Sistemas Embutidos
CC4031

Uma Máquina Enigma Contemporânea

Levantamento de Requisitos

Grupo 7

António Cunha	up201805142
Diogo Gomes	up201805367
Joaquim Oliveira	up201908075
Tarso Caldas	up202204297

Porto — 2023

Conteúdo

1 *Introduction*

1.1 Enigma

A máquina Enigma, criada pelo engenheiro eletrotécnico alemão Arthur Scherbius em 1918 foi o aparelho eletromecânico de cifra mais popular do século XX devido à importância que teve em cenários e eventos militares de grande dimensão tais como a Guerra Espanhola e a segunda Grande Guerra.

Este aparelho criptográfico, teve diversas implementações dada a sua enorme adesão tanto a nível comercial, civil como militar. Nas versões militares alemãs (versão D) a máquina consistia num teclado com 26 teclas (para input), um quadro luminoso com as mesmas teclas (para descodificar as mensagens cifradas), um *plugboard* que consiste num painel de trocas para acrescentar uma camada «fraca» de ofuscação no texto de input e por fim 3 rotores que atuam como 3 cifras de César associadas a um refletor que corresponde posições simétricas do alfabeto para passar o texto cifrado uma segunda vez pelo circuito.

1.2 *Project objectives*

O seguintes objetivos irão servir de guia no *design* e desenvolvimento de uma *Enigma machine*:

- Pesquisar e analisar o *design* mecânico e eletrónico da *Enigma machine* original.
- Selecionar componentes eletrónicos apropriados, incluindo micro-controladores, fios, dispositivos de *input/output* e um teclado para a implementação da *Enigma machine*.
- Desenvolver um algoritmo de encriptação e desencriptação de mensagens que simula a encriptação original.
- Desenvolver uma interface usando um *display* e um teclado para introduzir e visualizar o processo de encriptação e desencriptação.
- Testar e validar as funcionalidades da *Enigma machine* usando uma variedade de testes e cenários.
- Avaliar a segurança da *Enigma machine* e identificar possíveis vulnerabilidades.
- Sugerir melhorias e modificações para melhorar a segurança e funcionalidade da *Enigma machine*.

2 *Requirements analysis*

Requirements analysis envolve identificar e definir requerimentos funcionais e não funcionais para assegurar que o produto final cumpre os objetivos finais.

2.1 *Functional Requirements*

1. *Input*: O sistema tem de permitir o utilizador introduzir uma mensagem de texto usando um conjunto de botões.
2. *Display*: O sistema tem de mostrar a mensagem enquanto esta é escrita usando um monitor digital LCD.
3. *Rotors*: O sistema tem de permitir o utilizador mudar a posição de cada rotor para que a mensagem seja desencriptada corretamente.
4. *Plugboard*: O sistema tem de simular a *plugboard* usando *jumper wires*.
5. *Encryption*: O sistema tem de encriptar a mensagem usando o algoritmo de Enigma, baseado na posição atual do rotor e das defenições da *plugboard*.
6. *Output*: O sistema tem de mandar esta mensagem encriptada usando serial.
7. *Power Supply*: O sistema tem de ter uma fonte de alimentação para alimentar todos os componentes.

2.2 *Non-Functional Requirements*

1. *Accuracy*: O sistema tem de precisamente implementar o algoritmo de Enigma e produzir mensagens corretas encriptadas.
2. *Security*: O sistema não poderá ser usado como uma maneira de comunicação segura.
3. *Usability*: O sistema tem de fácil de usar, com instruções claras e comandos intuitivos.
4. *Maintainability*: O sistema tem de ser fácil de manter, com documentação clara e componentes fáceis de trocar ou de atualizar.
5. *Compatibility*: O sistema precisa de ser compatível com componentes comuns, como monitores digitais, botões e fontes de alimentação, para facilitar a sua construção e manutenção.

3 *Modeling*

3.1 *Use Case Diagram*

O *Use Case Diagram* mostra a interação entre o utilizador e o sistema. Neste caso, o utilizador interage com a *Enigma Machine* introduzindo uma mensagem de texto.

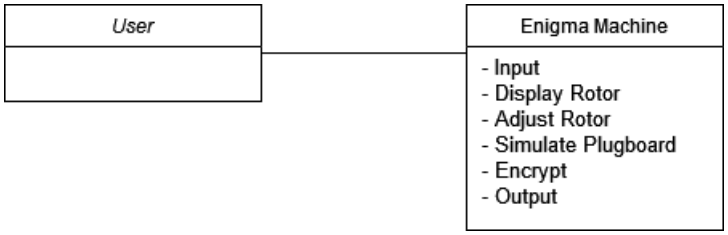

Figura 1: *Use Case Diagram*

3.2 *Class Diagram*

O *Class Diagram* mostra a estrutura do sistema e a relação entre diferentes classes. Neste caso, vamos dividir a *Enigma Machine* em quatro componentes: *input, output, encryption, and control*.

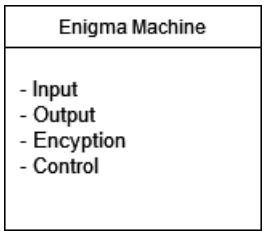

Figura 2: *Class Diagram*

Cada um destes componentes poderá ainda ser dividido e classes mais especificas. A componente *Input* irá incluir um conjunto de botões, o componente de *Output* irá incluir uma comunicação usando serial, a componente de *Encryption* irá incluir posições dos rotores e definições da *plugboard* e a componente de *Control* irá incluir botões e um monitor digital.

3.3 *Sequence Diagram*

O *Sequence Diagram* mostra a interação entre os diferentes componentes do sistema durante um processo especifico ou função. Neste caso, vamos mostrar um o processo de encriptar uma mensagem de texto na *Enigma Machine*

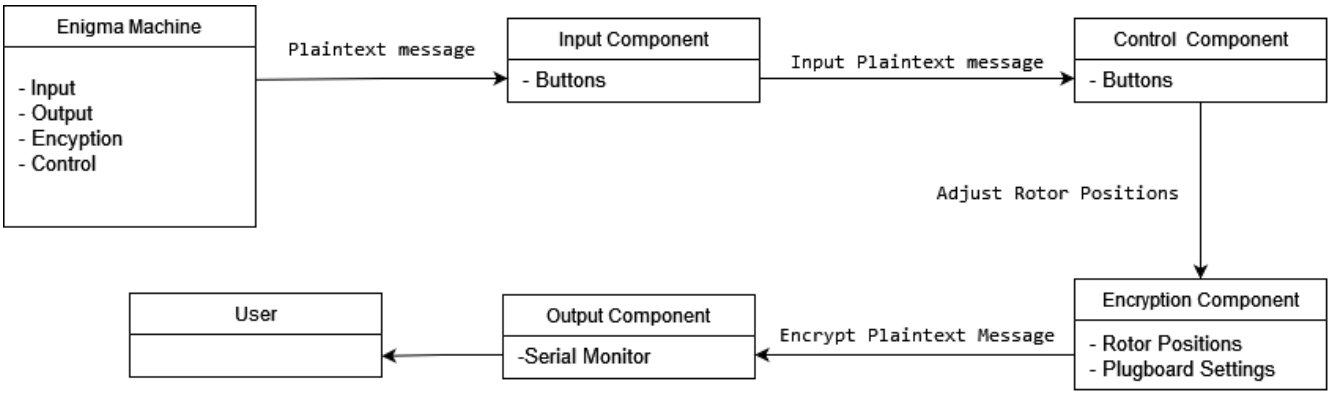

Figura 3: *Sequence Diagram*

Este diagrama mostra como a *Enigma Machine* interage com os componentes para encriptar uma mensagem de texto. O utilizador insere a mensagem usando botões e vê a mensagem num *serial monitor*.

4 *System architecture and specification*

O *System architecture and specification* define a estrutura global, comportamento e interação do sistema da *Enigma machine*, incluindo componentes de *hardware* e *software*, *data flow* e interação do utilizador, para assegurar que cumpre com os requisitos funcionais, requisitos não funcionais e objetivos do projeto.

4.1 *System Architecture*

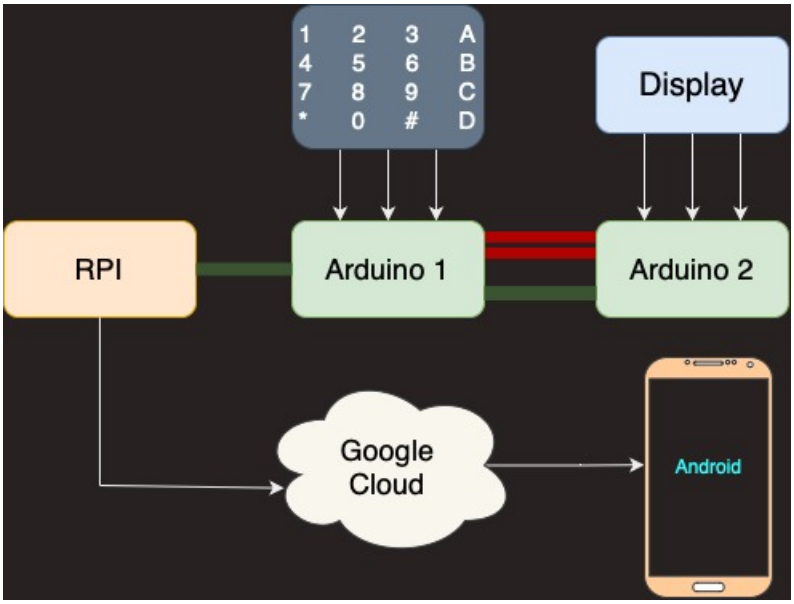

Figura 4: *System Architecture Diagram*

4.1.1 Components

- 2 Arduino Uno Rev3
- 1 LCD Keypad Shield for Arduino
- 1 keypad 4x4
- 1 Raspberry Pi
- 1 Android

4.2 *Logical Architecture*

1. *Input Processing Module*:
   - Converte caracteres de entrada nos seus correspondentes valores de *ASCII*.
   - Trata de caracteres especiais, como espaços e pontuação.
   - Prepara *input data* para o *plugboard module*.
2. *Plugboard Module*:
   - Permite o utilizador configurar o *plugboard* especificando os pares de letras a trocar.
   - Troca letras de acordo com a configuração do *plugboard*.
   - Envia informação para o *rotor module*.
3. *Rotor Module*:
   - Permite o utilizador configurar o *rotor* especificando a ordem e a posição inicial dos rotores.
   - Roda os rotores de acordo com a sua configuração e *input data*.
   - Encripta a informação passando pelo conjunto de rotores.
   - Envia informação para o *reflector module*.
4. *Reflector Module*:
   - Retorna a informação encriptada pelo conjunto de rotores no sentido oposto.
   - Envia a informação de volta pela *plugboard* para meter as letras na sua posição inicial.
   - Converte a informação de volta a caracteres com o seu *ASCII* correspondente.
5. *Output Processing Module*:
   - Envia o *ciphertext* para o *display*.
   - Fornece *feedback* ao utilizador.

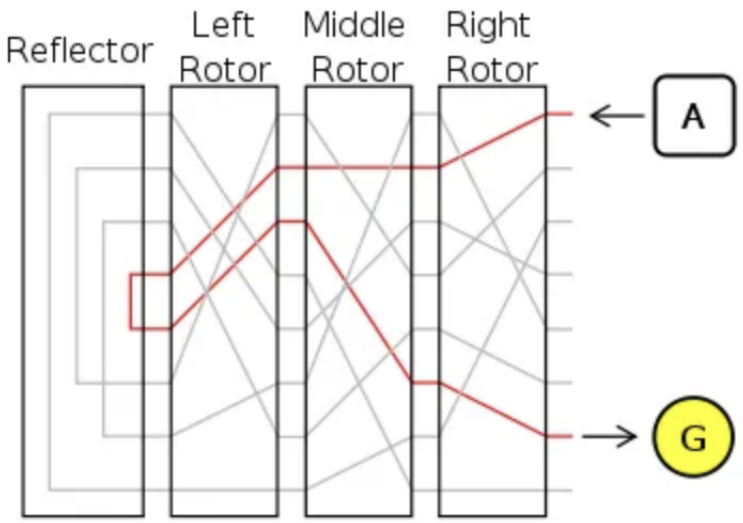

Figura 5: Método usado pela Enigma para encriptar um caratér

4.3 *Software Architecture*

1. *Input Processing Module*:
   - *Input Buffer*: Guarda temporariamente a informação introduzida pelo utilizador através do teclado, antes de ser processada.
   - *Input Parser*: Converte *user input* num formato que pode ser usado pelo *plugboard module*.
2. *Plugboard Module*:
   - *Plugboard Configuration*:
   - Permite o utilizador configurar a *plugboard* especificando os pares de letras a trocar.
   - *Plugboard Mapper*:
   - Troca as letras conforme a configuração da *plugboard*.
3. *Rotor Module*:
   - *Rotor Set Configuration*: Permite o utilizador configurar o *rotor* especificando a ordem e a posição inicial dos rotores.
   - *Rotor Set*: Roda os rotores de acordo com a configuração e *input data*.
   - *Rotor Map*: Mapeia letras de *input* para letras de *output* baseado na configuração do rotor.
4. *Reflector Module*:
   - *Reflector Configuration*: Permite o utilizador configurar o refletor especificando o mapeamento de letras.
   - *Reflector Map*: Mapeia letras de *input* para letras de *output* baseado na configuração do refletor.
5. *Output Processing Module*:
   - *Output Configuration*: Permite o utilizador configurar as definições do *display*.
   - *Output*: Mostra o *ciphertext* ao utilizador.

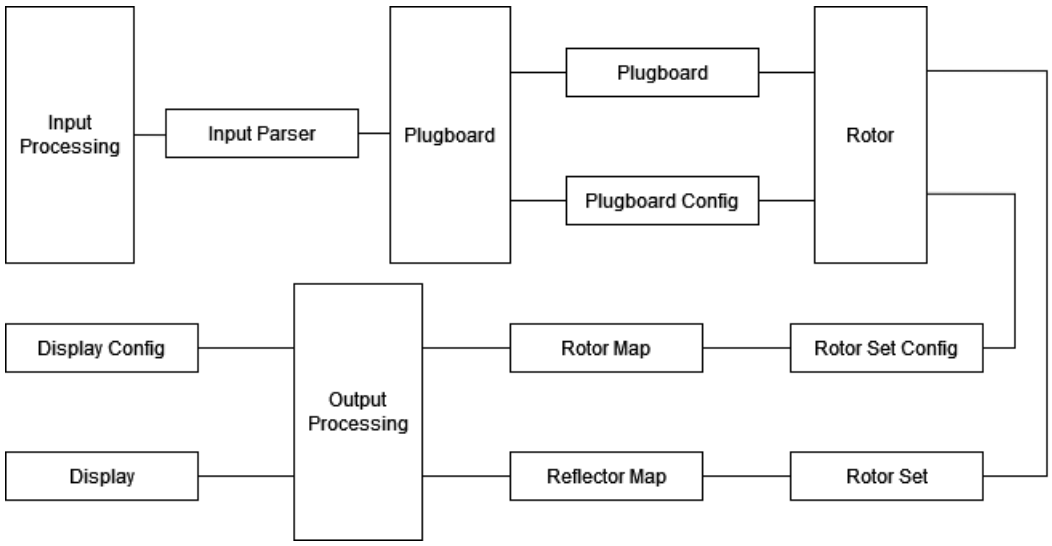

Figura 6: *Software Architecture Diagram*

4.4 *State Diagrams*

State diagrams são representações gráficas do comportamento do sistema ou de um componente, mostrando os diferentes estados e transições. No caso do nosso projeto, o comportamento do sistema a encriptar e desencriptar mensagens.

4.4.1 *Keyboard State Diagram*

State diagram para a componente do teclado terá os seguintes estados:

1. *Idle state*: Isto é o estado inicial do teclado quando nenhuma tecla está a ser pressionada.
2. *Key pressed state*: Quando uma tecla é pressionada, o teclado muda para este estado. Caso nenhuma tecla seja pressionada por muito tempo dá-se um timeout e o teclado volta ao estado inicial. Enquanto está neste estado o teclado espera que o utilizador escreva até que chegue um caractere de send.

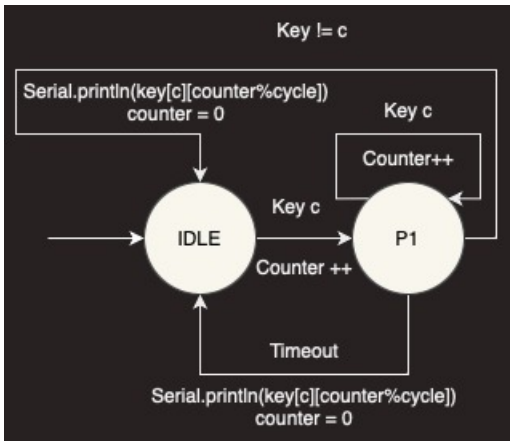

Figura 7: Funcionamento do teclado

4.4.2 *LCD Screen State Diagram*

State diagram para a componente do LCD Screen terá os seguintes estados:

1. *Idle state*: Isto é o estado inicial do LCD quando nenhum caractere foi recebido.
2. *Receiving State 1*: Assim que começa a receber caracteres o LCD dá display caractere a caractere até que chega ao fim da linha ou recebe um caractere de fim de texto. Caso chegue ao fim da linha passa para o próximo estado. Caso receba um caractere de fim de texto o ecrã é limpo e volta ao estado inicial.
3. *Receiving State 2*:Assim que começa a receber caracteres o LCD dá display caractere a caractere até que chega ao fim da linha ou recebe um caractere de fim de texto. Caso chegue ao fim da linha o ecrã é limpo e volta ao estado anterior. Caso receba um caractere de fim de texto o ecrã é limpo e volta ao estado inicial.

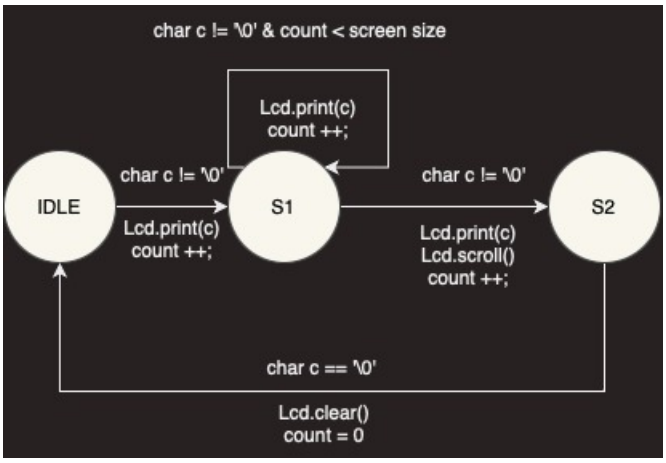

Figura 8: Funcionamento do display

5 Goals that were not Achieved

1. *Controlo de Rotores:* Devido a falta de tempo não foi possível acrescentar ao projeto um controlo de rotores por parte do utilizador quando escreve a mensagem, tinhamos inicialmente pensado em colocar um controlo de rotores no LCD Screen que, uma vez que tem um keypad incorporado, permitia controlar os rotores. Uma vez que tal não foi possível os rotores estão predefinidos no raspberry pi onde se dá a encriptação. O utilizador do Android por outro lado tem a opção de escolher os rotores para que se possa dar a desencriptação correta da mensagem.
2. *Plugboard:* Tal como os rotores, o plugboard foi implementado no raspberry pi em vez de ser simulado com jumper wires.
3. *Design:* Não foi possível construir, como dito no primeiro relatório, algo que de certa forma protege-se os componentes da nossa máquina e que trouxesse de certa forma alguma portabilidade ao nosso projeto

6 Links para o Git e Wiki

- Git: https://github.com/tarsobcaldas/projeto-enigma
- Wiki: https://github.com/tarsobcaldas/projeto-enigma/blob/main/README.md