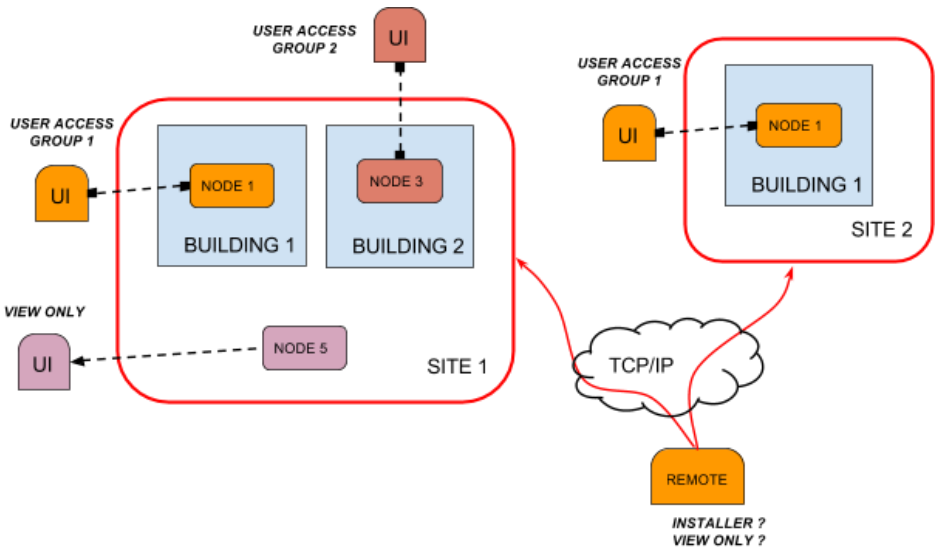


Figura 4-4 – Esquema de tipos de comunicação que ocorrem no sistema.

O menu referente a este teste encontra-se 8-5-2 (*Known Panels*), onde o usuário pode ativar o modo de teste de comunicação e certificar se a comunicação entre os painéis está realmente a acontecer. Para realizar o compartilhamento de informações entre os painéis tem-se o menu 8-5-4 *Broadcast Configuration*, caso ocorra algum erro de compartilhamento de configuração, ou os painéis sejam incompatíveis o próprio painel ativará um alerta de falha de comunicação, ou sinalizará que existem algum painel com as configurações diferentes dos outros painéis presentes na mesma rede de comunicação.

4.3 Teste de Monitorização de Bateria (Teste Funcional e Online)

A ideia principal é tornar os recursos de energia do sistema passíveis de serem gerenciados pelo sistema operacional via técnicas de monitoramento do seu consumo.

Para implementar este teste é necessário responder algumas perguntas e tomar algumas decisões acerca da maneira como ocorrerá a implementação do monitoramento. Qual será as características monitoráveis da bateria que são capazes de mostra ou calcular seu estado atual de tensão? Para responder a esta pergunta tem-se o conhecimento de qual bateria está a ser usada. Neste trabalho a bateria a ser utilizada é um conjunto de duas baterias do tipo DM12-2.2S (12V2.2AH/20MR). Na Figura 4-5 pode-se consultar o tipo de bateria a ser utilizada neste trabalho.

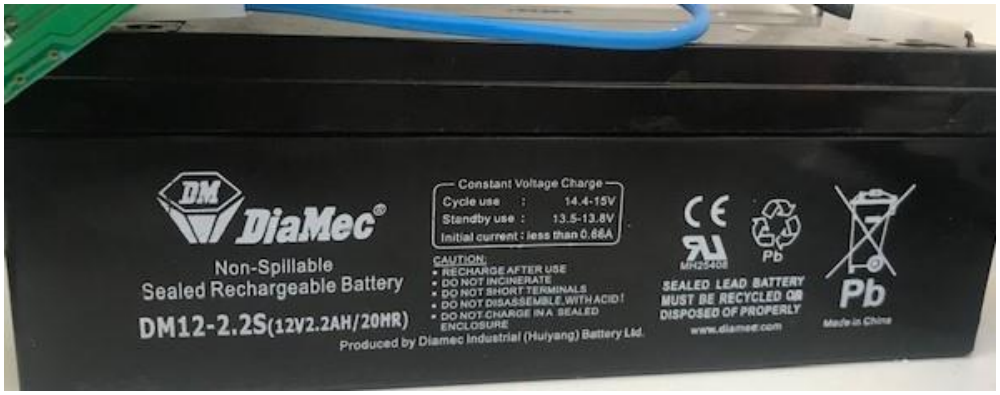


Figura 4-5 – Bateria do painel GEKKO

Como conceito fundamental, deve-se ter conhecimento que a bateria possui a características de ter sua tensão diminuída em decorrência de seu uso. E neste sistema embarcada de alarme e proteção contra incêndios possui um Conversor Analógico Digital (ADC) como periférico, e o acompanhamento (consulta) da tensão da bateria em um dado momento é feito através da utilização desse dispositivo por meio de leituras efetuadas em sua tensão e posteriormente a conversão para um valor digital utilizável pelo sistema.

Apesar de ser possível ter um constante acompanhamento de nível de tensão da bateria por meio do ADC, cada vez que se realiza a amostragem implica no consumo de energia do sistema por utilizar o periférico. Para minimizar as interferências do teste a implementação foi realizada com uma estrutura que permite acompanhar o consumo de

forma aproximada. Na Figura 4-6 pode-se ver o circuito utilizado para realizar o monitoramento da Bateria.

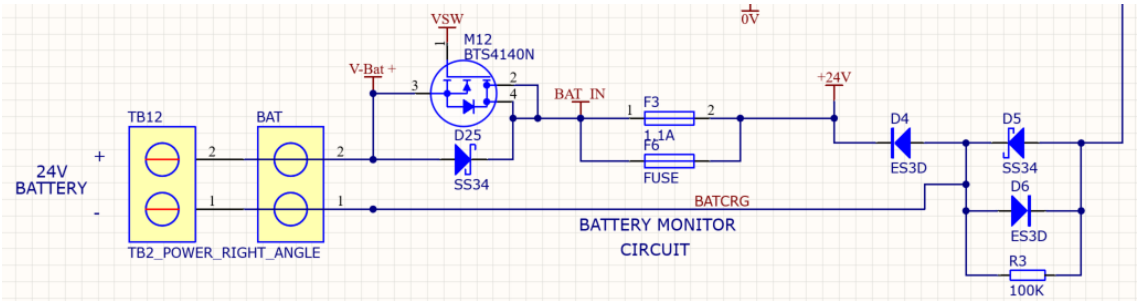


Figura 4-6 – Circuito de monitoramento de tensão da bateria

Foi implementado, a nível de código em linguagem C um Menu de acesso ao monitoramento da tensão da bateria. Para realizar o teste basta aceder ao Menu 7-8-1 do Painel Gekko e visualizar o nível de tensão das baterias conectadas ao painel. Na Figura 4-7 pode-se observar o Menu para aceder ao painel, e na Figura 4-8 a informação visualizada. A informação presente na Figura 4-8 é referente a tensão da bateria quando o painel está conectado a alimentação principal que é de 28V (ou seja, as baterias estão em modo de carregamento). Quando a alimentação principal do painel está desconectada, o painel passa a ser alimentado pelo conjunto de baterias e a tensão reportada é em torno de 24V.

Figura 4-7 – Ecrã do painel GEKKO no Menu de teste.

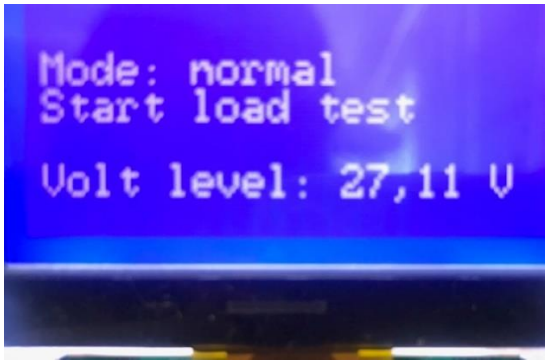


Figura 4-8 – Realização do Teste de bateria e consequente visualização da tensão monitorada

Tal teste desempenha grande importância pois os sistemas de alarme e proteção contra incêndios devem funcionar mesmo na ausência de energia que provem da rede elétrica de potência. Desta forma, pode-se monitorar de forma instantânea (o sistema realiza o teste de nível de tensão a cada 200 ms) o nível de tensão das baterias, além de se lançar um aviso para ser visualizado no ecrã de LCD quando o nível for menor que 24V (condições normais de funcionamento).

4.4 Auto-teste para falhas *non-latching*

Neste teste realizado automaticamente pelo painel existe a verificação da existência da falha no painel para aviso de falha ou retirada da mesma. A verificação é feita de forma periódica. No fluxograma da Figura 4-9 pode-se perceber a metodologia de funcionamento do auto-teste para falhas *non-latching.* Nele nota-se que há uma verificação periódica de falha a cada 500ms, esta é realizada no painel. Quando há a falha, o painel reporta as falhas de algumas formas: Através dos Leds que indicam falta; Fault quele; o alarme do painel liga (Buzzer); Led que indica avaria; mensagem de falha que aparece no visor LCD. O objetivo do deste auto-teste é retirar os avisos de falha quando não houver mais a falha no sistema.

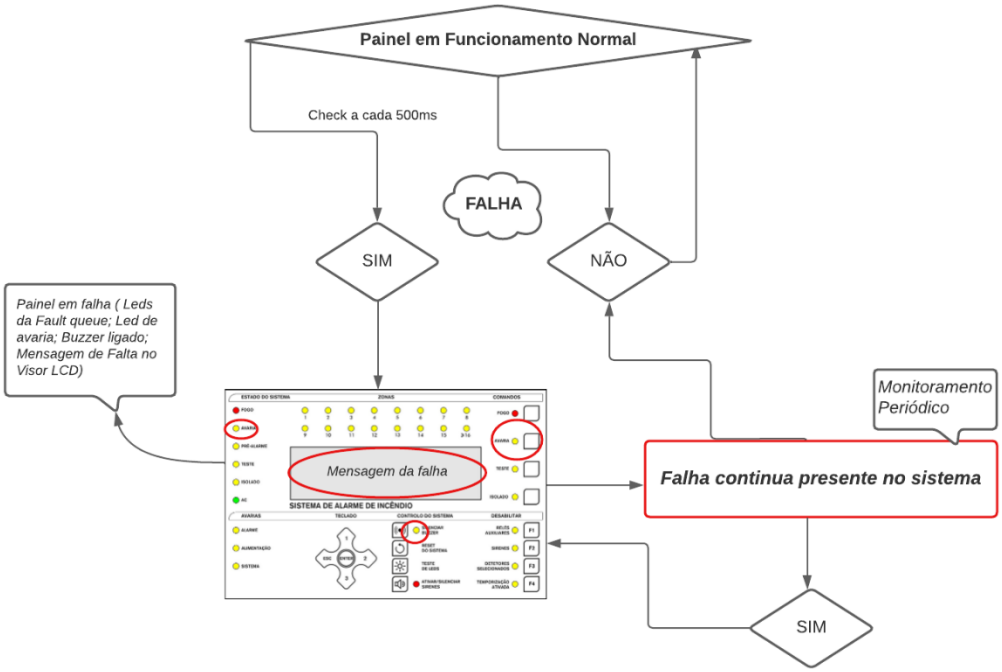


Figura 4-9 – Fluxograma do funcionamento do teste para falha Non-Latching

Na implementação deste teste automático foi acrescentado a função de recuperação (*void new_retrieve(void)*) pré-existente no código o reconhecimento da falha que supostamente está no painel. Deste modo o alerta de falha apenas deixará de existir se a falha que o causou também deixar de existir.

Também foi criado uma opção no Menu 0 (Flashvar) do tipo boleana que pode ser ativada ou desativada, de acordo com a necessidade do utilizador. Ou seja, se o utilizador precisar que as faltas funcionem no modo non-latching deve aceder ao Menu 0, procurar por "ac fault latch" e selecionar a opção "Falso". Por defeito o painel tem esta opção como "True".

No fluxograma da Figura 4-10 tem-se a representação genérica do funcionamento do código neste modo.

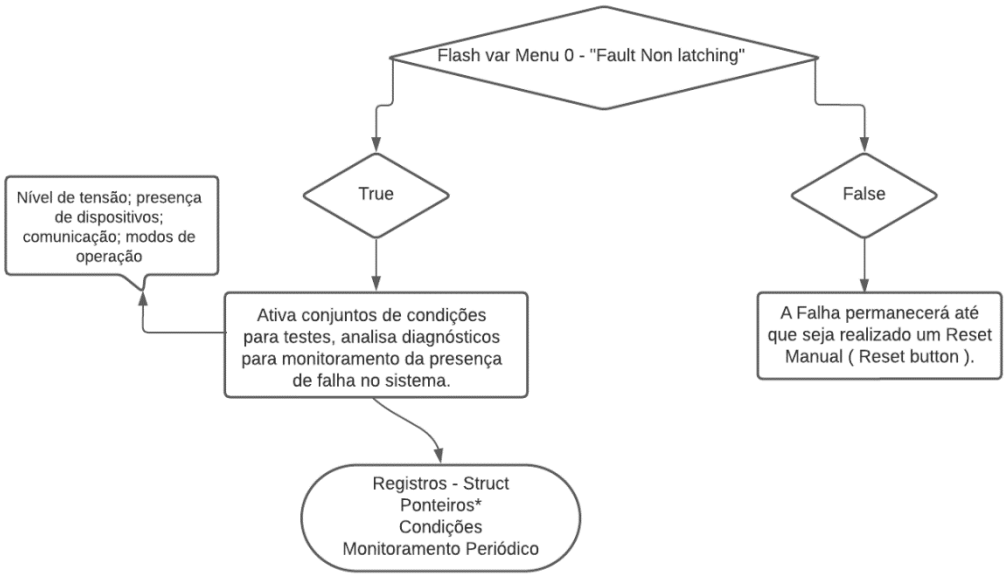


Figura 4-10 – Fluxograma das medidas implementadas no código

Ainda neste modo houve a criação de uma variável de registro do tipo de evento do Alarme sonoro. Isto porque o painel só deve sair do modo de falha se não houver a falha que o causou (e sempre levar em consideração as faltas anteriores e as seguintes). Importante: O painel não pode desabilitar o modo de falha caso exista um alarme de fogo.

Outra Flashvar também foi adicionada ao Menu 0 que auxilia no funcionamento deste teste. Que é a “delay before AC fault NOT EN54”, que tem como função ativar um contato de tempo quando ocorre uma queda de energia da rede de alimentação ao invés de ativar automaticamente todas as faltas no sistema. O objetivo é para lugares onde ocorrem faltas frequentes de energia por um curto período de tempo (até 4horas) não faça o painel passar o dia a ativar e desativar seus alarmes, confundindo o utilizador por achar se tratar de um fogo no sistema.

4.5 Implementação de Autotestes com testes de rotina (Teste Funcional e Online).

Os autotestes de rotina são referentes aos testes citados no ítem 4.1.5, e tem como objetivo tornar automático todo procedimento que executa 5 diferentes testes, são eles: teste de “Atrasos e Evacuação Imediata”; teste de “Override I/O Delay”; teste de “Inicia

o temporizador de evacuação"; teste de "Ativação de anulação de Atrasos"; teste de "Desativação seletiva". Na Figura 4-11 tem-se a representação básica, por meio de fluxograma, do funcionamento do autoteste.

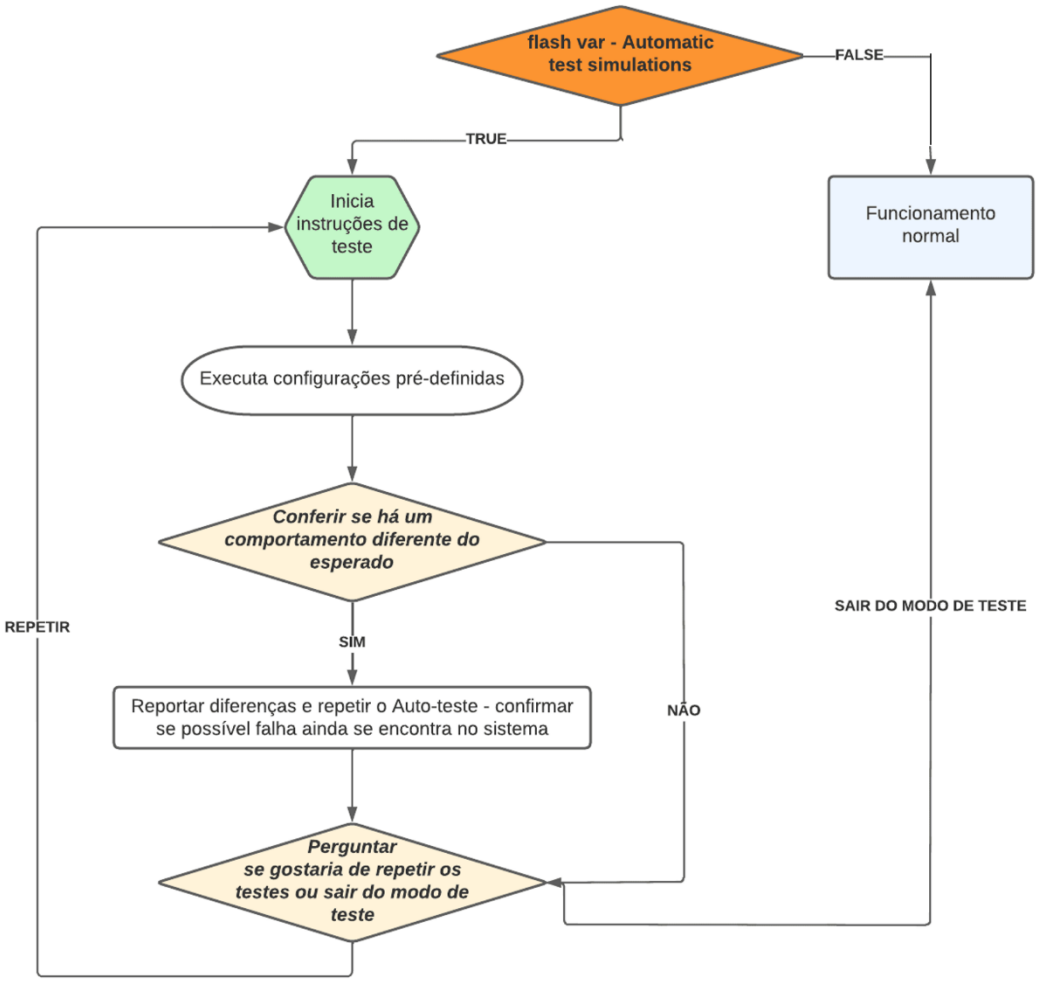


Figura 4-11 – Fluxograma de Funcionamento de Autotestes de rotina

4.6 Ferramenta eletrónica para automatização de testes em sistemas de alarme e proteção contra incêndios.

Para realizar os autotestes desenvolveu-se um dispositivo eletrónico, que possui o nome de *GFE Device Emulator*, que simula os dispositivos presentes no sistema. Além de manipular os valores analógicos reportados pelos dispositivos, através do protocolo de comunicação, ele também simula ações no sistema de forma simples por meio de um emulador.

O Device Emulator funciona por meio de uma porta serial conectada ao computador, ou seja, para correr o Device Emulator o computador deve ter conectado um cabo USB a um FTDI (Future Technology Devices International). O FTDI é um chip conversor USB – UART que faz uso dos pinos Tx e Rx para transmitir dados recebidos pela USB ou para transmitir dados para ela. A alimentação pode ser de 3,3 ou 5V, neste projecto a tensão de alimentação utilizada é de 3,3V. Desde modo, como este nível de tensão, o FTDI conta com regulador interno que disponibiliza essa tensão ao pino. Ressalta-se que para outro nível de tensão é necessário um regulador externo adicional.

Após a conexão do USB (no computador) ao FTDI, este é conectado ao dispositivo Device Emulator como pode-se observar na Figura 4-12. Nas Figura 4-13 e Figura 4-14 pode-se se ver uma imagem do FTDI utilizado neste projecto e do dispositivo Device Emulator desenvolvido para fins de autotestes.

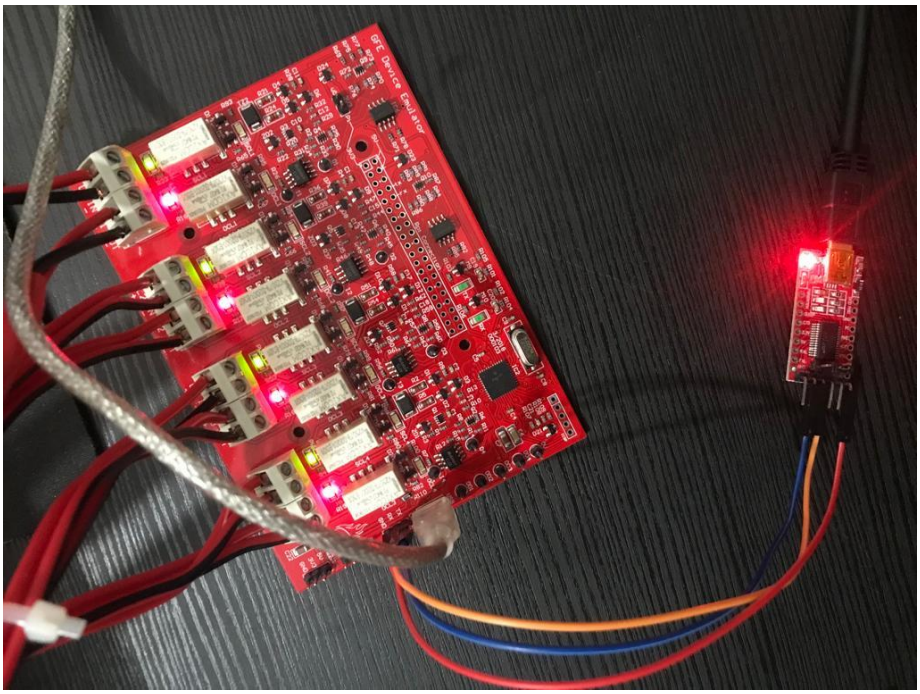


Figura 4-12 – Conexões entre o FTDI e o Device Emulator

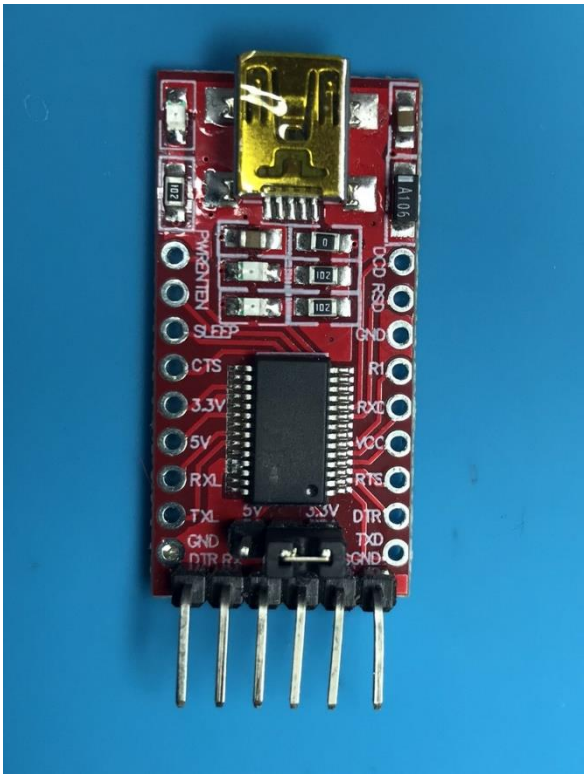


Figura 4-13 - FTDI (Future Technology Devices International).

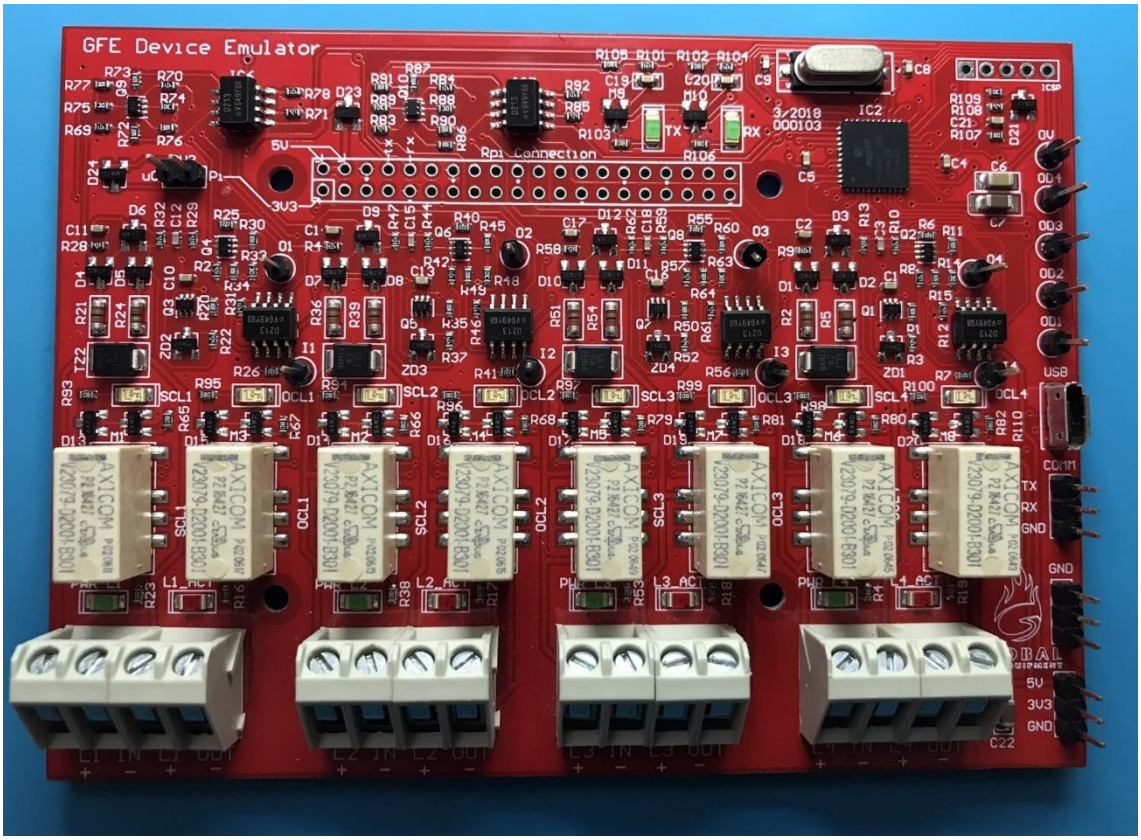


Figura 4-14 – GFE Device Emulator

Na Figura 4-15 também pode-se perceber a conexão do FTDI com o Device Emulator através do Rx, Tx e GND. O Rx do FTDI é conectado ao Tx do Device Emulator e o Tx do FTDI é conectado ao Rx do Device Emulator. O Tx e Rx dos dispositivos são pinos que são referentes as taxas de transmissão e recepção de dados, respectivamente.

A conexão do dispositivo com o painel de proteção e controle contra incêndios ocorre nas entradas e saídas do painel referentes aos 4 laços de conexão de dispositivos. O Device Emulator irá substituir toda necessidade de conectar vários dispositivos no laço para testar as funcionalidades do painel. Na Figura 4-16 pode-se perceber a conexão do dispositivo ao painel.

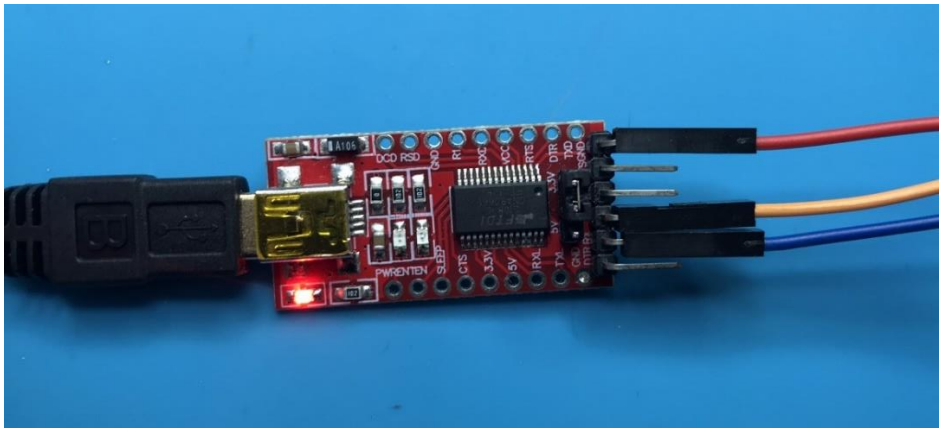


Figura 4-15 – Conexão USB no FTDI e pinos Rx, Tx e GND

Figura 4-16 – Conexão do Device Emulator ao Painel Gekko.

Um importante detalhe em relação a conexão é a alimentação da placa. Essa alimentação é feita por meio de um cabo USB (5V) que pode ser conectado a qualquer porta USB do computador.

O Device Emulator é um dispositivo em desenvolvimento que teve como inspiração um dispositivo que é comercializado pela GlobalFire I/O multi. O I/O multi é um dispositivo que contém Inputs e Outputs que podem ocupar vários endereços na central endereçável e sua função é realizar ativações em outros sistemas por meio de respostas do Sistema de Proteções de Incêndios. O Device Emulator começou a ser implementado por antigos funcionários da Global Fire equipament mas encontrava-se sem utilidade ativa atualmente.

4.6.1 Características do *Hardware*

Para detalhar todo funcionamento a nível de sistemas de testes, antes será mostrado algumas características do Device Emulator. A placa foi desenvolvida em 2018 (não foram encontrados registros mais precisos em relação ao seu desenvilmento, porém foi possível encontrar o projecto de *hardware* que pode ser aberto através do *software* de desenvolvimento - Altium.).

Na Figura 4-17 tem-se o projecto da placa de circuito impresso – PCB feita em duas camadas.

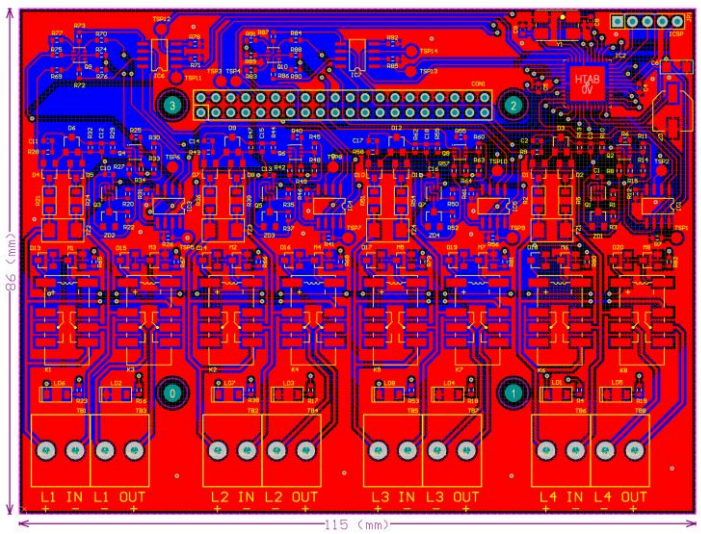


Figura 4-17 – Projecto para o circuito impresso (PCB) do Device Emulator (Arquivos GFE)

Na Figura 4-18 observa-se os circuitos referentes à conexão dos laços ao painel no dispositivo. Apesar de apenas ser mostrado um bloco de circuito referente ao Loop 1, existem 4 grupos iguais a este na placa, um para cada Loop. A entrada em L1A e L1B tem-se um circuito que permite a inversão do polo positivo e negativo do laço, não importando o sentido. A saída tem-se controlador de nível de tensão, no laço tem-se uma tensão de 29VDC em média, a saída do circuito terá 5V. Em LD6 tem-se um led verde que tem como função sinalizar a conexão do laço do painel no *device emulator*.

No circuito abaixo na mesma imagem, em IL213T, tem-se um opto acoplador analógico, que consiste em um LED de alto desempenho que ilumina dois foto díodos próximos. E está a ser utilizado para isolar sinais analógicos, trazendo mais flexibilidade e estabilidade do sinal analógico.

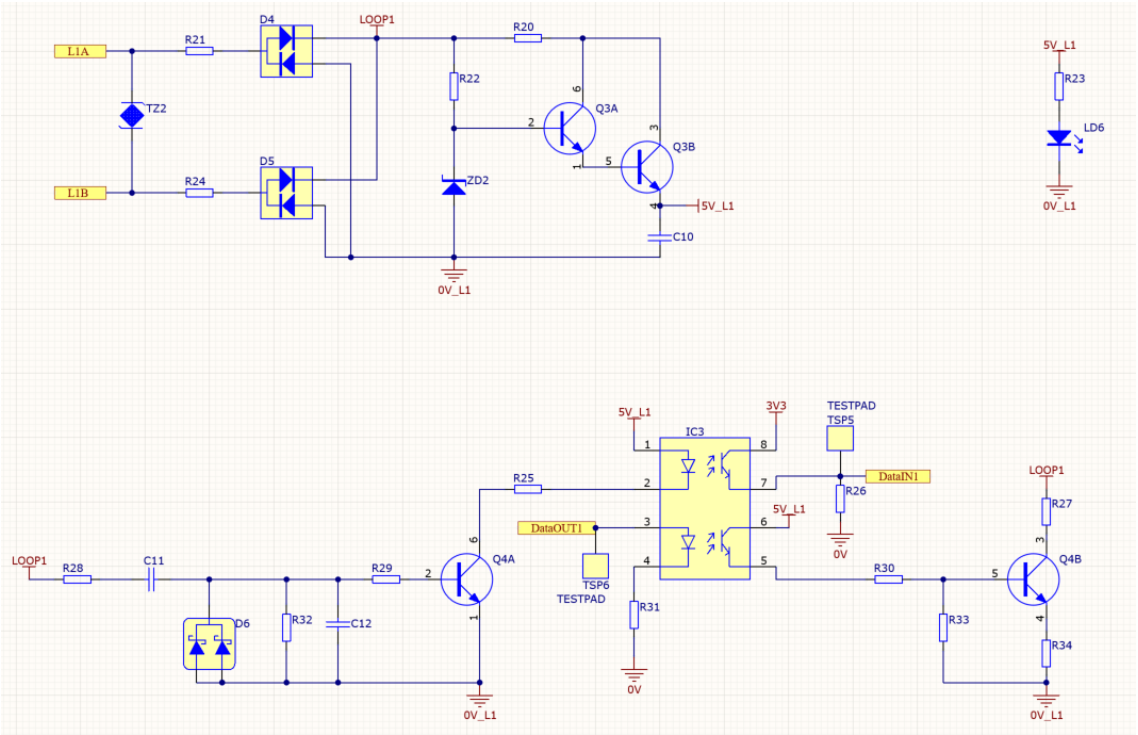


Figura 4-18 - Circuitos referentes a conexão dos Laços do Painel no dispositivo. Todos os Laços são iguais. (Arquivos GFE)

Os circuitos da Figura 4-19 são responsáveis pelos testes de Loop aberto e curto-circuito no loop. Semelhante a Figura 4-20, há um circuito por Loop, este é referente ao Loop 1. A ativação do relé K1 simula uma situação de curto-circuito no laço. Já se o relé K3 ativar, tem-se a simulação de um laço aberto (sem redundância).

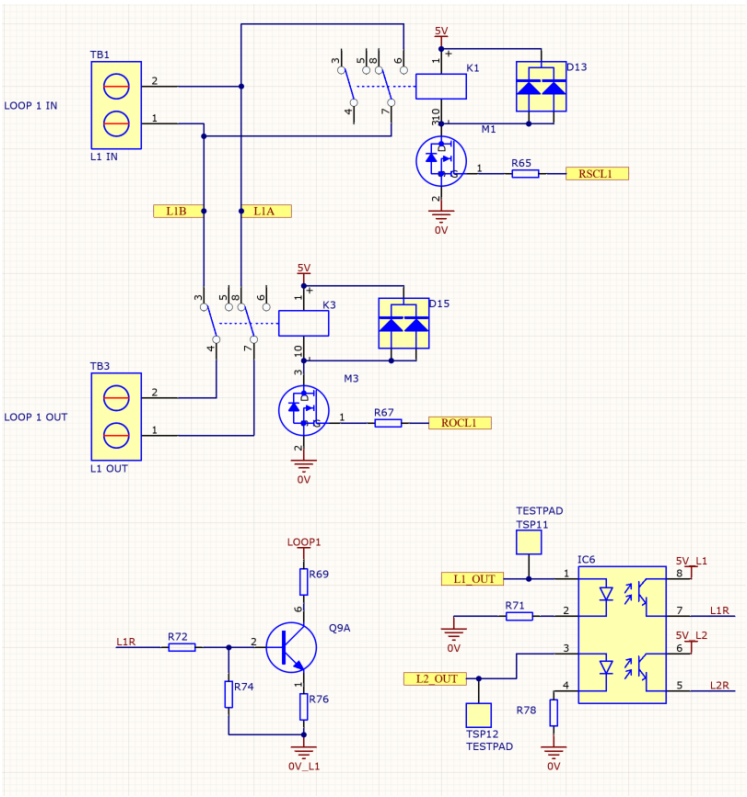


Figura 4-19 - Circuitos referentes a simulação de laço aberto ou de curto circuito no laço. Todos laços possuem o mesmo circuito. (Arquivos GFE)

Na Figura 4-20 tem-se o circuito principal, onde ocorre todo processamento do dispositivo no PIC32. Aqui tem-se a conexão de todos os periféricos do microprocessador PIC32. A direita da imagem pode-se ver um conjunto de Leds vermelhos que sinalizam a conexão de sinal analógico entre o Device emulator e painel, onde há a comunicação através protocolo interno da Global Fire Equipaments.

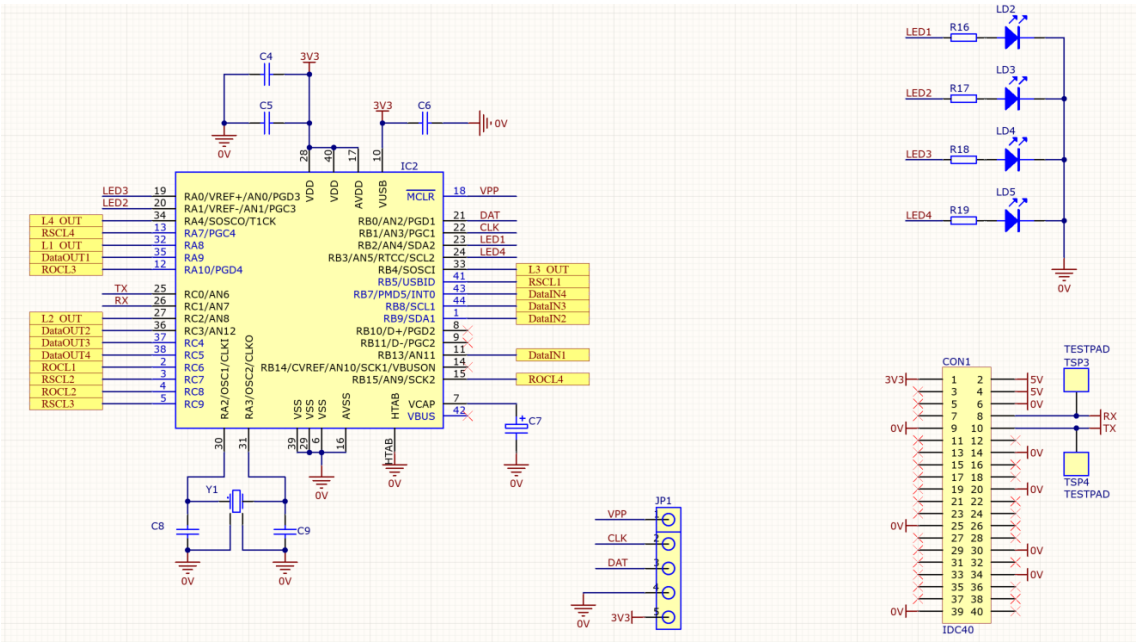


Figura 4-20 - Circuito principal de processamento – PIC 32MX250F128D (Arquivos GFE)

Na Figura 4-21 tem-se a demonstração sinal analógico entre o painel e o dispositivo. Nele há o protocolo de comunicação GFE entre ambos, onde todas informações são realizadas.

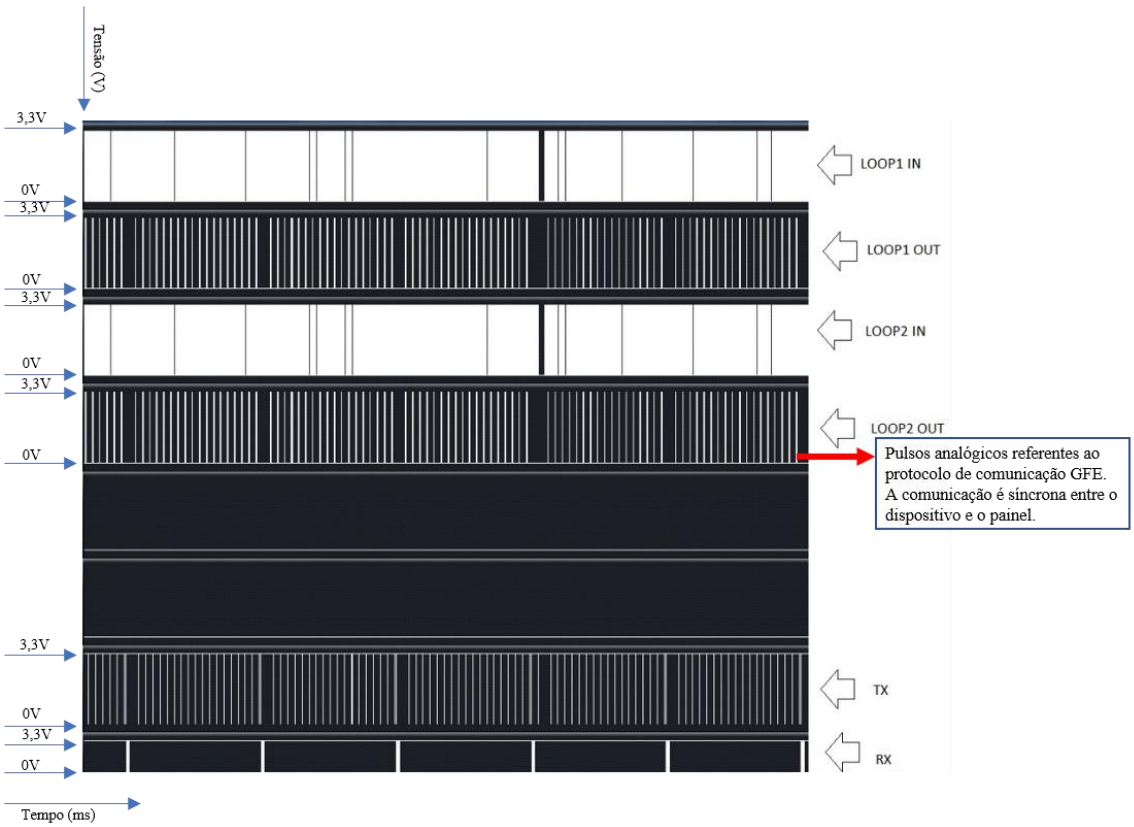


Figura 4-21 - Sinal Analógico de 2 Laços e sinal Tx e Rx.

4.6.2 Características do *Firmware*

A programação do PIC32 é feita em linguagem C, e realizada por meio MPLAB X IDE. Essa parte tem extrema importância, pois será ela que irá definir todos os endereços nos laços onde simulará dispositivos presentes, além de interpretar as informações e valores analógicos enviados pelo painel. Este código faz toda manipulação de valores analógicos e flags necessários para simular a presença de dispositivos no sistema. A estrutura de variáveis principais e suas funções estão na Figura 4-22.

Variáveis principais	Função
bool altera = *false*;	Verifica alterações a gravar
bool combo = *false*;	Controla combo boxes
bool sir = *false*;	Controlar Ativar/silenciar sirenes
bool serie = *false*;	Controlo recepção série
byte countcall = 0;	Contar botão de chamada em fogo
byte cnt = 1;	Contabilizar a resposta da porta série
byte count126 = 0;	Contabilizar quantidades de endereço 126 em evacuação
byte[] x = new byte[4] { 1, 1, 1, 1 };	Controlo da cor do label do pooling
byte loop = 0;	Define *open* e *short circuit*
byte[] readserie = new byte[5];	Recebe dados da porta série: 0-endereço, 1-com1/volt1, 2-com2/volt2, 3-com3/volt3
string titulo;	Controlo do título(ficheiro)
byte[] data = new byte[13] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};	Envia dados pela porta série
byte[] datacall = new byte[5] { 0, 0, 0, 0, 0 };	Chamada de dados
byte[] copy = new byte[126];	Ajuda a copiar laços

Figura 4-22 – Descrição das variáveis da simulação dos auto-testes

No código os ficheiros .c são referentes a programação em linguagem C. Já os ficheiros .h são onde todas as variáveis são declaradas.

O laço pode ter até 125 dispositivos, ou seja, um endereço para cada dispositivo. No protocolo de comunicação o painel interroga cada dispositivo individualmente, porém ao invés de ser apenas 125, ele envia 127 endereços. O endereço 126 é para programar as sirenes e enviar o comando para as mesmas, e ele é enviado a cada 8s. Já o endereço 127

é referente ao comando ASET (Automatic Address Setting) que é uma função da central que programa os dispositivos dando endereço aos mesmos. O 127 é enviado a cada 1s.

Os demais 125 endereços são interrogados em sequência e continuamente, sendo que todo procedimento ocorre simultaneamente em todos os laços.

O código é composto por 6 ficheiros tipo ".c". Que são: *hardware*.c; loop1.c; loop2.c; loop3.c; loop4.c e main.c. O arquivo principal (main.c) é onde todas as funções são chamadas e executadas. O Código *main* encontra-se em Anexo B.

4.6.3 Características do *Software*

O Zeos Loop Emulator é uma aplicação que realiza a emulação de testes por meio de sistema operacional, que neste caso é o Windows. Possui esse nome por causa do dispositivo mais importantes na deteção de incêndios, o detetor Zeos comercializado pela Global Fire Equipments. O Zeos Loop Emulator está em desenvolvimento na linguagem C# no Visual Studio. Nesta aplicação de front-end é realizada toda manipulação de dados, processada no Device Emulator, para interação com usuário. As soluções a nível de teste em sistema são muito satisfatórias.

Através do Zeos Loop Emulator é possível simular a presença de 100% dos dispositivos necessários para preencher os laços, algo que se torna dispendioso para testes rotineiros devido ao grande setup necessário para comportar 125 dispositivos por laço e 500 dispositivos por painel.

Apenas é necessário um dispositivo Device Emulator por painel para simular a presença de 500 dispositivos por painel. No Zeos Loop Emulator pode-se simular situações de funcionamento normal, pré-alarme, alarme, falha, dispositivo desconectado, dois dispositivos em um único endereço, laço aberto e curto circuito no laço. Os dispositivos comercializados pela Global Fire Equipments que estão implementados no Zeos Loop Emulator para simulação em testes são os seguintes: Detectores de incêndio (ZEOS-AD-S; ZEOS-AD-H; ZEOS-AD-SH); Botão de emergência(GFE-MCPE-A); Sirenes(Valkyrie; Vulcan 2; Vulcan 2 Aux Smoke; Vulcan 2 Aux Heat; Vulcan 2 Aux Multi) e Módulos (Input; I/O; LSC; ZMU).

Além das funcionalidades relacionadas com os testes automáticos o Zeos Loop Emulator realiza funções como gerar ficheiros com extensão gle com projectos feitos na aplicação. Assim como também é possível abrir ficheiros, salvar e imprimir.

Na gestão do laço pode-se copiar e colar os dispositivos no laço para transferir para outro laço (o painel interpreta todos os passos de forma instantânea como se houvesse dispositivos presentes no laço. A conexão é síncrona e cíclica).

Existe uma série de funções que podem ser implementadas no decorrer das versões do Emulador. Mas esta versão 1.0.0 é uma ferramenta poderosa no que se trata de testes automáticos na central de proteção contra incêndios da rede Chameleon. O Emulador possui designer e layout básico e intuitivo para o usuário. Na parte inferior da aplicação tem-se a informação geral sobre a quantidade de dispositivos que estão a ser inseridos e simulados no sistema. A última aba após o Loop4 tem-se o Log, nele pode-se acompanhar todos os passos e procedimentos realizados na sessão de teste contendo a informação de Data, hora, loop, endereço e ação executada. Na Figura 4-23 tem-se o Zeos Loop Emulator.

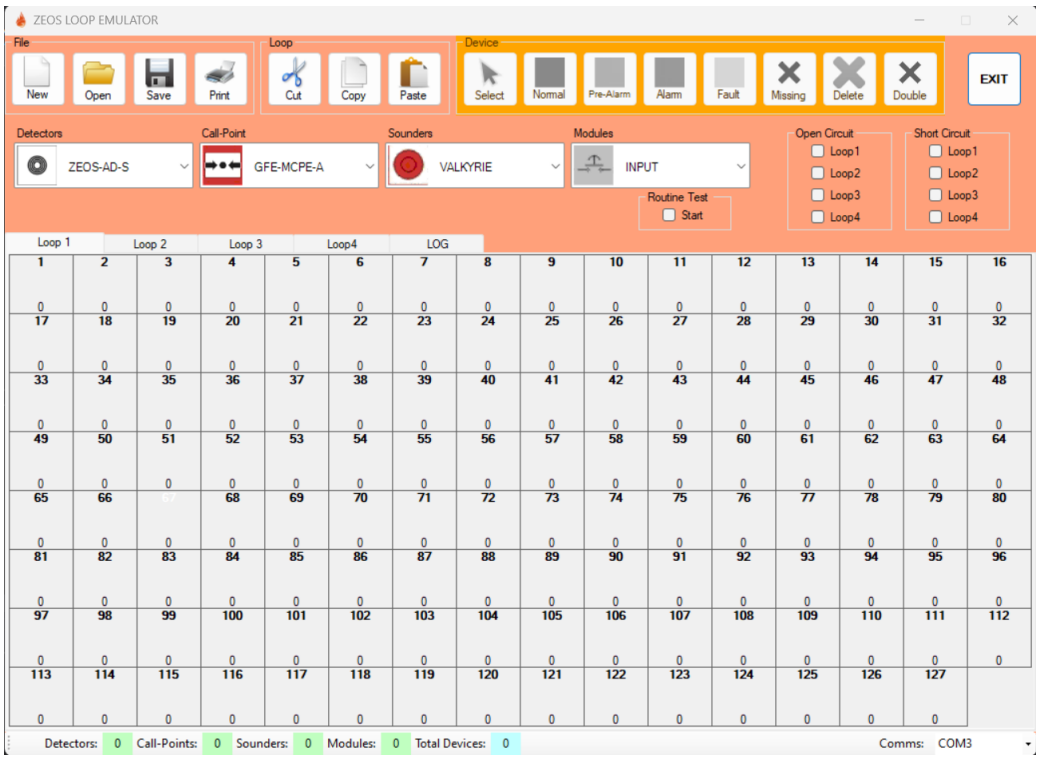


Figura 4-23 – Zeos Loop Emulator.

E em relação aos testes automáticos o Zeos Loop Emulator possui uma checkBox onde é possível definir todo projecto de testes pré-definido. Na saída do log é possível ver os passos executados e o painel irá responder com as ações esperadas, caso tenha alguma falha de execução o Log do Emulador irá apontar. Para iniciar o teste, deve ser enviado ao painel as configurações de teste pré-definidas. Isso é feito de modo simples ao enviar um ficheiro com extensão .gfc do *Software* Connector para o painel. Após o envio, somente será necessário realizar todas conexões do Device Emulator ao painel, abrir a aplicação e marcar a checkBox referente aos testes de rotina. Ressaltando-se que a cada ciclo de teste é necessário realizar o reset ao painel, pois como modo de segurança, o sistema precisa de uma reposição manual toda vez que ocorre situação de alarme ou falha no sistema (com exceção das faltas programadas para ser non-latching).

A implementação em C# dos testes de rotina através do Device Emulator ocorreram por etapas: Implementação de endereços, dispositivos, estados e condições. A seguir pode-se observar a implementação das condições atribuídas ao dispositivo inserido virtualmente ao laço. Neste caso trata-se do laço 1. Para cada laço há um ficheiro de código referente ao processamento do laço (todos são iguais e o processamento ocorre de forma simultânea).

Como é possível observar no código presente no Anexo C, toda simulação e teste é possível devido a comunicação existente entre o painel e o Device Emulator. A trama do protocolo leva informações como o endereço do dispositivo, tipo do dispositivo, valor analógico que o dispositivo está a reportar (isso que caracteriza o estado do dispositivo se é normal, falta ou alarme). Por motivo de segurança a comunicação ocorre de forma redundante, ou seja, o painel interroga o endereço e o dispositivo responde e confirma o endereço, assim como o estado, tipo e valor analógico. Na Figura 4-24 pode-se observar os dispositivos inseridos no laço 2, exatamente como mostra os testes de rotina. Observe que o dispositivo presente no laço 2, endereço 2 está em vermelho, ou seja, estará a simular situação de Fogo. No painel é possível ver na Figura 4-25 o dispositivo presente no Menu 7-1 (Menu que mostra todos dispositivos no sistema), além de ter a sinalização clara e evidente do dispositivo que se encontra em Alarme.

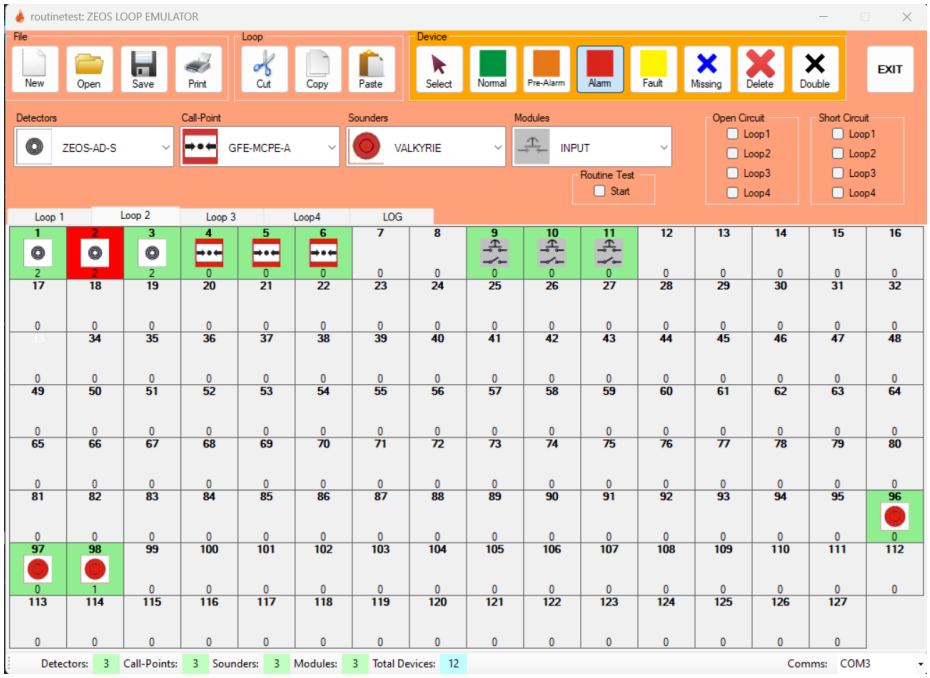


Figura 4-24 – Dispositivos referentes ao teste de rotina.

Figura 4-25 – Painel Gekko a ser testado em situação de Alarme.

Na Figura 4-26 pode-se observar o Zeos Emulator a simular situação de falha e pré-alarme. No Software a sinalização referente a falha é amarela, ou seja, ao redor do campo do endereço em que se encontra o dispositivo em falha, fica em amarelo. Já pré-alarme fica em cor laranja. No painel as sinalizações de falha ficam ativas.

Na Figura 4-27 o que ocorre de maneira diferente a Figura 4-26 é a simulação conjunta do *loop*2 em falha de circuito aberto e do *loop*4 em falha de curto circuito no laço.

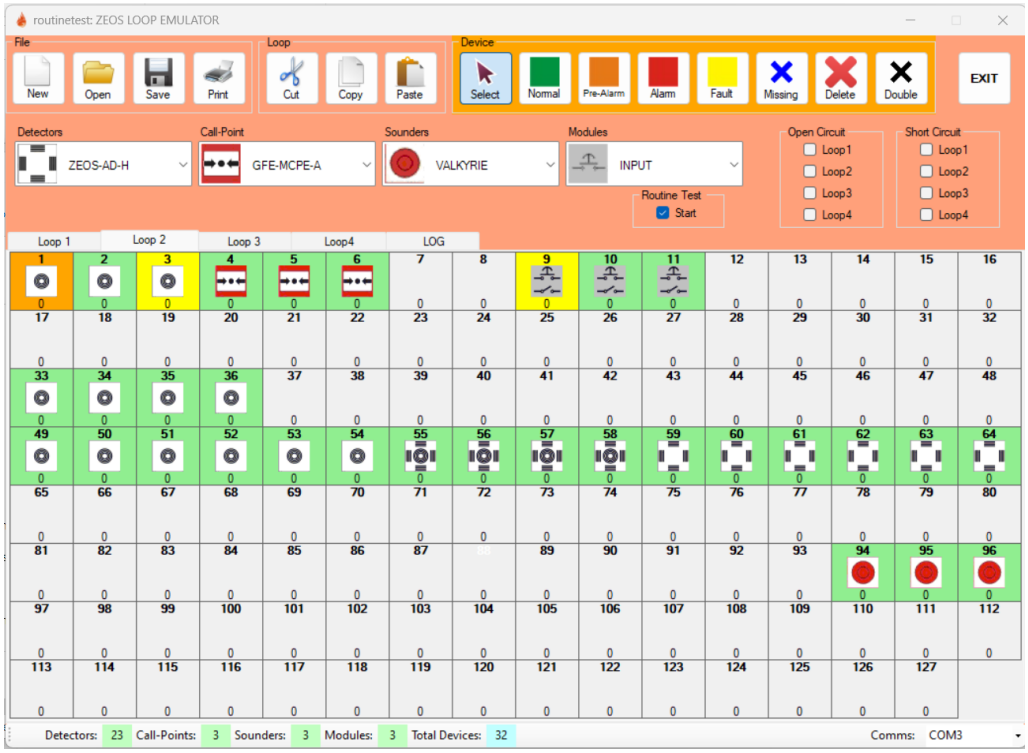


Figura 4-26 – Visualização de dispositivos em Falha (Amarelo) e em Pré-Alarme (Laranja)

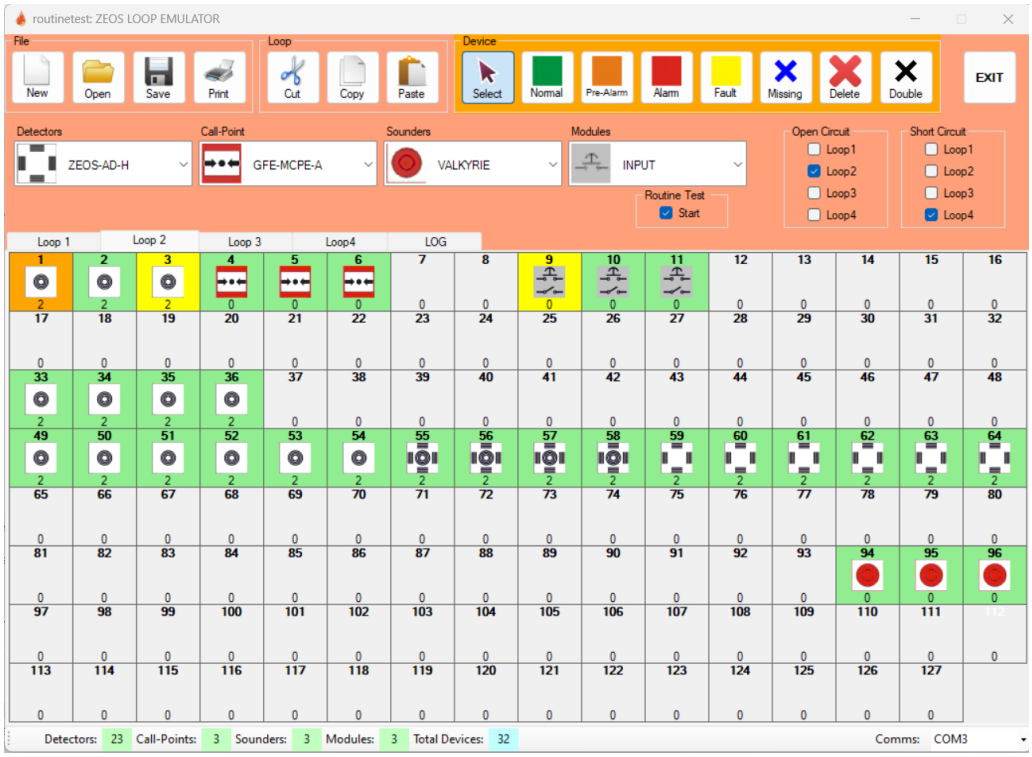


Figura 4-27- Visualização de simulação de circuito aberto no loop2 e Curto circuito no loop4.

4.7 Análise dos resultados

Ao executar a função de engenheira tester é necessário diariamente executar variados testes ao sistema, o que muitas vezes pode se tornar uma tarefa repetitiva e consequentemente mais passível de erros. Após a implementação de alguns testes executados pelo próprio *software* de controlo e também de alguns autotestes a tarefa tornou-se mais confiável. Por se tratar de funcionalidades testadas diretamente pelo usuário final é necessário manter um alto nível de qualidade e confiabilidade do sistema.

Neste relatório tem-se o modelo de um procedimento de testes em um sistema de controlo de proteção contra incêndios além da implementação de testes que devem funcionar de forma automática, ou seja, ao detetar a falha o próprio sistema procurará alternativas, anteriormente implementadas, que executem de forma automática soluções para o problema que causou a falha.

4.7.1 Comunicação entre painéis

Quando se trata de comunicação entre painéis uma das coisas mais importantes a se ressaltar é o fato de todos os painéis reconhecerem todos os outros painéis que se encontrarem na rede. Para dessa forma eles poderem compartilhar informação e estarem em perfeita sincronia no momento que ocorrer falhas nos sistemas. A Figura 4-28 mostra uma rede de painéis formada por 3 painéis (1 Octo e 2 Gekkos) que estão a se comunicar por interface de comunicação RS422 e protocolo GFE.

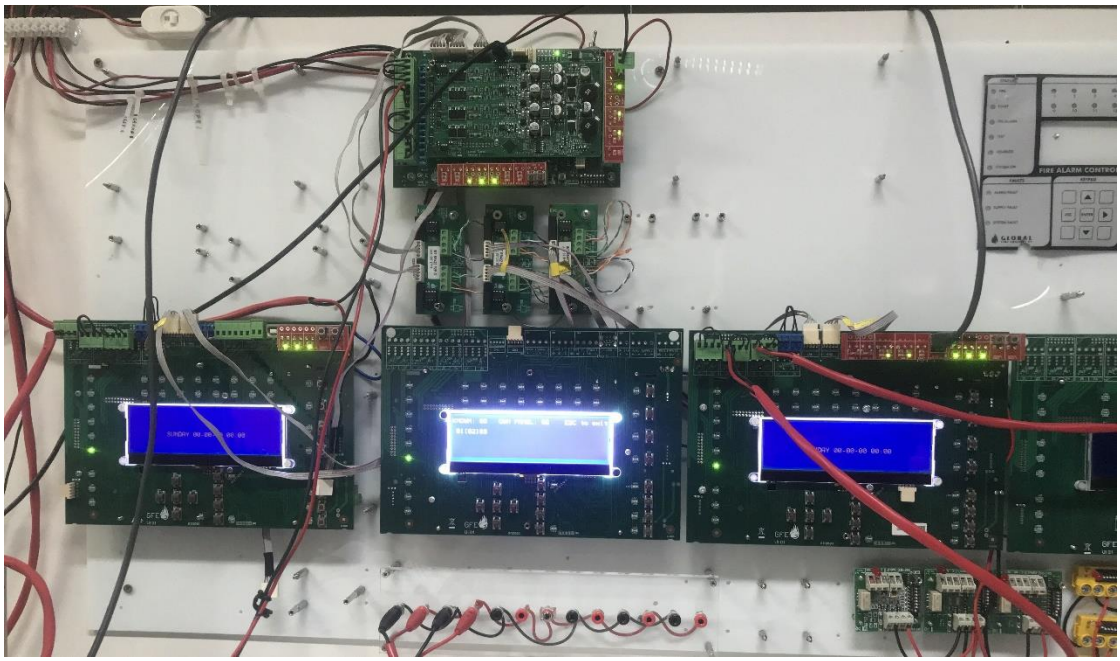


Figura 4-28 – Painéis em Rede: Painel 1 – Gekko; Painel 2 – Octo e Painel 3 – Gekko.

Esses painéis estão a compartilhar status periodicamente. Se houver falha de comunicação os painéis mostrarão uma falha que indica o painel em falha, como mostra a Figura 4-29. O teste executado foi de acordo com o funcionamento normal e esperado do sistema. Os painéis ficaram ligados em rede por uma semana, sem perder alimentação. Durante a semana não houve qualquer falha na comunicação entre os painéis. Então, para mostrar a veracidade do aviso de falha, foi introduzido ao sistema, de forma proposital, uma falha de comunicação (um cabo de conexão do painel 1 foi retirado da interface RS422), logo o painel detetou a falha e disparou a falta e o alarme.

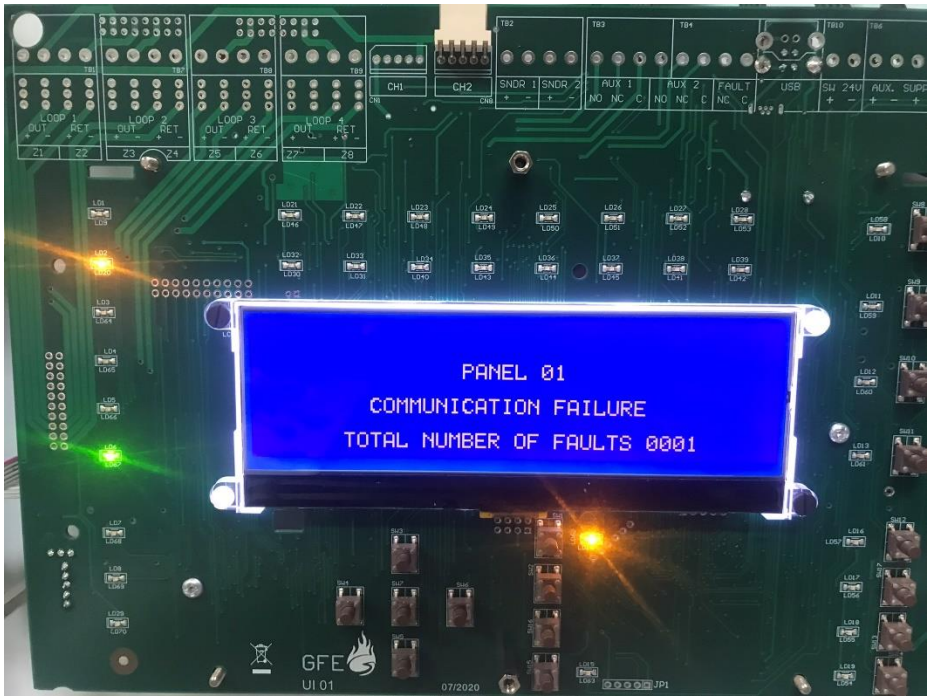


Figura 4-29 – Painel a mostrar falha de comunicação durante teste.

Em funcionamento normal para certificar que os painéis estão a comunicar de forma devida somente é necessário navegar até o Menu 8-5-2 do painel e ver se todos os painéis estão presentes no sistema. O painel que está a ser explorado estará ressaltado com o número entre parênteses retos, como mostra a Figura 4-30.

Figura 4-30 – Identificação e presença dos painéis em rede.

4.7.2 Falha *non-latching*

O Autoteste para falhas non-latching foi implementado e aprovado para comercialização para o cliente. A função atualmente ativa é em relação a falha de falta de alimentação na rede (AC faut). Ou seja, se houver uma falha na tensão de alimentação principal no sistema, o painel irá detetar em tempo real e anunciará a falha. E caso a função *non-latching* estiver ativa no menu 0-1-1 (*Panel Flash Vars Setup*), assim que o sistema detetar que a tensão de alimentação está regular novamente, toda sinalização de falha sairá do visor do painel. Na Figura 4-31 pode-se ver a ativação da função.

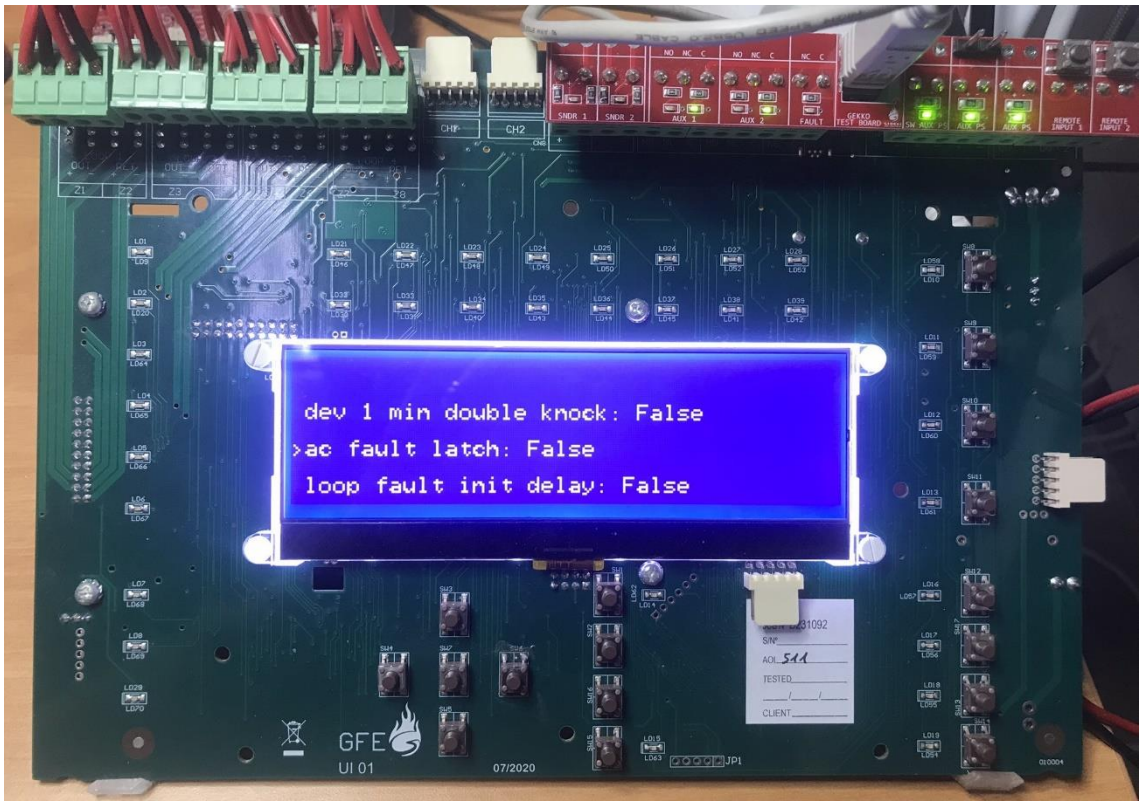


Figura 4-31 - Função “AC fault latch: False” (Menu 0-1-1 Panel Flash Vars Setup)

Tal implementação foi testada e aprovada. O painel demonstrado na Figura 4-31 possui uma alimentação de 28VCC principal e 28VCC na bateria, ambas realizadas por meio de uma fonte de tensão. Para simular a falha, retirou-se a alimentação principal e em cerca de 10 segundos o painel apresentou os alertas de acordo com o esperado. A Figura 4-32 mostra deste resultado do teste.

Figura 4-32 - Painel em falta devido a falta de alimentação principal

Após o estado anterior apresentado, foi reposta a alimentação principal de 28V no painel. Também em cerca de 10s o painel deixou de apresentar todas as falhas. Figura 4-33 mostra o painel em funcionamento normal, após o processamento do autoteste.

Figura 4-33 - Painel após autoteste – sem apresentar falhas.

Caso a falha volte a ocorrer, o painel terá o mesmo comportamento. Sempre a detetar falhas e a reportar de forma rápida e eficiente.

4.7.3 Autotestes de Rotina

Todos os testes de rotina foram executados utilizando-se a ferramenta desenvolvida para testes automatizados, o device Emulator. A seguir tem-se a análise de resultados de todos os passos referentes as cinco simulações dos testes de rotina.

Simulação 1

Na Figura 4-34 pode-se observar que o botão “start” referente aos testes de rotina está ativado, e assim permanecerá até o encerramento de todo processo. Também é possível perceber que todos os dispositivos referentes aos testes se encontram presentes no laço 2. As configurações do painel foram enviadas por meio do *software* connector. E então deu-se início a primeira etapa de testes, ou seja, simulação número 1.

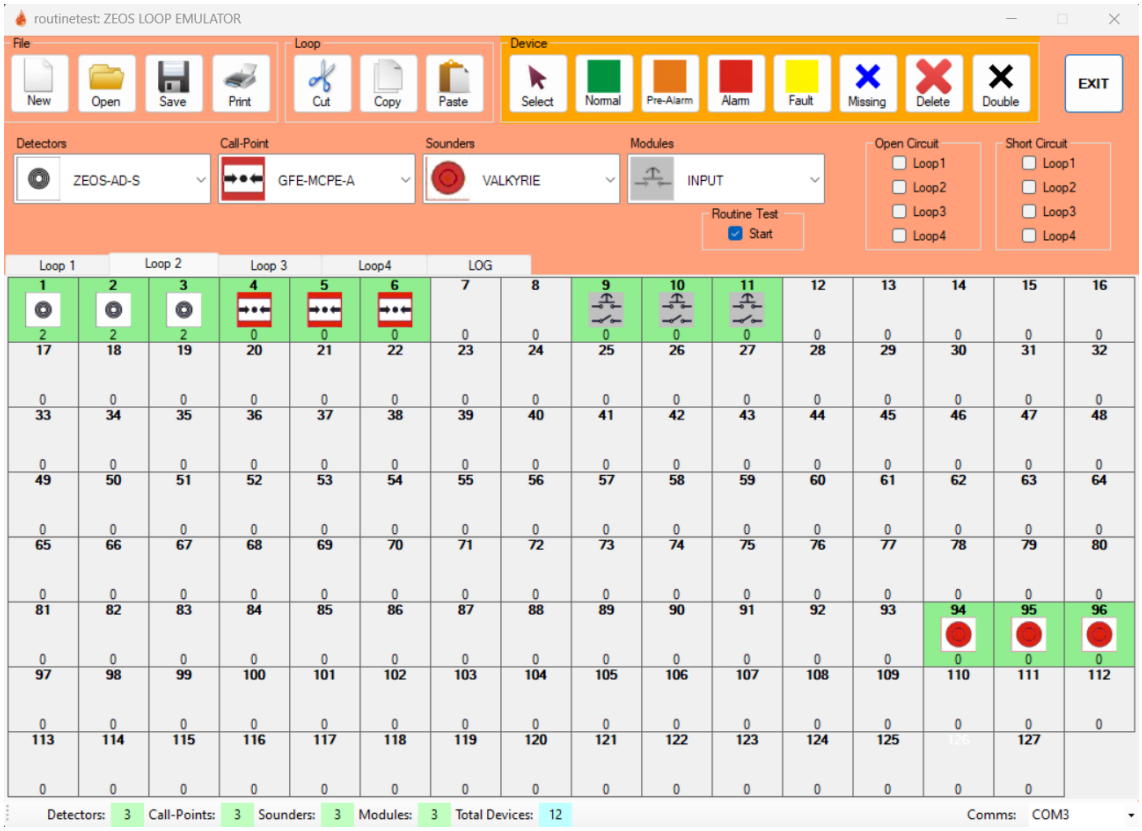


Figura 4-34 – Configuração para testes

Nesta primeira etapa o dispositivo Optical Detector no laço 2 endereço 1 irá entrar em Fogo. E isso testará a respostas do painel e nível de alarme, reconhecimento de dispositivo, análise e endereçamento do fogo. Na Figura 4-35 pode-se ver o resultado do teste no painel. Ele mostra todas as referências exatamente como em uma situação real. Já na Figura 4-36 pode-se ver no Zeos Loop Emulator o dispositivo que "entrou em fogo" ficar vermelho (o que sinaliza o fogo), e todos os dispositivos programados para responder ao alerta também ficam vermelhos sinalizando o sua ativação.

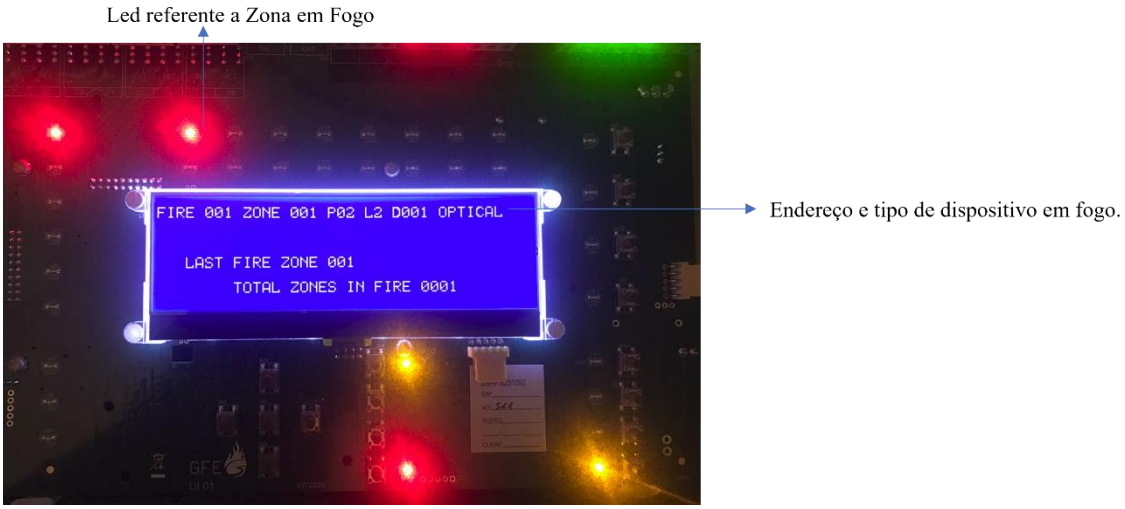


Figura 4-35 - Resposta do painel ao teste – Alertas e endereçamento de dispositivo ativo

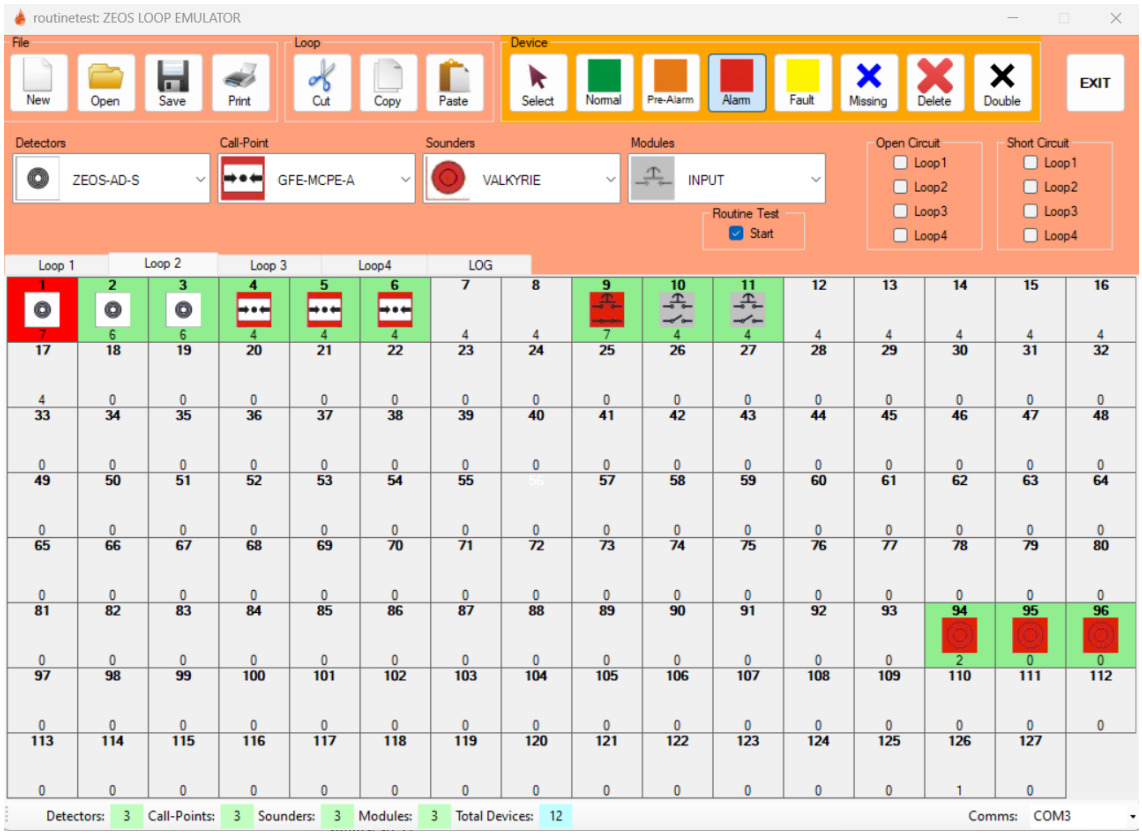


Figura 4-36 - Simulação 1 (teste "Atrasos e Evacuação Imediata") – Etapa 1

Algo muito importante é em relação ao comando processado pelo dispositivo para central entrar em Fogo, falta e pré-alarme. Observa-se que abaixo do dispositivo tem um número. Estes números são referentes a comandos recebidos pelo Zeos Loop Emulator que são processados e analisados tendo-se como base a comunicação entre o Device Emulator e o painel. Dependendo do comando o dispositivo apresentará uma condição diferente. Os comandos foram declarados no FW do painel e enviado por meio do protocolo GFE. Também observa-se que apesar de todas as sirenes (presentes nos endereços 94, 95 e 96) estarem vermelhas (o que sinaliza fogo), apenas a sirene do endereço 94 está a receber o comando 2. Ou seja, todas a sirenes estão a ver o comando 126 (que está a 1) a mandar as sirenes tocar, porém apenas a sirene 94 está programada para tocar nesta situação de fogo.

Na segunda etapa da primeira simulação, dois dispositivos entram em alarme. E essa sequência ativa grupo de I/Os e Grupo de sirenes. Tudo pode ser observado no Zeos Loop Emulator. Na Figura 4-37 a demonstração da segunda etapa.

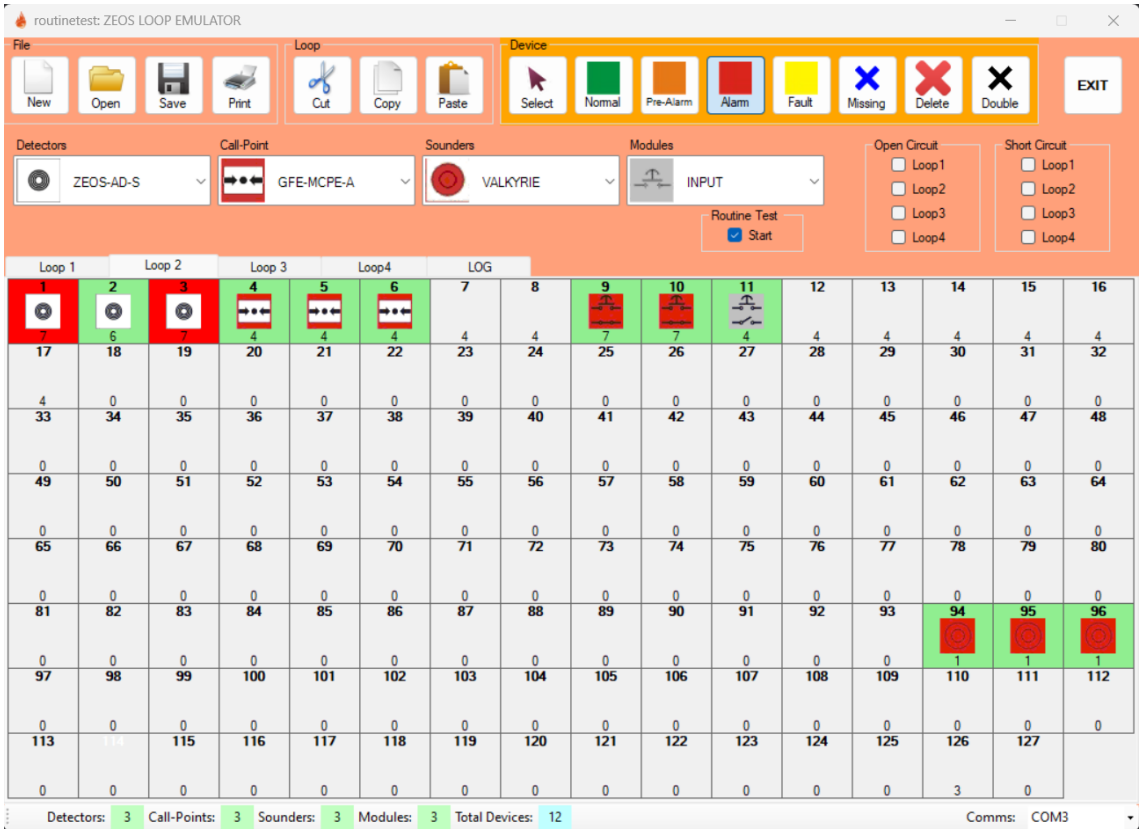


Figura 4-37 - Simulação 1 (teste "Atrasos e Evacuação Imediata") – Etapa 2

Os Detectores Opticos que estão a reporta alarme, recebem do painel o comando 7 que é o comando enviado pelo painel para entrarem em fogo. Os I/Os que se encontram nos endereços 9 e 10 do laço 2 também recebem o comando 7, ou seja, eles estão configurados para ativarem suas saídas após a ativação dos detectores em questão. Realizar tal teste utilizando o Zeos Loop Emulator tornou a tarefa muito mais rápida e prática, comparado com o que costuma ser.

Simulação 2

Neste teste de rotina as funcionalidades relacionas ao desabilitamento de dispositivos são testadas, além das ativações consequentes. Ao desabilitar um detetor, neste caso o Detector Optico presente no laço 2 endereço 2 observa-se se o mesmo apresenta qualquer ação (o que não deve ocorrer) e se a saída do I/O configurada para ativar com desabilitamentos responde corretamente e sem delay. Na Figura 4-38 pode-se ver o teste.

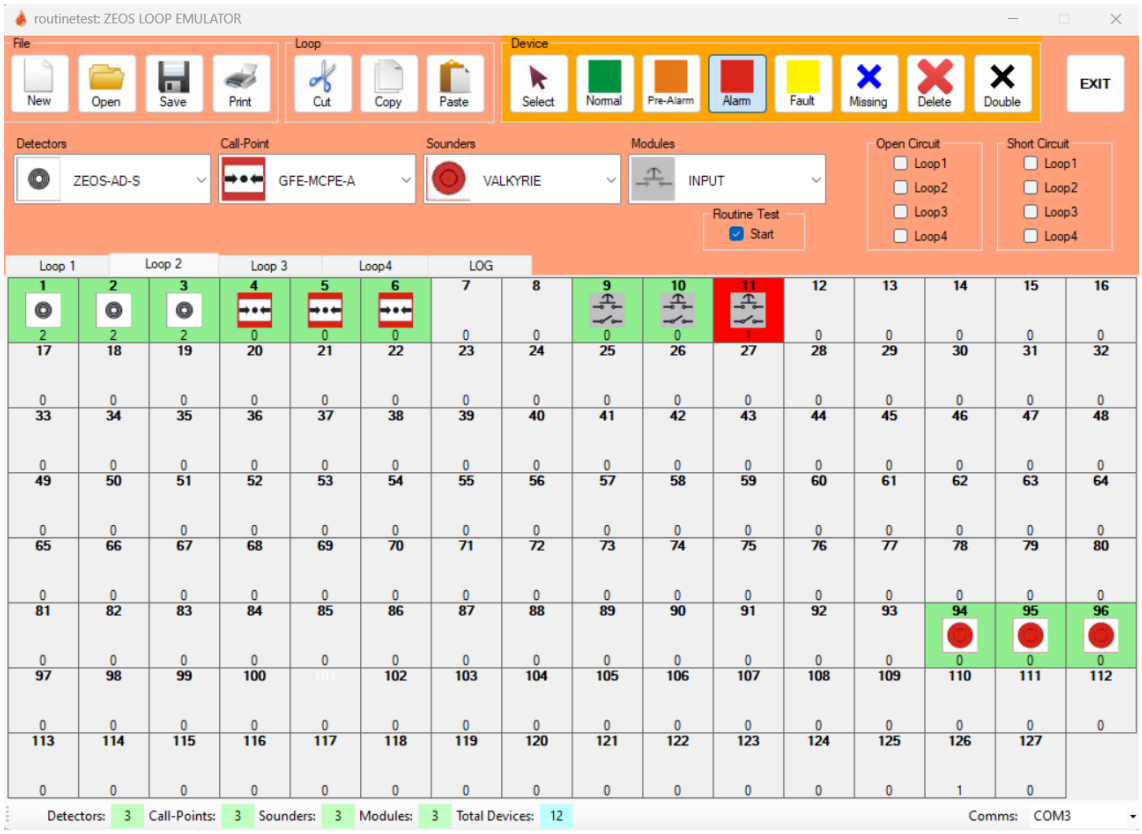


Figura 4-38 - Simulação 2 (teste "Override I/O Delay")

O detetor desabilitado não apresenta qualquer ação e o I/O recebe corretamente o comando 7 enviado pelo painel para atuar, como pode se observar na Figura 4-39.

Figura 4-39 - Painel a sinalizar desabilitamento e a mandar atuar o I/O

Simulação 3

Este teste é temporizado, além de ter algumas atuações mais complexas onde o objetivo é colocar o painel em estado de evacuação após o tempo de 45s configurado. Na

Figura 4-40 é possível conferir o estado dos dispositivos, e perceber que as sirenes estão ativadas em evacuação, isso ocorre após 45segundos em seguida do início da segunda atuação.

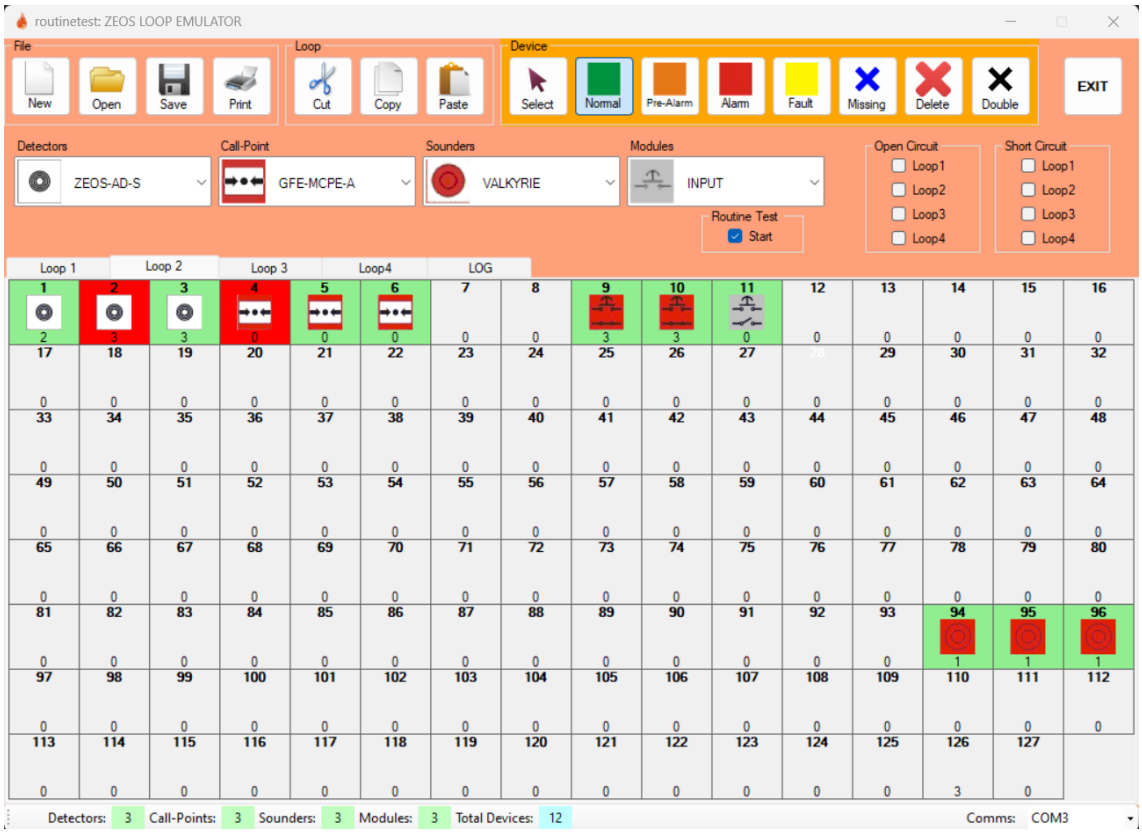


Figura 4-40 - Simulação 3 (teste "Inicia o temporizador de evacuação")

Simulação 4 e Simulação 5

Nesta simulação 4 observa-se na Figura 4-41 algumas funcionalidades a mais como o comando enviado pelo painel para ativar as sirenes que podem tocar de duas formas: 1 – toque contínuo e 2 – toque pulsante. Pode-se perceber que a sirene do endereço 94 está pulsante ao receber 2 e as sirenes 95 e 96 estão em toque contínuo. Já a simulação 5 demonstrada na Figura 4-42 o I/O 11 apenas recebe o valor 3 do painel que mostra que algo não foi enviado corretamente, já que o mesmo deveria receber o comando 7 e atuar. Com esse teste foi possível aceder a uma falha de configuração, resolvê-la e garantir o bom funcionamento do painel de controle e proteção contra incêndios.

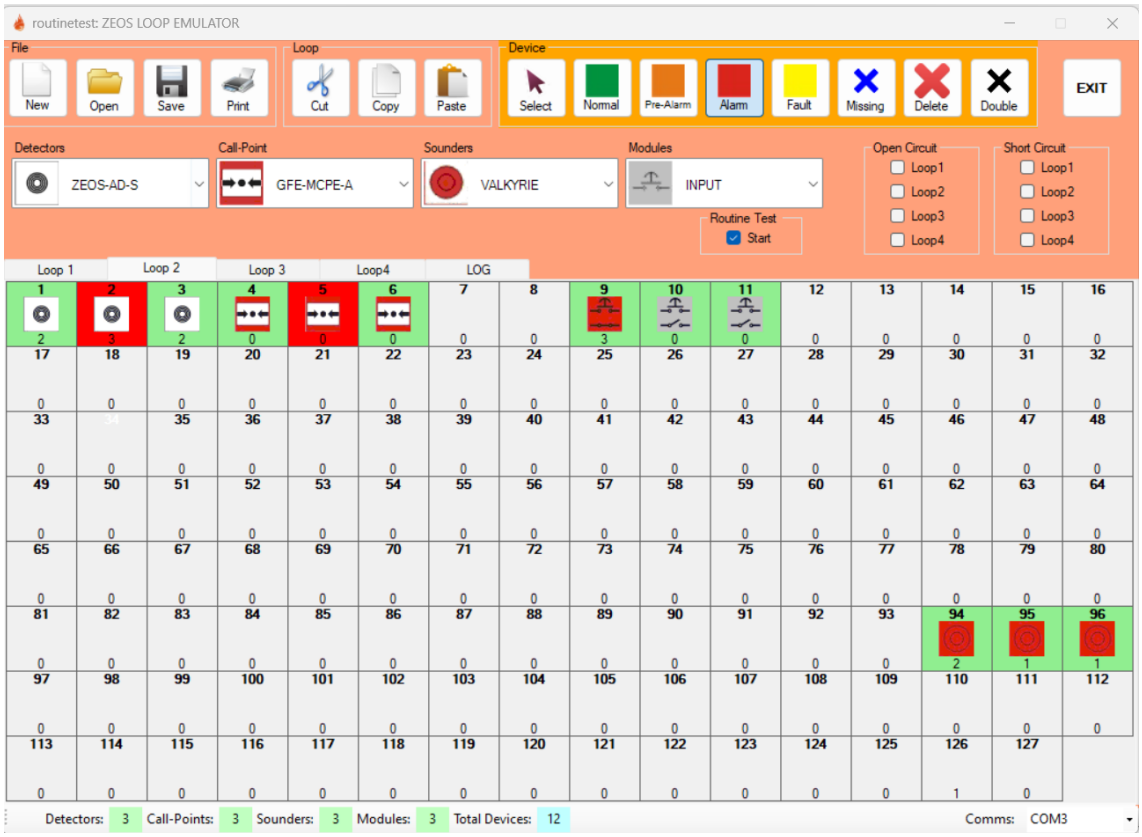


Figura 4-41 - Simulação 4 (teste "Atrasos de Ativação Anulam")

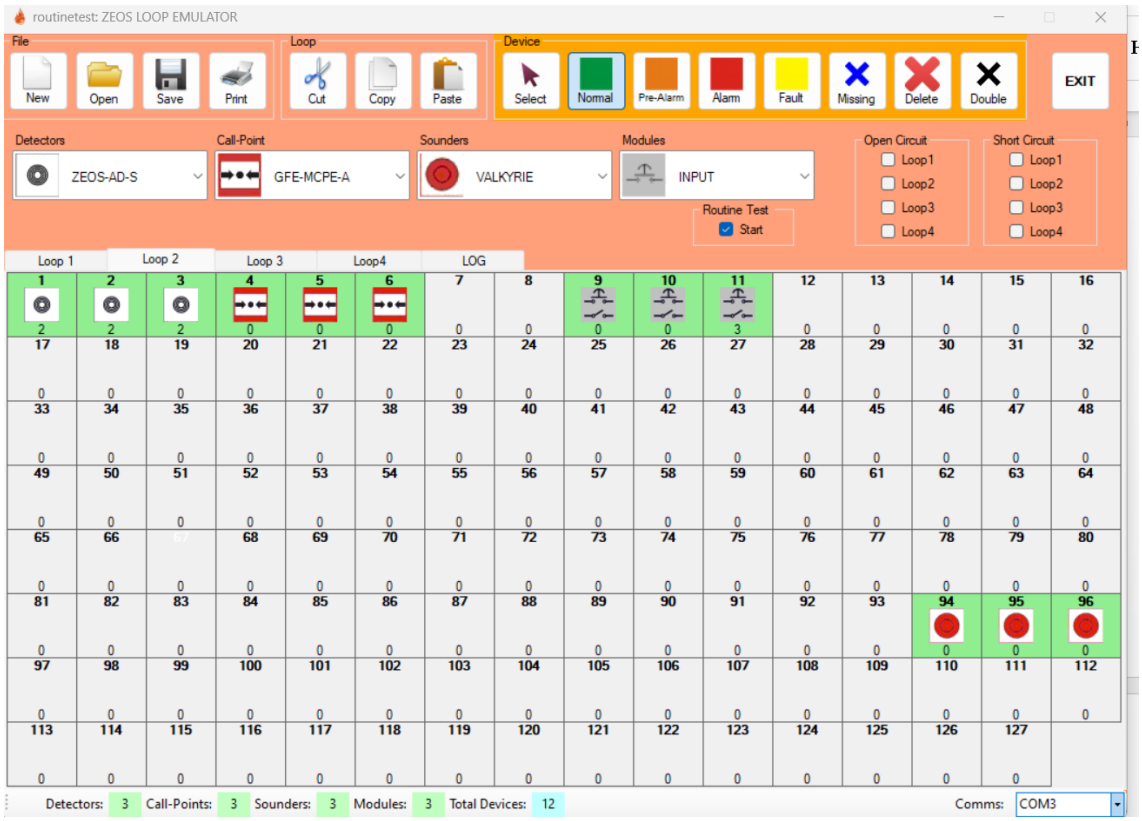


Figura 4-42 - Simulação 5 (teste "Desativação seletiva")

A seguir tem-se o Log de saída do Zeos Loop Emulator. O mesmo é capaz de mostrar todos os passos realizados pelo emulador durante os testes. Como ressaltado anteriormente, o alarme e reset deve ser efetuado manualmente no painel, e por isso não constam na saída do Log do Zeos Loop Emulator.

1/12/2023 11:15:57 PM	2	Loop	Set Smoke Detector in Normal condition
1/12/2023 11:15:57 PM	2	Loop	Set Smoke Detector in Normal condition
1/12/2023 11:15:57 PM	2	Loop	Set Smoke Detector in Normal condition
1/12/2023 11:15:57 PM	2	Loop	Set CallPoint in Normal condition
1/12/2023 11:15:57 PM	2	Loop	Set CallPoint in Normal condition
1/12/2023 11:15:57 PM	2	Loop	Set CallPoint in Normal condition
1/12/2023 11:15:57 PM	2	Loop	Set Input/Output Module in Normal condition
1/12/2023 11:15:57 PM	2	Loop 0	Set Input/Output Module in Normal condition
1/12/2023 11:15:57 PM	2	Loop 1	Set Input/Output Module in Normal condition
1/12/2023 11:15:57 PM	2	Loop 6	Set Valkyrie in Normal condition
1/12/2023 11:15:57 PM	2	Loop 7	Set Valkyrie in Normal condition
1/12/2023 11:15:57 PM	2	Loop 8	Set Valkyrie in Normal condition
1/12/2023 11:16:05 PM	2	Loop 4	Set Valkyrie in Normal condition
1/12/2023 11:16:06 PM	2	Loop 5	Set Valkyrie in Normal condition
1/12/2023 11:16:39 PM	2	Loop	Set Smoke Detector in Alarm condition
1/12/2023 11:16:40 PM	2	Loop	Set CallPoint in Alarm condition
1/12/2023 11:17:14 PM	2	Loop 4	Set Valkyrie in Normal condition
1/12/2023 11:17:15 PM	2	Loop 4	Set Valkyrie in Normal condition
1/12/2023 11:17:15 PM	2	Loop 4	Set Valkyrie in Normal condition

1/12/2023 11:17:16 PM	2	Loop	5	Set Valkyrie in Normal condition
1/12/2023 11:17:16 PM	2	Loop	6	Set Valkyrie in Normal condition
1/12/2023 11:17:17 PM	2	Loop	4	Set Valkyrie in Normal condition
1/12/2023 11:17:18 PM	2	Loop	4	Set Valkyrie in Normal condition
1/12/2023 11:17:19 PM	2	Loop	5	Set Valkyrie in Normal condition
1/12/2023 11:17:19 PM	2	Loop	6	Set Valkyrie in Normal condition
1/12/2023 11:17:22 PM	2	Loop	4	Set Valkyrie in Normal condition
1/12/2023 11:17:22 PM	2	Loop	5	Set Valkyrie in Normal condition
1/12/2023 11:17:23 PM	2	Loop	6	Set Valkyrie in Normal condition
1/12/2023 11:17:28 PM	2	Loop		Set Smoke Detector in Alarm condition
1/12/2023 11:17:41 PM	2	Loop		Set Smoke Detector in Alarm condition
1/12/2023 11:17:45 PM	2	Loop		Set Smoke Detector in Normal condition
1/12/2023 11:18:54 PM	2	Loop		Set CallPoint in Normal condition
1/12/2023 11:18:54 PM	2	Loop		Set Smoke Detector in Normal condition
1/12/2023 11:20:03 PM	2	Loop		Set Smoke Detector in Alarm condition
1/12/2023 11:20:06 PM	2	Loop		Set CallPoint in Alarm condition
1/12/2023 11:22:16 PM	2	Loop		Set Smoke Detector in Normal condition
1/12/2023 11:22:19 PM	2	Loop		Set CallPoint in Normal condition

Capítulo **5**

5 Considerações Finais

5.1 Conclusão

O desenvolvimento do projeto possibilitou uma análise sobre a importância dos procedimentos de teste em sistemas de segurança e alarme contra incêndios. Pode-se ter uma noção básica sobre os paradigmas de teste e suas vantagens, bem como desvantagens. Adicionalmente, foi possível expor técnicas de teste aplicáveis para o cenário proposto.

Os principais objetivos do relatório foi mostrar os métodos e teste que são usados atualmente nas centrais de controlo, qual a importância da aplicação destes testes e sugerir a implementação de novos testes automáticos que tem como principal função facilitar todo processo realizado na metodologia do teste, diminuir a incidência de falhas durante o teste e possibilitar que o usuário final possa aceder ao painel e facilmente e ativar o modo de teste através de Menu 0-1-1. Ao buscar maneiras de tornar os testes mais rápidos, eficientes e compactos (devido ao grande setup para testes por causa da grande quantidade de dispositivos e cabos) foi adaptada uma ferramenta anteriormente desenvolvida para testes, para auxiliar nos testes automáticos e não automáticos. O Device Emulator teve um excelente desempenho nos testes de rotina. Uma das maiores vantagens observada

com o uso dessa ferramenta através do Zeos Loop Emulator foi a praticidade de se ter todos os equipamentos “em um só”. Apenas com um dispositivo é possível simular 125 equipamentos por laço e 500 por painel ao mesmo tempo, algo que se torna completamente inviável em uma bancada de testes no setor de desenvolvimento. Além disso, ver o protocolo interrogar os endereços em tempo real, manipular os estados dos dispositivos de forma instantânea e visualizar as respostas enviadas pelo painel aos dispositivos são mais algumas das grandes vantagens trazidas pelo Device Emulator.

A cobertura de falhas pode ser aumentada com a introdução de rotinas (algoritmos) de teste, no entanto pode-se concluir que os resultados alcançados são satisfatórios. A rotina de testes do sector de desenvolvimento melhorou muito com essa ferramenta e trouxe muitas ideias para desenvolvimentos futuro. Além disso, os testes realizados no sector de pesquisa e desenvolvimento referentes ao sistema Chameleon passaram a possuir procedimentos a serem seguidos, o que tem como resultado mais fiabilidade no sistema de protecção contra incêndios e abriu espaço para novos métodos e rotina no trabalho do desenvolvedor na busca da excelência no produto final.

5.2 Trabalhos Futuros

- Implementação de Autoteste com tecnologia *Digital Twins*;
- Desenvolvimento de um aplicativo que controla o sistema de teste;
- Implementar o Device Emulator para trabalhar em uma plataforma online e remota. Desta forma este dispositivo pode se transformar em um produto que pode ser comercializado aos clientes e ser uma ferramenta de testes para instaladores.
- Implementar o Zeos Loop Emulator para criar testes automáticos por meio de projectos. Desta forma não será necessário a implementação a nível de código toda vez que fosse necessário executar um novo teste.

Referências

[1] WONG, W. Eric, et al. Coverage Testing Embedded Software on Symbian/OMAP. In: SEKE. 2006. p. 473-478.

[2] Grieves, Michael. "Digital twin: manufacturing excellence through virtual factory replication." White paper 1.2014 (2014): 1-7.

[3] Tian, P., Wang, J., Leng, H., & Qiang, K. (2009, August). Construction of distributed embedded software testing environment. In 2009 International Conference on Intelligent Human-Machine Systems and Cybernetics (Vol. 1, pp. 470-473). IEEE.

[4] Batty, Michael. "Digital twins." Environment and Planning B: Urban Analytics and City Science 45.5 (2018): 817-820.

[5] Broekman, Bart, and Edwin Notenboom. Testing embedded software. Pearson Education, 2003.

[6] Qi Zhichang, Tan Qingping, Ning Hong. Engenharia de Software. Pequim: Higher Education Press, 20012

[7] Weyuker E J. Evaluation Techniques for Improving the Quality of Very Large Software Systems in a Cost2Effective Way. The Journal of Systems and Software , 1999, 47: 972103

[8] King S , Hammond J , Chapman R , et al . Is Proof More Cost2Effective Than Testing ? IEEE Transactions on Software Engineering , 2000 , 26 (8): 6752686

[9] Arthur E. Cote, “Fire Protection Handbook,” Nineteenth Edition, Volumes I & II, Copyright© 2003, National Fire Protection Association, Inc., Quincy, Massachusetts, USA.

[10] M. J. Howes and D. V. Morgan, editors, Reliability and Degradation – Semiconductor

[11] Meirelles, Paulo Roberto Miranda. "Teste integrado de software e *hardware*: reusando casos de teste de software em teste de microprocessadores." (2008).

[12] Bushnell, Michael, and Vishwani Agrawal. Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits. Vol. 17. Springer Science & Business Media, 2004.

[13] S. T. Chakradhar and V. D. Agrawal, “VLSI Design,” in A. Kent and J. G. Williams, editors, *Encyclopedia of Microcomputers, Volume 20*, pp. 97–111, New York: Marcel Dekker, 1997.

[14] W. R. Simpson and J. W. Sheppard, *System Test and Diagnosis.* Boston: Kluwer Academic Publishers, 1994.

[15] Goodenough J B , Gerhart S L. Toward a Theory of Test Data Selection. IEEE Transactions on Software Engineering , 1975 , SE23(6): 1562173

[16] Zhu Hong, Jin Lingzi. Teste e garantia de qualidade de software. Pequim: Science Press, 1997.

[17] Tracey N J. A Search2Based Automated Test2Data Generation Framework for Safety2Critical Software: [PhD thesis].Department of Computer Science , University of York , 2000

[18] Li Liuying Pesquisa e Implementação de Tecnologia de Teste UML: [Dissertação] Changsha: Escola de Pós-Graduação em Tecnologia da Universidade Nacional de Defesa, 2000

[19] Korel B , Al2Yami A M. Assertion2Oriented Automated Test Data Generation. In : Proceedings of the 18th InternationalConference on Software Engineering ,Berlin :1996. 71280

[20] Weyuker E J. The Applicability of Program Schema Results to Programs. International Journal of ComputerInformation Sciences , 1979 , 8(5): 3872403

[21] Ramanmoorthy C V , Ho S2B F , Chen W T. On the Automated Generation of Program Test Data. IEEE Transactionson Software Engineering , 1976 , SE22(4): 2932300

[22] Michael C C , McGraw G, Schatz M A. Generating Software Test Data by Evolution. IEEE Transactions on SoftwareEngineering , 2001 , 27 (12): 1 08521 110

[23] Harrold M J. Testing: A Roadmap. In: Finkelstein A eds. The Future of Software Engineering. New York :ACMPress , 2000

[24] Chen T Y, Lau M F. A New Heuristic for Test Suite Reduction. Information and Software Technology , 1998, 40:3472354

[25] Onoma A K, Tsai W2T , Poonawala M H , et al . Regression Testing in an Industrial Environment. CACM , 1998,41(5): 81286

[26] Martins E , Toyota C M , Yanagawa R L . Constructing Self2Testable Software Components. In: Proceedings of theInternational Conference on Dependable Systems and Networks, Goteborg:2001. 1512160

[27] Meyer B. Object2Oriented Software Construction. New Jersey :Prentice Hall , 1997

[28] *Devices and Circuits.* Chichester, UK: Wiley-Interscience, 1981.

[29] Correia, João Pedro Pais de Figueiredo. "Sistema de detecção de incêndios." (2008)

[30] Z. Liu, J. Makar and A. K. Kim, “Development of Fire Detection Systems in the Intelligent Building,“ 12th International Conference on Automatic Fire Detection, Gaithersburg, MD., 2001, pp. 561-573, www.nrc.ca/irc/ircpubs, Institute for Research in Construction, National Research Council of Canada, Ottawa, Canada.

[31] Resende, Rogério Manuel Teixeira. "Detecção e alarme de incêndio: sistemas actuais." (2009).

[32] Manual Chameleon Connector – Global Fire Equipments

[33] Autoridade Nacional de Protecção Civil (ANPC). Nota Técnica nº 12 – Sistemas Automáticos de Detecção de Incêndio. 2007.

[34] Clarke E M , Grumberg O , Peled D . Model Checking. Cambridge , Massachusetts , London , England : The MIT Press , 1999

Anexos

Anexo A

```
void monitoring_battery(unsigned char ch)
{
   switch(ch){
      case NULL_KEY:
      {
         // First Run

         init_num_str(&func_entry, 0, 0, 1, 1);
         init_num_str(&func_zone, 0, 0, 3, 1);
         render_clear_line(lcd_programming_screen, LINE3);
         battery_manager.monitor.show_bat_level = 1;

         break;
      }
      case ESCAPE:
      {
         if(func_entry.num)
         {
            update_num_str(&func_entry,LEFTARROW);
         }

         else
         {
            battery_manager.monitor.show_bat_level = 0;
            exit_function();
            return;
```

```
    }

    break;
}
case ENTER:
{
  if (func_entry.num == 0)
    {
       //set charger status (on/off)
       if (func_zone.num == 1){
          isolate_battery();
       }
       else if (func_zone.num == 2){
          isolate_charge_battery();
       }
       else if (func_zone.num == 3){
          battery_charge_off();
       }
       else if (func_zone.num == 0){
          normal_battery_state();
       }

    }
    else  if (func_entry.num == 1)
    {
       //enable load test procedure
       battery_load_state = BATTERY_LOADTEST_START;
       func_zone.num = 1;

    }

    break;
}
```

```
        case LEFTARROW:
        case RIGHTARROW:
        {
          update_num_str(&func_entry,ch);
          break;
        }

        case DOWNARROW:
        case UPARROW:
        {
          if (func_entry.num == 0){
            update_num_str(&func_zone,ch);
          }
          break;
        }

    }

void show_battery_voltage (void){
    float v_bat, v_total;
    uint16_t v_total_decimal;

    if (flag_time_to_show_battery_voltage){
        flag_time_to_show_battery_voltage = false;

        v_bat = bat_volt_monitor.current_value * 3.33 / 1024;
        v_total = ( v_bat / (2.2/(2.2 + 9.1))) + 15
        /*
        - (primary_fuse.count ? 0 : 0.5) /*diode*/ /*;
        v_total_decimal = (int)((uint16_t)(v_total*100)%100)

    }
}
```

Anexo B

Zeos Emulator

MAIN

```
“
#include <stdio.h>
#include <stdlib.h>
#include <plib.h>
#include <xc.h>
#include <p32xxxx.h>
#include "hardware.h"
#include "loop1.h"
#include "loop2.h"
#include "loop3.h"
#include "loop4.h"

//com cristal 8MHz
// DEVCFG3
// USERID = No Setting
#pragma config PMDL1WAY = OFF       // Peripheral Module Disable Configuration (Allow multiple reconfigurations)
#pragma config IOL1WAY = OFF        // Peripheral Pin Select Configuration (Allow multiple reconfigurations)
#pragma config FUSBIDIO = OFF       // USB USID Selection (Controlled by Port Function)
#pragma config FVBUSONIO = OFF      // USB VBUS ON Selection (Controlled by Port Function)

// DEVCFG2
#pragma config FPLLIDIV = DIV_2     // PLL Input Divider (2x Divider)
#pragma config FPLLMUL = MUL_20     // PLL Multiplier (20x Multiplier)
#pragma config UPLLIDIV = DIV_2     // USB PLL Input Divider (2x Divider)
#pragma config UPLLEN = ON          // USB PLL Enable (Disabled and Bypassed)
#pragma config FPLLODIV = DIV_2     // System PLL Output Clock Divider (PLL Divide by 2)

// DEVCFG1
#pragma config FNOSC = PRIPLL       // Oscillator Selection Bits (Primary Osc w/PLL (XT+,HS+,EC+PLL))
#pragma config FSOSCEN = OFF        // Secondary Oscillator Enable (Disabled)
#pragma config IESO = OFF           // Internal/External Switch Over (Disabled)
#pragma config POSCMOD = HS         // Primary Oscillator Configuration (HS osc mode)
#pragma config OSCIOFNC = OFF       // CLKO Output Signal Active on the OSCO Pin (Disabled)
```

```
#pragma config FPBDIV = DIV_1          // Peripheral Clock Divisor (Pb_Clk is Sys_Clk/1)
#pragma config FCKSM = CSECME          // Clock Switching and Monitor Selection (Clock Switch Enable, FSCM Enabled)
#pragma config WDTPS = PS4096          // Watchdog Timer Postscaler (1:4096)
#pragma config WINDIS = OFF            // Watchdog Timer Window Enable (Watchdog Timer is in Non-Window Mode)
#pragma config FWDTEN = ON             // Watchdog Timer Enable (WDT Disabled (SWDTEN Bit Controls))
#pragma config FWDTWINSZ = WISZ_25     // Watchdog Timer Window Size (Window Size is 25%)

// DEVCFG0
#pragma config JTAGEN = OFF            // JTAG Enable (JTAG Disabled)
#pragma config ICESEL = ICS_PGx1       // ICE/ICD Comm Channel Select (Communicate on PGEC1/PGED1)
#pragma config PWP = OFF               // Program Flash Write Protect (Disable)
#pragma config BWP = OFF               // Boot Flash Write Protect bit (Protection Disabled)
#pragma config CP = OFF                // Code Protect (Protection Disabled)

void main(void)
{
  unsigned char i;
  unsigned char j;

  picini();                                   //config PIC

  if(!RCONbits.WDTO)                          //if reset by watchdog
    RCONbits.WDTO = 1;
  else if(RCONbits.WDTO)                      //if not
  {
    for (i = 0;i < 126;i++)                   //clear all devices data
    {
      for (j = 0;j < 2;j++)
      {
        data1[i][j] = 0;
        data2[i][j] = 0;
        data3[i][j] = 0;
        data4[i][j] = 0;
      }
    }
  }

  led1 = led2 = led3 = led4 = 1;
  DelayMs(50);
  led1 = led2 = led3 = led4 = 0;

  //Config External 2 (loop1)
```

```
    ConfigINT2(EXT_INT_PRI_5 | FALLING_EDGE_INT | EXT_INT_ENABLE);
//enable external interrupt to loop1
    mINT2ClearIntFlag();

    //Config External 1 (loop2)
    ConfigINT1(EXT_INT_PRI_5 | FALLING_EDGE_INT | EXT_INT_ENABLE);
//enable external interrupt to loop2
    mINT1ClearIntFlag();

    //Config External 3 (loop3)
    ConfigINT3(EXT_INT_PRI_5 | FALLING_EDGE_INT | EXT_INT_ENABLE);
//enable external interrupt to loop3
    mINT3ClearIntFlag();

    //Config External 4 (loop4)
    ConfigINT4(EXT_INT_PRI_5 | FALLING_EDGE_INT | EXT_INT_ENABLE);
//enable external interrupt to loop4
    mINT4ClearIntFlag();

    INTEnable(INT_SOURCE_UART_RX( UART1 ), INT_ENABLED);
    INTSetVectorPriority(INT_VECTOR_UART( UART1 ),
INT_PRIORITY_LEVEL_2);
    INTSetVectorSubPriority(INT_VECTOR_UART( UART1 ),
INT_SUB_PRIORITY_LEVEL_0);

    INTConfigureSystem(INT_SYSTEM_CONFIG_MULT_VECTOR);
    //INTEnableSystemMultiVectoredInt();                    //start all enabled
interrupts

    INTEnableInterrupts();

    ClearWDT();

    while(1)
    {
      while(!flag1 && !flag2 && !flag3 && !flag4);
      if(flag1)
      {
        send_data(loopadr1);
      }
      else if(flag2)
      {
        send_data(loopadr2);
      }
      else if(flag3)
      {
        send_data(loopadr3);
      }
      else if(flag4)
      {
```

```
        send_data(loopadr4);
      }
      flag1 = flag2 = flag3 = flag4 = 0;
      ClearWDT();
    }
  }
  ”
```

Anexo C

```
  “… //Select ou Normal condition
                    if ((selectButton.Checked == true || normalButton.Checked
== true) && (loop1[i].type != 0))
                    {
                        L1picBox[i].BackColor    =    L1lbl[i].BackColor    =
L1labcom[i].BackColor = Color.LightGreen;
                        loop1[i].cond = 1;
                        if  (loop1[i].type  ==  1  ||  loop1[i].type  ==  2  ||
loop1[i].type == 3 ||
                            loop1[i].type  ==  7  ||  loop1[i].type  ==  8  ||
loop1[i].type == 9)//se detector
                            loop1[i].devanalogue = 25;
                        else
                            loop1[i].devanalogue = 16;
                    }
                    //PreAlarm condition
                    else if (prealarmButton.Checked == true && loop1[i].type !=
0)
                    {
                        if  (loop1[i].type  ==  1  ||  loop1[i].type  ==  2  ||
loop1[i].type == 3 ||
                            loop1[i].type  ==  7  ||  loop1[i].type  ==  8  ||
loop1[i].type == 9)//se detector
                        {
                            L1picBox[i].BackColor   =   L1lbl[i].BackColor   =
L1labcom[i].BackColor = Color.Orange;
                            loop1[i].cond = 2; //pré-alarme condição
                            loop1[i].devanalogue = 45;
                        }
                    }
                    //Fault condition
                    else if (faultButton.Checked == true && loop1[i].type != 0)
                    {
                        if  (loop1[i].type  !=  4  &&  loop1[i].type  !=  5  &&
loop1[i].type != 6 &&
                            loop1[i].type  !=  7  &&  loop1[i].type  !=  8  &&
loop1[i].type != 9)
                        {
                            L1picBox[i].BackColor   =   L1lbl[i].BackColor   =
L1labcom[i].BackColor = Color.Yellow;
                            loop1[i].cond = 4; //condição de falta
                            loop1[i].devanalogue = 4;
                        }
                    }
                    //Alarm condition
                    else if (alarmButton.Checked == true && loop1[i].type != 0)
                    {
```

```
                        if (loop1[i].type != 5 && loop1[i].type != 6 &&
loop1[i].type != 12)
                        {
                            L1picBox[i].BackColor = L1lbl[i].BackColor =
L1labcom[i].BackColor = Color.Red;
                            loop1[i].cond = 3; //condição de alarme
                            loop1[i].devanalogue = 64;
                        }
                        //if (loop1[i].type == 4)
                        //{
                        //    myQue.Enqueue(i); //endereço
                        //    myQue.Enqueue(1); //laço
                        //    countcall = (byte)myQue.Count;
                        //}
                    }
                    //Missing condition
                    else if (missingButton.Checked == true && loop1[i].type !=
0)
                    {
                        loop1[i].cross_blue = !loop1[i].cross_blue;
                        if (loop1[i].cross_blue == true)//colocar cruz azul
                        {
                            loop1[i].cond = 5;
                            loop1[i].devanalogue_copy = loop1[i].devanalogue;
                            loop1[i].devtype_copy = loop1[i].devtype;
                            loop1[i].devanalogue = loop1[i].devtype = 0;
                        }
                        else if (loop1[i].cross_blue == false)//retirar cruz
azul e colocar fundo anterior
                        {
                            loop1[i].cond = 6;
                            loop1[i].devanalogue = loop1[i].devanalogue_copy;
                            loop1[i].devtype = loop1[i].devtype_copy;
                            loop1[i].devanalogue_copy = loop1[i].devtype_copy =
0;
                        }
                    }
                    //Delete condition
                    else if (deleteButton.Checked == true && loop1[i].type !=
0)
                    {
                        loop1[i].cross_blue = false;
                        loop1[i].cond = 0;
                        loop1[i].select = true;
                        L1picBox[i].Image = null;
                        L1picBox[i].BackColor = L1lbl[i].BackColor =
L1labcom[i].BackColor = SystemColors.Control;
                        loop1[i].type = loop1[i].devtype = loop1[i].devanalogue
= 0;
                    }
                    //Double condition
                    else if (doubleButtom.Checked == true && loop1[i].type !=
0)
                    {
                        loop1[i].cross_black = !loop1[i].cross_black;
                        if (loop1[i].cross_black == true)//colocar cruz preta
                        {
                            loop1[i].cond = 7;
                            loop1[i].dobro = 1;
                        }
                        else if (loop1[i].cross_black == false)//retirar cruz
preta e colocar fundo anterior
```

```
                        {
                            loop1[i].cond = 6;
                            loop1[i].dobro = 0;
                        }
                    }
                    //efectuar contagem global
                    actualizar_contagem_loop1();
                    printlog(Convert.ToByte(i),        1,        loop1[i].cond,
loop1[i].type);//address , loop, condition, device
                    this.Refresh();
                }
            } …”
```