Catarina Filipa Camacho Querido

Licenciada em Ciências da Engenharia Eletrotécnica e de Computadores

Money Link - Integração de Dispositivos Dispensadores de Moedas

Dissertação para obtenção do Grau de Mestre em

Engenharia Eletrotécnica e de Computadores

Orientador: Luís Filipe Santos Gomes,
Professor associado com agregação,
Universidade Nova de Lisboa

Júri

Presidente: Doutor João Francisco Alves Martins
Arguentes: Doutora Anikó Katalin Horváth da Costa
Vogais: Doutor Luís Filipe dos Santos Gomes

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2019

Money Link

Copyright © Catarina Filipa Camacho Querido, Faculdade de Ciências e Tecnologia, Universidade NOVA de Lisboa.
A Faculdade de Ciências e Tecnologia e a Universidade NOVA de Lisboa têm o direito, perpétuo e sem limites geográficos, de arquivar e publicar esta dissertação através de exemplares impressos reproduzidos em papel ou de forma digital, ou por qualquer outro meio conhecido ou que venha a ser inventado, e de a divulgar através de repositórios científicos e de admitir a sua cópia e distribuição com objetivos educacionais ou de investigação, não comerciais, desde que seja dado crédito ao autor e editor.

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Dedico esta dissertação aos meus pais.

Agradecimentos

A realização desta dissertação de mestrado contou com importantes apoios e incentivos sem os quais não se teria tornado uma realidade e aos quais estarei eternamente grata.

Em primeiro lugar quero agradecer ao orientador desta dissertação, o Professor Luís Gomes, da Faculdade de Ciências e Tecnologias da Universidade Nova de Lisboa. Antes de mais, agradeço pelo trato simples, correto e científico, com que sempre abordou as reuniões de trabalho sem nunca ter permitido que o desânimo se instalasse, mesmo que muitas vezes as tarefas não decorressem como o desejado. Esteve sempre disponível para ajudar a solucionar dúvidas e problemas que foram surgindo ao longo da realização deste projeto. Agradeço pelo apoio e orientação prestados, pelo incentivo e saber que transmitiu e por muitas mais razões, mas que por esse motivo motivos que tornaram possível a execução desta dissertação.

Também agradecer à empresa A-to-Be powered by Brisa, mais precisamente ao Dr. Rui Dias e ao João Ribeiro, em primeiro lugar pelo tema proporcionado e em segundo por me acolherem da melhor forma. Tornaram-se na minha equipa de trabalho, foram sempre muito prestáveis, transmitiram-me sempre todo o conhecimento, disponibilizaram-me documentos e a ajuda necessária para a elaboração e desenvolvimento prático da dissertação.

Aos meus pais e familiares, que graças à sua perseverança, dedicação, trabalho, educação e audácia permitiram-me entrar neste mestrado e tornaram-se na minha maior inspiração para me tornar como eles. É a eles que estou muito grata pela oportunidade que me proporcionaram de poder frequentar o ensino superior, pelos conselhos que me deram para me tornar numa pessoa melhor a nível pessoal, académico e profissional.

Um grande obrigada aos meus companheiros da faculdade, Ricardo Santos, Diogo Santos, Rodrigo Junqueira, Pedro Trabuco, André Barriga, Marco Gonçalves, João Silveira e Nuno Sequeira que desde o primeiro ano académico tiveram um grande impacto na minha vida, que me ajudaram a ultrapassar os melhores e piores momentos, eles que sempre apostaram no melhor para mim e juntos conseguimos formar uma excelente equipa de estudos e trabalhos, mantendo sempre o foco.

Por fim, e não menos importante, um especial agradecimento aos meus amigos do coração Bárbara Sereni, Raquel Santos, Rita Pereira, Margarida Martins, Thiago Pereira, Carolina Mendes, Joana Carlos e Rodrigo Martins que nunca duvidaram de mim, sempre me apoiaram , incentivaram e inspiraram, basicamente obrigada pela amizade e paciência.

Resumo

Fazer o manuseamento de dinheiro pode ser uma ação realizada por um humano, mas com o avanço das tecnologias tem vindo a aumentar o número de controladores e sistemas embutidos, sendo por fim máquinas responsáveis por realizar operações matemáticas, controlar o dinheiro e recolher ou dispensar o mesmo.

Neste caso em concreto, trata-se de estudar e desenvolver um dispositivo que seja capaz de controlar dinheiro em numerário para as máquinas de pagamento automático existentes nas portagens operadas pela Brisa. No presente momento, esse dispositivo já existe e já está implementado nas máquinas em operação, no entanto, foram comprados a empresas externas à Brisa pelo que limita em algumas funcionalidades.

Diante deste problema, a Brisa sentiu a necessidade de desenvolver um dispositivo que fosse ao encontro das suas necessidades e, com o restante sistema já em funcionamento, basicamente, criar um novo equipamento com base num já existente, mas com mais especificações e funcionalidades direcionadas às imposições da Brisa.

Esta dissertação incide no desenvolvimento desse equipamento com o auxílio da empresa A-to-Be powered by Brisa. Para tal, foram feitas diversas análises referentes a dispositivos já existentes, bem como a definição dos requisitos para o controlador a desenvolver com a elaboração da arquitetura do circuito elétrico.

Por fim, foram realizados testes e análises através de *software* desenvolvido ao longo do processo que permitiram verificar os cenários reais que o controlador tem que monitorizar durante a operação, e deste modo foi possível confirmar o correto comportamento do controlador.

O objetivo é desenvolver um controlador, *Money Link*, capaz de controlar e manusear dinheiro em numerário com o auxílio de um periférico, um *hopper*, através do protocolo de comunicação série ccTalk. Posteriormente, este dispositivo já estará equipado para operar com outros tipos de periféricos, pelo que basta desenvolver o *software* necessário e deste modo, completar todo o sistema de controlo de pagamentos em numerário.

Palavras-chave: Controlador, Protocolo de comunicação série ccTalk, Periféricos, Sistemas embutidos, *software*, Interface, *host*, *device*.

Abstract

Handling money can be done by a human, but with the advancement of technologies, the number of controllers and embedded systems has increased and finally machines are responsible for performing mathematical operations, controlling money and collecting or dispensing with it.

In this particular case it is a matter of studying and developing a device capable of controlling cash for the automatic payment machines on the tolls operated by Brisa. At the moment, this device already exists and is already implemented in the machines in operation, however, they were bought from companies outside Brisa so it limits some functionalities.

Faced with this problem, Brisa felt the need to develop a device that would meet their needs and the rest of the system already in operation, basically create new equipment based on an existing one but with more specifications and functionalities aimed at the impositions from Brisa.

This dissertation focuses on the development of this equipment with the help of A-to-Be powerd by Brisa. For this, several analyzes were made referring to existing devices, as well as the definition of the requirements for the controller to develop with the elaboration of the architecture of the electric circuit.

Finally, tests and analyzes were performed using software developed throughout the process that allowed to verify the real scenarios that the controller has to monitor during the operation, thus confirming the correct behavior of the controller.

The goal is to develop a controller, Money Link, capable of controlling and handling cash with the aid of a peripheral, a hopper, through the ccTalk serial communication protocol. Later, this device will be equipped to operate with other types of peripherals, so just develop the necessary software and thus complete the entire cash payment control system.

Keywords: Controller, ccTalk Series Communication Protocol, Peripherals, Embedded Systems, Software, Interface, Host, Device.

Índice

Lista de Figuras

Lista de Tabelas

Glossário

Barramento É o nome que se refere a um canal capaz de transmitir e receber dados entre diferentes componentes, de forma bi-direcional. É um termo que engloba *hardware* nomeadamente, cabos, fios e/ou fibra óptica.

Baud Rate É a medida de velocidade utilizada para a taxa de transmissão de *bits* na comunicação entre dispositivos, ou seja, é a taxa de transmissão de dados e indica o número de *bits* transmitidos por segundo..

Bit de Paridade O *bit* de paridade é utilizado para detetar erros de transmissão de dados de uma forma simples. O *bit* de paridade só tem dois estados, o par ou ímpar, se o numero de *bits* "1" da mensagem for ímpar adiciona-se o valor "1" no final da mensagem, se o numero de *bits* "1" da mensagem for par adiciona-se o valor "0" no final da mensagem. É detetado um erro de transmissão, por exemplo, se existir um numero ímpar de *bits* "1" na mensagem, e o valor do *bit* de paridade for "0", pois o *bit* de paridade devia de ter valor "1" .

Bluetooth É uma tecnologia capaz de ligar e trocar informações entre dispositivos a curta distância. Este protocolo funciona de forma segura através de emissão de rádio frequência de baixa potência.

Buffer O *buffer* encontra-se na entrada do dispositivo de *hardware*, neste dispositivo ficam alocados os dados de forma temporária até que estes sejam transferidos para outro lugar para serem processados. Neste contexto, o *buffer* foi desenvolvido no âmbito de melhorar o congestionamento de transferência de dados entre a porta de entrada e de saída.

Dispositivo São os componentes que, no nosso caso, se ligam ao *host* e são monitorizados pelo *host*. Neste contexto, um dispositivo pode ser um aceitador de moedas, um dispensador de moedas, etc.

Half-Duplex Quando nos referimos a um protocolo *half-duplex* estamos a referir-nos à comunicação entre um transmissor e um recetor, no entanto ambos podem receber ou transmitir dados, mas nunca em simultâneo.

Handshaking Também conhecido por aperto de mão, que ocorre quando dois dispositivos se reconhecem e estão aptos para proceder à comunicação. Este método de reconhecimento é utilizado nos protocolos de comunicação.

Host É um termo utilizado no mundo da informática e representa a máquina principal que interliga outros dispositivos, de modo a montar uma rede, o *host* dispõe de informações, recursos, serviços e aplicações aos dispositivos ligados a este.

MDB *Multi-Drop Bus* é um protocolo de comunicação em série em que todos os dispositivos estão ligados a um único barramento de dados e o *host* manda a mensagem para todos e apenas o dispositivo respetivo é responsável por responder.

Microcontroladores PIC Microcontroladores são circuito integrados, que possuem um processador, pinos de entrada e saída e memória. Estes circuitos integrados são capazes de efetuar operações e tarefas uma vez que são programados e desenvolvidos para tal. O PIC utilizado neste trabalho é um microcontrolador desenvolvido pela Microchip Technology Inc..

RS-232 Trata-se de um protocolo de comunicação em série.

Start Bit É utilizado em comunicações assíncronas entre dispositivos e é o *bit* responsável por indicar ao recetor que vai ser iniciado o encaminhamento de dados.

Stop Bit É o *bit* responsável por indicar que um *byte* de dados acabou de ser transmitido, ocorre em comunicações assíncronas, ou seja, é o *bit* que informa que terminou a transmissão de um *byte*.

Siglas

ACK *Acknowledgement*.

ASCII American Standard Code for Information Interchange.

ATPM *Automatic Toll Payment Machine*.

BCD Binary Coded Decimal.

CC *Cash Controls*.

CMF Cryptographic Mapping Function.

COM *Communication Port - Porta Série*.

CPU *Central Process Unit*.

CRC Cyclic Redundancy Check.

DES Data Encryption Standard.

FPU *Floating-Point Unit*.

I/O Input/Output.

IDE Integrated Development Environment.

NAK Not Acknowledged.

PC *Personal Computer*.

RAM *Random Access Memory* ou Memória de Acesso Aleatório.

RF *Radio frequency*.

ROM *Read-Only Memory* ou Memória só de Leitura.

RX Recetor.

SDK *Kit* de desenvolvimento de *software*.

SUH Serial Universal Hopper.

TX Transmissor.

UART *Universal asynchronous receiver/transmitter*.

USB *Universal Serial Bus*.

VCOM *Virtual Communication Port - Porta Série*.

Capítulo 1

Introdução

1.1 Motivação e Enquadramento

Nos dias de hoje, é possível encontrar uma máquina de pagamento automático em diferentes tipos de estabelecimentos, tais como portagens, supermercados, parques de estacionamento, quiosques de venda de bilhetes para transportes públicos, máquinas automáticas de venda de alimentos e bebidas, entre outras. Estas máquinas de pagamento automático vieram trazer diversas vantagens, nomeadamente a facilidade, a segurança, a rapidez e a precisão com que são efetuados os cálculos matemáticos para efetuar a operação de troco, com o avanço da tecnologia geraram-se novos postos de trabalho para desenvolvimento e aperfeiçoamento destas máquinas, no entanto veio reduzir outros postos de trabalho, nomeadamente de algumas pessoas que geriam e faziam o controlo destes sistemas de forma manual.

As portagens operadas pela Brisa, em Portugal, já fazem parte das nossas vidas desde 1972 e nos dias de hoje já controlam cerca de 1 628km das vias portuguesas [11], tendo a empresa uma expansão a nível internacional[15].

Numa fase inicial, o pagamento era apenas realizado através de um operador, este estava responsável pela categorização do veículo, pela gestão e controlo do dinheiro.

No entanto, com o aumento do número de veículos a circular nas vias rodoviárias, esta solução originava longas filas de espera. Com o objetivo de solucionar este problema, a Universidade de Aveiro, em conjunto com outras entidades, desenvolveu em 1991 um sistema com o nome de Via Verde [46], que permite aos clientes que adotem este método possuírem um identificador no veículo que contém a informação dos dados da viatura, como a matrícula e categoria. Quando o veículo passa na portagem, os sensores categorizam-no e verificam se a matrícula e a categoria correspondem à mesma informação presente no dispositivo e o pagamento é feito de maneira automática através da associação ao cartão

de multibanco do utilizador.

Em meados de 2011, surgiram as máquinas de pagamento automático nas portagens. Este sistema foi implementado de forma a ser possível efetuar o pagamento de modo mais rápido e seguro através de dinheiro ou multibanco. Inicialmente, os operadores temeram pelos seus postos de trabalho [41], no entanto não houve grandes consequências com a inserção desta nova tecnologia, além de que, com a utilização desta tecnologia, foi possível os operadores começarem a comemorar os feriados festivos mais importantes, coisa que antes era impossível, pois não havia forma de ser feito o controlo dos veículos sem ser manualmente.

Com esta situação, surge a ideia de criar um sistema autónomo e produzido na totalidade pela empresa - A-to-Be powered by Brisa. Esta é pois uma empresa associada à Brisa, sendo responsável por desenvolver e melhorar soluções tecnológicas para prestadores de serviços de mobilidade.

1.2 Âmbito e Objetivos

Esta dissertação, explorada em parceria com a empresa A-to-Be powered by Brisa, consiste no desenvolvimento de um sistema capaz de controlar as máquinas de pagamento automático nas portagens operadas pela Brisa

No presente momento, a A-to-Be recorre a empresas externas para adquirir os equipamentos necessários a fim de construir a máquina de pagamento automático. Sendo um dos equipamentos o controlador da máquina, e sendo esse comprado a empresas externas, traz como consequência uma certa limitação no que toca à implementação de determinadas funções. Para tal, a A-to-Be reconheceu a necessidade de produzir um controlador que correspondesse às especificações e necessidades praticadas pela Brisa, havendo deste modo uma menor dependência de empresas externas e possibilitando atingir um mercado maior. O controlador passará a designar-se de *Money Link*.

Face ao exposto, a A-to-Be propôs este tema de dissertação à qual consiste na elaboração de um controlador para as máquinas de pagamento automático. Ao desenvolver um controlador para as máquinas de pagamento automático, este tem o objetivo de assegurar o controlo do dinheiro em numerário.

Numa primeira fase, realizou-se um levantamento dos elementos a utilizar no sistema, para tal, foi necessário fazer um estudo de algumas soluções já existentes no mercado de modo a averiguar quais os tipos de periféricos, características, protocolos de comunicação e custos que pudessem ser importantes e úteis para implementar neste controlador, deste modo, foi possível definir a arquitetura do sistema, tanto a nível de *software* como de *hardware*. Após ter delineado quais os dispositivos a incorporar no sistema, elaborou-se o desenho do circuito com todos os elementos do sistema, envolvendo as vertentes eletrónicas e mecânicas com o suporte da empresa A-to-Be.

Numa segunda fase, utilizando um controlador protótipo e já com as diversas interfaces implementadas, deu-se inicio à fase de programação do sistema embutido, permitindo

a implementação nos diferentes cenários com que os produtos da empresa lidam na vida real. A vertente de *software* consiste, maioritariamente, em garantir a comunicação entre o controlador e os periféricos, sendo que o controlador tem que ter a capacidade para interpretar, controlar e monitorizar as mensagens recebidas e enviadas, a fim disso, realizou-se o estudo para todas as possíveis situações.

1.3 Organização e Estrutura da Dissertação

Esta dissertação está descrita em seis capítulos. No primeiro capítulo é feita uma breve introdução sobre o que foi idealizado e realizado ao longo deste projeto bem como os principais objetivos do mesmo.

No segundo capítulo é realizado um levantamento dos periféricos utilizados nas máquinas de pagamento automático, assim como os seus objetivos e funcionalidades. Também é apresentado o estudo de algumas soluções já existentes e que se assemelham ao sistema a ser desenvolvido, por forma a contextualizar as tecnologias e implementações já abordadas.

O terceiro capítulo descreve detalhadamente o protocolo de comunicação ccTalk utilizado para transmitir mensagens entre o controlador e os periféricos. Este capítulo assume grande ênfase, pois o desenvolvimento de *software* incide na maioria em torno desta informação e regras de implementação.

O quarto capítulo contém a solução proposta e é possível fazer uma análise mais aprofundada de duas soluções já existentes e que mais se assemelham à solução a desenvolver, fazendo assim uma comparação entre os controladores e, por sua vez, a otimização e definição dos elementos para o controlador implementado, o *Money Link*. Além disso, é apresentada a arquitetura do circuito elétrico do controlador *Money Link*, dando mais uma vez ênfase ao circuito da interface ccTalk.

No final do quarto capítulo, é apresentado o ambiente de desenvolvimento, ou seja, as ferramentas, recursos e algoritmos utilizados na implementação do software, nomeadamente o Integrated Development Environment (IDE), a *board* protótipo, alguns testes e resultados preliminares, bem como os seus casos de uso.

No quinto capítulo, aborda-se a implementação realizada para o sistema, bem como todos os resultados obtidos a nível experimental, tanto num ambiente virtual como num ambiente real, com a integração de um periférico, neste caso, um *Hopper* e da interface ccTalk, assim como de testes e validação.

O sexto capítulo contém as conclusões sobre todo o trabalho desenvolvido, bem como críticas positivas e construtivas e, para terminar, alguns desenvolvimentos que possam ser realizados no futuro.

No fim do documento encontra-se a bibliografia consultada durante a escrita e desenvolvimento da dissertação.

Capítulo 2

Dispositivos Utilizados e Soluções Existentes

As máquinas de pagamento automático são tratadas como um sistema completo e por isso têm diversos equipamentos diferentes a operar em conjunto a fim de executar todo o procedimento, percorrendo todos os equipamentos necessários para determinada ação e sempre monitorizado pelo controlador. Deste modo, será apresentado uma secção relativamente aos dispositivos utilizados nas máquinas de pagamentos operadas pela Brisa, bem como as suas funcionalidades.

É também importante mencionar soluções atuais, uma vez que o controlador *Money Link* será uma otimização e melhoramento de alguns destes sistemas já existentes no mercado.

2.1 Dispositivos utilizados no Sistema

Como já foi dito anteriormente, uma máquina de pagamento automático é um sistema constituído por diversos dispositivos e recursos a fim de corresponder às operações do sistema a implementar.

No caso desta dissertação, o sistema a implementar é uma máquina de pagamento automático para o controlo de dinheiro direcionado às portagens operadas pela Brisa, pelo que, nesta secção serão apresentados os periféricos base para se obter um sistema completo que corresponda às expectativas.

No mercado, há várias marcas e empresas que desenvolvem e exploram estes dispositivos, nomeadamente *Crane Payment Innovations* [16], Grupo Azkoyen [26] e JCM Global [32]. Abaixo, serão apresentado os vários modelos de equipamentos, bem como a sua função e o tipo de protocolo de comunicação que utilizam de modo a informar o controlador *host*.

O protocolo de comunicação é utilizado com objetivo de haver a transmissão e receção de mensagens entre o controlador e o periférico, estas mensagens podem ser apenas de teor informativo, como podem ser de pedidos de tarefas, entre outras funcionalidades coordenadas pelo controlador *host*.

- **Aceitador de moedas:**
  - Descrição: os aceitadores de moedas são responsáveis por aceitar as moedas mas também por verificar a veracidade das mesmas, comparando as propriedades físicas da moeda com os requisitos exigidos pelo dispositivo, sendo estas características o peso, o tamanho, entre outros. [30]
  - Protocolo de Comunicação: ccTalk e MDB [45] [37].
- **Reciclador de moedas:**
  - Descrição: o reciclador de moedas, também conhecido como dispensador de moedas, é o dispositivo responsável por processar transações, contagem e balanceamento e, por fim, concluir a transação disponibilizando troco caso seja necessário. [35]
  - Protocolo de Comunicação: ccTalk e MDB.
- ***Hopper* de moedas:**
  - Descrição: o *hopper* de moedas não é capaz de operar sozinho, funciona com o auxílio do aceitador de moedas e do reciclador de moedas. O *hopper* de moedas tem como princípio o encaminhamento das moedas entre o aceitador e o dispensador. Nos modelos mais recentes, os produtores deste tipo de equipamento optam por juntá-lo todos num único dispositivo, tornando-o mais eficiente quanto à transferência de moedas entre o aceitador de moedas e o dispensador de moedas. [8] [43]
  - Protocolo de Comunicação: ccTalk.
- **Aceitador de notas:**
  - Descrição: o funcionamento de um aceitador de notas é semelhante ao funcionamento do aceitador de moedas, ou seja, recebem notas e verificam a a autenticidade das que forem introduzidas; este só aceita uma nota de cada vez. Este tipo de dispositivo dispõe de vários sensores óticos e eletromagnéticos para garantir a veracidade das notas, portanto, quanto mais sensores forem aplicados, maior será a fiabilidade e mais rápida será a operação de verificação, no entanto, mais dispendioso é o equipamento. [27]
  - Protocolo de Comunicação: ccTalk, RS-232, MDB e ID003.

- **Reciclador de notas:**
  - Descrição: o reciclador de notas é um dispositivo que aceita notas, verifica a sua autenticidade, tal e qual como o aceitador de notas, residindo a diferença no facto de, neste dispositivo, ser possível distribuir as notas por diferentes compartimentos. Este tipo de equipamento tem a vantagem de poder receber, efetuar cálculos, armazenar e dispensar notas para a eventualidade de ser necessário efetuar troco. [36]
  - Protocolo de Comunicação: ccTalk, MDB e ID003.
- **Dispensador de notas:**
  - Descrição: este tipo de dispositivo é muito semelhante ao anterior com a diferença que o dispensador de notas está apto a dispensar quantidades maiores de notas, deste modo apenas é necessário efetuar uma única operação para ceder as notas. [28]
  - Protocolo de Comunicação: ccTalk
- **Leitor do cartão bancário:**
  - Descrição: nos dias de hoje é possível fazer a leitura do cartão através da banda magnética, do CHIP e por contacto. Na maioria dos casos, este tipo de pagamento não requer PIN, no entanto, o dispositivo incorpora uma secção onde é possível o utilizador inserir o PIN do cartão, caso seja necessário. [31]
  - Protocolo de Comunicação: RS-232 e ccTalk
- **Impressoras:**
  - Descrição: têm a função de imprimir um talão da operação efetuada e fornecê-lo ao utilizador.
  - Protocolo de Comunicação: ccTalk

2.2 Soluções Atuais

Das soluções apresentadas nesta secção há que dar especial atenção a dois modelos nomeadamente o *Paylink* e o KS1000 desenvolvido pela empresa e^{2}c.

O *Paylink* é o controlador dos subsistemas de numerário presente nas máquinas de pagamento automático das portagens operadas pela Brisa em Portugal, cujo nome é eToll, ou seja, este dispositivo apenas controla operações em numerário. No entanto, há ainda um outro dispositivo denominado de *Lane Controller*, sendo este o controlador responsável por monitorizar todo o sistema.

A *Automatic Toll Payment Machine* (ATPM) é uma máquina que está a ser vendida pela a A-to-Be para os Estados Unidos da América e ,neste momento, está presente em três

estados, mais especificamente em Illinois, com mais de 100 máquinas em operação, no estado de Virgínia, com 12 em operação, e a explorar mercado na Califórnia. Na ATPM, o modelo KS1000 é o *Deck Controller*, ou seja, cada controlador KS1000 monitoriza um nível da máquina, sendo depois orquestrados por outra máquina dentro da ATPM - o ATPM *Controller*.

A ATPM comparada com o eToll, caracteriza-se por ser um equipamento independente, enquanto os equipamentos individuais do eToll são diretamente controlados pela via, neste caso via consiste no *Lane Controller*, controlador de via. Deste modo, tem-se que o *Personal Computer* (PC) que corre o *software* gera uma via de portagem.

2.2.1 Milan/Paylink Interface

Esta é uma solução desenvolvida pela empresa *Aardvark* [1] e representa um sistema responsável pelo controlo de dinheiro e pela comunicação entre diferentes dispositivos. A interface *Milan* é produzida e vendida pela *Money Controls* como *Paylink*. Este é o equipamento utilizado nos dias de hoje nas máquinas de pagamento automático nas portagens de Portugal, sendo o nome destas máquinas ETOL.

O *Paylink* é um sistema simples embutido que, para além de ser responsável por fazer a comunicação entre os diferentes periféricos, também apresenta uma interface capaz de comunicar entre um PC e os equipamentos utilizados no sistema para a gestão do dinheiro.

A Figura 2.1 representa o controlador produzido pela *Aardvark*, tratando-se de um micro- controlador de 16 *bits* que é capaz de comunicar com os diferentes dispositivos através de diversos protocolos de comunicação, tal como ccTalk, MDB, ID003 e MFS.

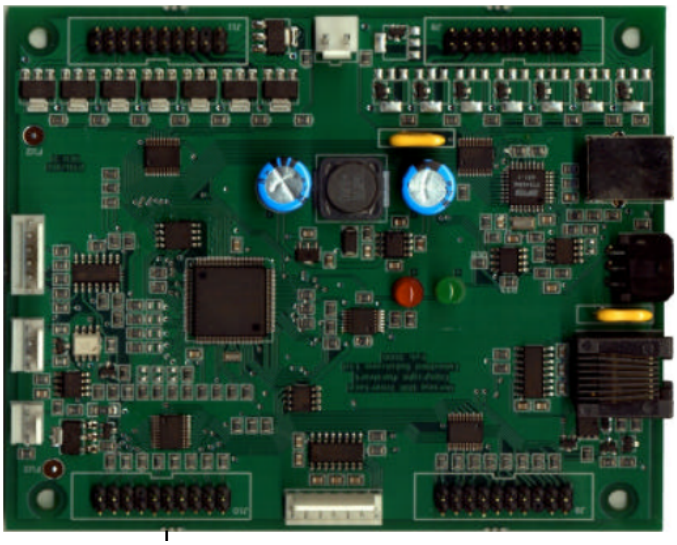


Figura 2.1: Interface *Paylink*

Este controlador funciona como uma interface sendo considerada como uma única unidade que se liga através de uma porta *Universal Serial Bus* (USB) a um PC e em conjunto

dedicam-se ao controlo e gestão dos pagamentos das máquinas. O *Paylink* dispõe de *software* básico para que o sistema seja capaz de efetuar as operações de pagamento e comunicação entre dispositivos sem que seja necessária a intervenção do utilizador para criar *software*, no entanto, o utilizador pode sempre fazer as alterações e adaptações ao *software* de modo a tornar o sistema mais eficiente ou completo. [42] [3] [4]

O *Paylink* é um controlador acessível para qualquer tipo de periférico, assim sendo, sempre que é desenvolvido um novo produto, a equipa técnica de *software* de *Aardvark* desenvolve novos *scripts* e basta que o utilizador que utilize esse novo periférico, faça o *download* do mesmo e atualize o controlador através da ligação USB entre PC e controlador.

O registo de todas as operações efetuadas pelo controlador fica guardado na memória interna não volátil, como por exemplo, num cartão SSD ou numa memória *flash*.

Na tabela abaixo, 2.1, é possível ver quais os periféricos suportados pelo controlador e o respetivo protocolo de comunicação entre o controlador e o periférico.[7] [6]

Tabela 2.1: Periféricos suportados pelo *Paylink* e o respetivo protocolo de comunicação

Nome do dispositivo	**Protocolo de comunicação**
Aceitadores de moedas	ccTalk
Recicladores de moedas	MDB
Hopper de moedas	ccTalk
Aceitadores de notas	ccTalk, ID003, MDB
Recicladores de notas	ID003
Dispensadores de Notas	MFS

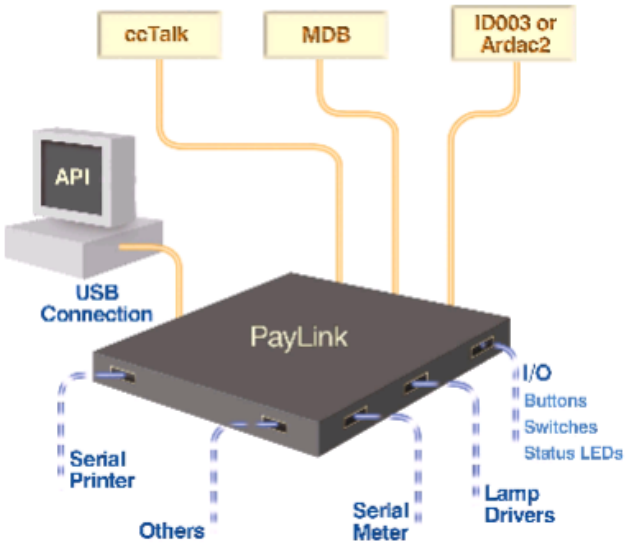


Figura 2.2: Esquema representativo da implementação do controlador *Paylink*

Como se pode verificar na Figura 2.2, o *Paylink* suporta periféricos que comuniquem através do protocolo de comunicação série ccTalk e MDB [2], e aceita dois periféricos que comuniquem através do protocolo de comunicação série RS-232. A ligação entre o *Paylink* e o PC é feita através de USB. A interface apresenta também oito portas de saída de alta potência e oito portas de saída de baixa potência, dezasseis portas Input/Output (I/O). [5]

Quando ao nível de segurança, o *Paylink* dispõe de criptografia no *software* de modo a que a comunicação entre dispositivos são seja intercetada ou, na eventualidade de serem encaminhados dados para o dispositivo errado, deste modo, através da criptografia, só o dispositivo responsável por responder ou executar alguma função é que vai corresponder ao código encriptado. Assim são garantidos todos os métodos de comunicação segura.

2.2.2 e^{2}c

A e^{2}c é uma empresa que desenvolve soluções de sistemas embutidos para máquinas de pagamento automático, máquinas de jogos e *smart vending*.

Nesta secção serão apresentados três tipos de controladores produzidos pela empresa e^{2}c, desde o mais simples até ao mais completo. As três soluções são:

- CC100
- CC200
- KS1000

Os controladores em série *Cash Controls* (CC) são *smart hubs* e são produzidos pela e^{2}c, são desenvolvidos a fim de interligarem os periféricos que efetuam ações de pagamento e um PC através de uma ligação USB, utilizando a aplicação iSocket *Payment Device Manager* para a gestão dos periféricos, ou seja, todas estas soluções operam em conjunto com uma aplicação, iSocket, de modo a implementar a interface entre o PC e o controlador e deste modo teremos o sistema embutido completo. [21]

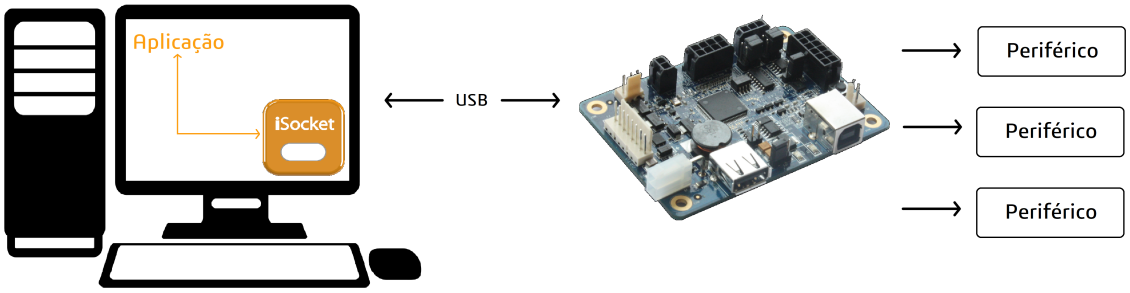


Figura 2.3: Esquema representativo de um sistema desenvolvido pela e^{2}c [22]

Na Figura 2.3 possível ver um exemplo de um sistema desenvolvido pela e^{2}c. A ligação entre o controlador e os periféricos é feita através uma ligação USB e este suporta todas as interfaces físicas necessárias, como o ccTalk ou MDB.

2.2.2.1 iSocket

O iSocket é a aplicação utilizada nos sistemas desenvolvido pela empresa e^{2}c [23]. É uma aplicação *middleware* que permite estabelecer a interface entre os periféricos, controlador e PC, ou seja, serve como um único canal entre todos os dispositivos do sistema, tendo em vista o controlo do pagamento das portagens e de todas as operações em volta desse processo.

Sendo uma aplicação completa, é responsável pela gestão e otimização dos dispositivos de modo a ter-se um sistema seguro e tolerante a falhas. O iSocket foi desenvolvido para que este fosse compatível com uma vasta gama de periféricos, sendo que este suporta os mais variados e complexos dispositivos, além disso, admite todas as principais ferramentas de desenvolvimento de aplicativos.

No que toca ao desenvolvimento de *software*, o iSocket tem vários exemplos disponíveis para auxílio ao utilizador e está apto a executar *scripts* de testes de modo a garantir as funcionalidades do dispositivo. No entanto, todas as funções do sistema podem ser verificadas sem a utilização da aplicação.

Esta aplicação controla e regista todas as comunicações realizadas entre dispositivos, deste modo é possível obter um maior controlo sobre todas as ações e, no caso de ocorrer uma falha em algum evento ou ação, a aplicação saberá onde agir, consequentemente obtém-se melhor desempenho operacional e reduz-se o risco de erros.

2.2.2.2 CC100

O CC100 é um controlador utilizado para a gestão e controlo do dinheiro, sendo esta uma versão reduzida do controlador CC200 apresentado na secção seguinte, 2.2.2.3. Ou seja, o controlador CC100 da e^{2}c apresenta um método simples da interface e controlo dos dispositivos. [18]

Como se pode verificar na Figura 2.4, é possível observar uma interface USB que está destinada para o PC, *host*, do qual o *host* apresenta um processador ARM ou X86. O CC100 é capaz de ligar até dezasseis dispositivos em simultâneo, sendo esses validadores de notas e moedas e *hopper* de moedas. Os dispositivos comunicam através do protocolo de comunicação série ccTalk ou MDB e via USB.

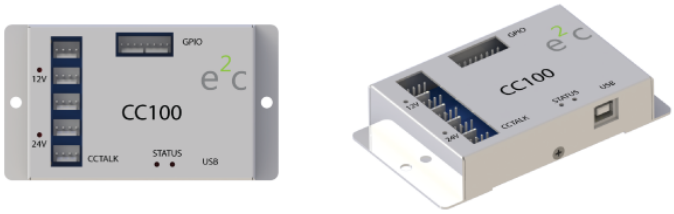


Figura 2.4: Controlador CC100

Pode-se observar na Figura 2.5, como é realizado, de uma forma geral, a implementação do controlador com os dispositivos.

Figura 2.5: Implementação típica do controlador CC100

Como referido anteriormente e não sendo exceção, o controlador CC100 faz uso da aplicação iSocket como *middleware*, esta aplicação é executada num PC, de modo a lidar com a comunicação entre dispositivos, através do protocolo de comunicação série ccTalk. Assim, é possível alcançar o controlo de todas as ações executadas neste sistema tornando-o tolerante a falhas.

O iSocket está habilitado a abstração de *hardware* de todos os dispositivos de baixo nível, ou seja, todos os programas de baixo nível já estão inseridos na aplicação, de modo a ser possível a rápida integração dos periféricos no sistema. Caso seja necessário implementar cenários mais específicos, será essencial adaptar o código.

Para concluir, das três soluções desenvolvidas pela e^{2}c, esta é que apresenta um menor custo, porém, não é possível conectar periféricos destinados a pagamentos bancários, tornando assim esta solução bastante limitada para a finalidade desta dissertação.

2.2.2.3 CC200

O CC200, mostrado na Figura 2.6, é outra solução desenvolvida pela e^{2}c para o controlo e gestão dos dispositivos das máquinas de pagamento automático, no entanto também é possível ser utilizada noutras tecnologias, como máquinas de jogos. Esta é uma solução melhorada da versão do CC100, uma vez que neste equipamento é possível agregar uma maior variedade de periféricos que, no caso do CC100, não era permitido, tal como leitor de cartões bancários, leitor de CHIP , impressora, dispositivos de segurança, etc. [19] [20]

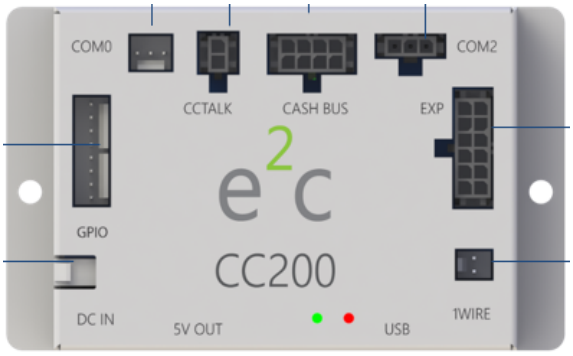


Figura 2.6: Controlador CC200

Também este controlador faz uso da API iSocket, tendo como objetivo as mesmas metas a cumprir que o CC100, com a diferença que, neste caso, o iSocket está preparado para executar operações mais complexas, como o manuseamento de operações com multibanco ou impressão de um talão. Para tal, o iSocket possui *scripts* básicos para efetuar todas essas ações, caso seja necessário implementar cenários mais avançados, será preciso fazer alterações no código.

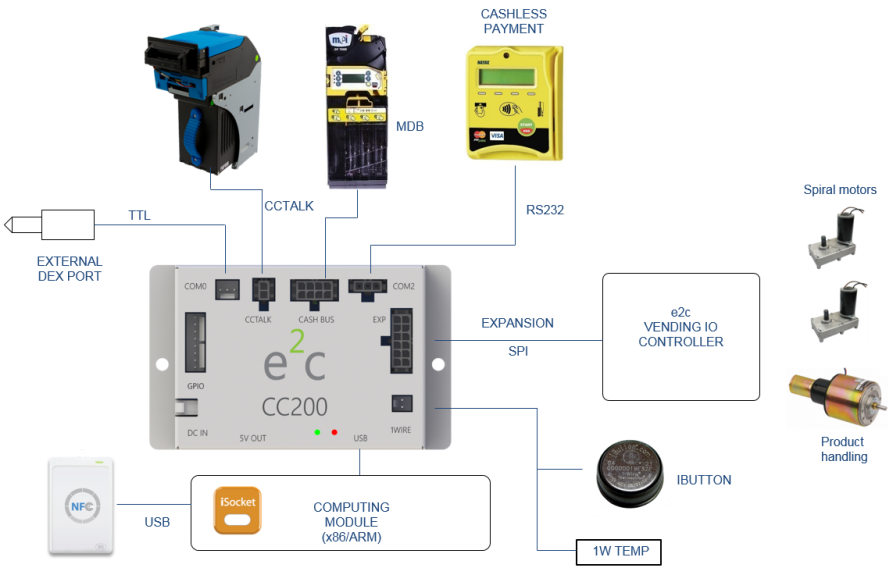


Figura 2.7: Implementação típica do controlador CC200

Na Figura 2.7, está representado o modo como são feitas as ligações do sistema. Também este controlador se liga ao *host* através de uma ligação USB; quanto aos restantes periféricos, estão disponíveis diversas interfaces físicas de modo a estabelecer a ligação.

2.2.2.4 KS1000

O KS1000, representado na Figura 2.8, é das soluções mais inovadoras presentes no mercado nos dias de hoje, sendo uma das soluções mais completas, pois é a que abrange uma maior gama de dispositivos, dos mais simples até aos mais avançados. [24]

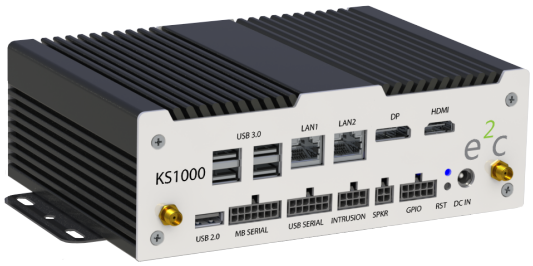


Figura 2.8: Controlador KS1000

Este controlador trabalha em conjunto com um processador Intel x86 Quad que, quando comparado com as outras soluções aqui apresentadas, é o que apresenta um processador mais eficaz. Ao desenvolver este novo controlador, a e^{2}c integrou vários recursos de segurança, de modo a obter um registo de transações, com um *backup* por bateria e processamento de criptografia para autenticação de *software*. Além disso, no KS1000 pode ser conectado um monitor. [17] [25]

O KS1000 é tão poderoso que em conjunto com o iSocket é possível ligar mais de 250 periféricos.

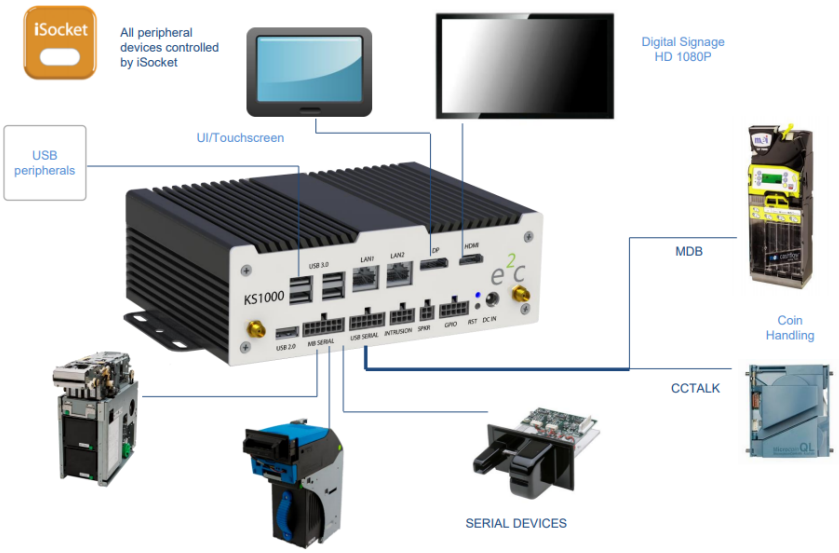


Figura 2.9: Implementação do controlador KS1000

O KS1000 é então um controlador com monitor duplo incorporado e apresenta um design compactado e sem ventilador. Outra das vantagens é que este equipamento dispõe de placa de rede, sendo possível incluir LAN, Wi-Fi e 3G.

Como já referido anteriormente, esta é das soluções mais inovadoras presentes no mercado, no entanto, este equipamento é o mais dispendioso.

Capítulo 3

Protocolo de comunicação série ccTalk

O ccTalk é o protocolo de comunicação série que é utilizado pela maioria das soluções já existentes no mercado e também será o protocolo utilizado no sistema a ser desenvolvido nesta dissertação. Sendo um protocolo aberto, é possível fazer as alterações e adaptações necessárias de modo a obter mais eficiência em cada sistema proposto nos capítulos anteriores. [10]

O ccTalk é um protocolo de controlo e é responsável por comunicar entre dispositivos e periféricos da rede.

3.1 Comunicação Série Vs. Comunicação Paralelo

- Comunicação em paralelo – Esta comunicação, Figura 3.1, é um tipo de transmissão de dados em que os vários *bits* de dados são enviados todos ao mesmo tempo através de vários canais de comunicação a interligar os periféricos. Comparando com a comunicação em série, estas interfaces apresentam uma maior velocidade de transmissão de dados e em alguns dispositivos apresentam uma forma mais simplificada para transmitir informação. No entanto, é um método que por vezes exige um elevado número de cabos implicando um aumento dos custos e pode apresentar problemas de sincronização.

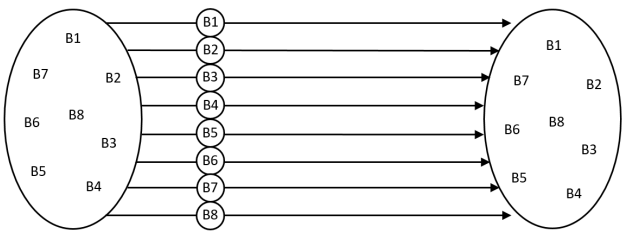


Figura 3.1: Comunicação em Paralelo

- Comunicação em série – Esta comunicação, Figura 3.2, é um tipo de transmissão de dados entre dispositivos, onde os *bits* são enviados um de cada vez sequencialmente, através de um Barramento ou canal de comunicação. Neste modo de comunicação, não há problemas de sincronização e os custos não são tão elevados. A comunicação em série permite ligar dois ou mais periféricos de forma simples e eficiente, reduzindo ainda mais os custos de produção, uma vez que não é necessário que os diferentes dispositivos não sejam ligados a um único controlador *Host*. [13]

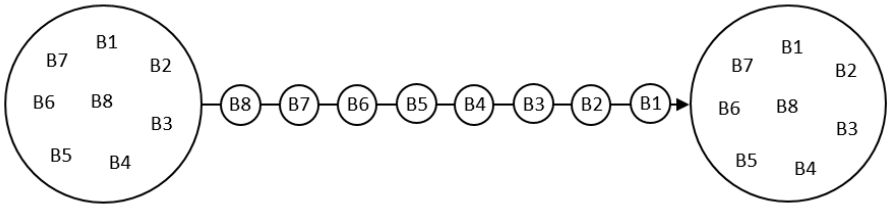


Figura 3.2: Comunicação em Série

Do ponto de vista das máquinas de pagamento automático, o melhor método de comunicação é a comunicação série, pois permite transmitir dados de forma consistente, segura e simples entre uma vasta gama de dispositivos tais como aceitadores de moedas, validadores de notas, entre outros.

3.2 Definição do ccTalk

O ccTalk é um protocolo de comunicação série utilizado para controlo de dinheiro e opera em redes de baixa e média velocidade. Este protocolo de comunicação foi idealizado pela empresa *Coin Controls*, no entanto é um protocolo que está aberto a todos os utilizadores, sendo possível adaptar o protocolo de modo a ser mais conveniente para a solução a implementar. A sua interface está representada na Figura 3.3.

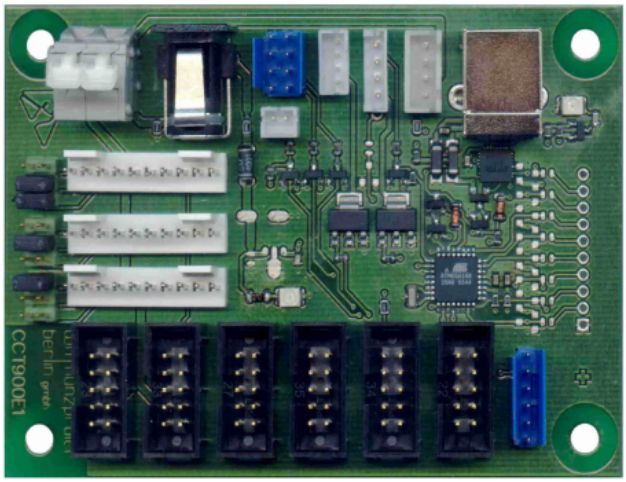


Figura 3.3: Interface do ccTalk [29]

Este protocolo foi desenvolvido para permitir a comunicação entre o controlador e os periféricos com diferentes endereços, tais como os aceitadores e dispensadores de moedas, leitores de cartões magnéticos, entre outros. Sendo um protocolo aberto, existem no mercado várias soluções que utilizam este método para o controlo de dinheiro. É um protocolo fácil de implementar e não necessita de circuitos integrados específicos.

O protocolo ccTalk foi desenvolvido a partir do protocolo RS-232 [13] que é uma solução mais simples e antiga e que foi utilizada para o controlo de moedas durante vários anos; esta solução é também um protocolo de comunicação em série que troca dados binários entre um *Data Terminal equipment* e um *Data Communication equipment*. Deste modo, o ccTalk é uma evolução do RS-232, trazendo como vantagem a facilidade de se interligar com outros sistemas de transferência de dados, tais como Bluetooth, USB, *Ethernet*, etc. O facto de o ccTalk ter sido desenvolvido com base no RS-232 permitiu que o ccTalk fosse concebido a fim de ser utilizado para as máquinas de pagamento automático, no entanto é possível expandir-se para outras tecnologias, tornando-se num protocolo bastante simples, seguro e com a possibilidade de ligar vários periféricos.

Uma das grandes mudanças em relação ao protocolo RS-232, é que o ccTalk possibilita que as ligações entre periféricos sejam feitas através de ligações USB, permitindo assim maior velocidade e segurança através de um Barramento multidrop USB, podendo este protocolo ser chamado do ccTalk *over* USB. A comunicação é assegurada através de portas *Virtual Communication Port - Porta Série* (VCOM) em vez de portas físicas *Communication Port - Porta Série* (COM) , deste modo é possível que os periféricos estejam fisicamente isolados através de *hubs* e seja feita uma transmissão de dados mais segura. Assim, cada periférico tem uma porta VCOM exclusiva e os endereços de destino do ccTalk são redundantes.

O ccTalk não é um protocolo *multi-master*, deste modo não é possível que haja mais do que um dispositivo com a função de master; assim, só um periférico é que pode iniciar a transferência de dados. Como o ccTalk não é um protocolo *multi-master*, também não é possível os *slaves* comunicarem entre si.

Do ponto de vista desta indústria, isto é considerado como vantagem pois o *Host* tem o controlo total sobre a transferência de dados e é iniciado no *master/host* e, deste modo, é mantida a segurança, pois um terminal externo de dados não é capaz de aceder à rede.

A estrutura das mensagens processadas pelo ccTalk tem o formato de *bytes* em vez de *bits* e, embora este modo de estrutura utilize mais memória, torna-se mais fácil de implementar e de ser aplicado nos micro-controladores recentes. O endereçamento lógico do ccTalk permite que estejam ligados até 254 dispositivos *slaves* a um único controlador *host*, todavia os endereços dos *slaves* não determinam necessariamente o tipo de dispositivo.

O protocolo ccTalk foi inicialmente implementado para operar em microcontroladores de 8 *bits* de baixo custo e recorriam a quantidades limitadas de *Random Access Memory* ou Memória de Acesso Aleatório (RAM) e *Read-Only Memory* ou Memória só de Leitura (ROM), contudo, nos dias de hoje, é possível ser implementado noutros processadores mais avançados. Os comandos básicos do ccTalk conseguem ser executados sem grande

utilização de memória, no entanto, recursos mais avançados como criptografia e programação em *flash* necessitam de utilizar mais memória RAM.

O ccTalk pode ser descrito num microcontrolador do 8 *bits* da seguinte forma:

- 1 *Kbytes* até 3 *Kbytes* de ROM
- 30 *bytes* até 200 *bytes* de RAM
- Uma *Universal asynchronous receiver/transmitter* (UART) - embora possa ser feito em *software* sujeito a algumas restrições de tempo
- Um temporizador de 16 *bits*

O protocolo de comunicação em série pode ser controlado por interruptores ou por votação, embora nas implementações mais recentes seja utilizado nos processadores o método de interruptores, pois apresenta mais segurança e confiança. Com uma frequência de 9600 *baud*, cada *byte* recebido ou transmitido demora cerca de 1,04ms. Portanto, uma típica funcionalidade num micro-controlador pode ser realizada a cada 1ms numa porta série, garantido que nenhum dado seja perdido e apresenta a velocidade ideal de modo a garantir que o objetivo principal do protocolo não seja comprometido.

3.3 Taxa de Transmissão do ccTalk

O tempo utilizado para taxa de transmissão de *bits* em série no protocolo de comunicação ccTalk é um fator a ter em conta, sendo que se manteve a ideologia utilizada pelo protocolo RS-232 para comunicação assíncrona para taxa de dados baixa. O RS-232 tem vários parâmetros e é necessário 10 *bits* por cada *byte* transmitido e são configurados da seguinte maneira:

Tabela 3.1: Parâmetros para a taxa de transmissão dos protocolos ccTalk

	Parâmetros
RS-232	9600 Baud Rate 1 Start Bit 8 DataBits 1 Stop Bit Não tem Bit de Paridade

No ccTalk e no RS-232 não é utilizado o Bit de Paridade [40], no entanto, a deteção de erros é verificada através de soma de pacotes.

Ao contrário do RS-232, os sinais de Handshaking [13] não são suportados no protocolo ccTalk, uma vez que o ccTalk é um protocolo utilizado para o controlo de pequenos pacotes de dados e é improvável que ocorram problemas devido ao excesso de dados. [13]

Taxas de transmissão suportadas pelo protocolo ccTalk:

- 9600 *baud* - Um *byte* necessita de 1,042ms para ser transmitido.
- 4800 *baud* - Um *byte* necessita de 2,086ms para ser transmitido. Esta é um opção de baixa velocidade.

Numa fase inicial, a taxa de transmissão eleita foi de 9600 *baud*,pois apresentava melhor relação entre custo e velocidade. Enquanto isso, uma *baud rate* de 4800 ficou destinada para periféricos de baixa potência.

Atualmente, a comunicação entre os periféricos é feita através de USB, *"ccTalk over USB"*. Para a maioria das mensagens de controlo, a velocidade adicional não faz diferença, mas há um benefício distinto ao programar grandes memórias *flash* o ccTalk, tratando-se de períodos de vários minutos que podem ser reduzidos a vários segundos. A taxa de transmissão padrão do protocolo ccTalk é de 9600 *baud*, sendo ainda utilizada por uma vasta gama de periféricos. Caso a *baud rate* não seja definida em lugar algum, então é assumido automaticamente como 9600 *baud*.

De certo modo, o ccTalk opera de maneira independente da taxa de transmissão e a taxa de transferência pode ser aumentada à medida que a tecnologia da camada física evolui para um *hardware* mais económico.

3.4 Barramento de Dados do ccTalk - Porta Série

Tanto as mensagens transmitidas como as mensagens recebidas são feitas através de um único *Barramento* bi-direcional.

A interface do ccTalk por norma utiliza um transístor NPN de coletor aberto [34] na linha de dados e também recorre a uma resistência *Pull Up* na extremidade do *host* da linha de dados. O valor da resistência depende da capacidade de dissipação de corrente dos dispositivos de comunicação, do grau de imunidade de ruído e depende também do número máximo de periféricos que podem ser conectados ao Barramento de dados.

Quanto menor for o valor da corrente, melhor será a imunidade ao ruído.

Para o controlo de uma rede de baixa velocidade, não há requisitos de triagem, pois trata-se de ligações com curtas distâncias entre dispositivos, sendo esta solução um sistema não balanceado.

Caso seja necessário implementar o ccTalk numa rede que apresente longas distâncias entre periféricos, é recomendado o emprego de *drivers* do tipo RS-485, em vez de utilizar um transístor NPN de coletor aberto na linha de dados e uma resistência *Pull Up* na extremidade do *host*. O RS-485 é uma interface de transmissão balanceada, é utilizado para comunicar numa rede em que os dispositivos apresentam longas distâncias entre eles e, por consequência, são ambientes com mais ruído. Neste sistema, é utilizado um cabo extra para transmitir dados em série e necessita de um sinal para controlar o acesso ao Barramento *multi-point*. Esse sinal normalmente é um *handshake* e permite alterar o estado de direção ao enviar *bytes*.

A interface do ccTalk, ao utilizar um transístor NPN de coletor abert torna-se muito mais simples de implementar, comparando com a interface RS-485, embora o ccTalk seja menos robusto no que toca à comunicação de longas distâncias entre dispositivos.

É possível observar na Tabela 3.2 uma comparação entre as interfaces dos protocolos de comunicação ccTalk, RS-232 e RS-485.

Tabela 3.2: Comparação de três interfaces a nível lógico para a comunicação

Interface	ccTalk	RS-232	RS-485
Tipo	Não Balanceada *Multi-drop*	Ponto-a-Ponto	Balanceada *Multi-drop*
Cabos de dados	1	2	2
Controlo de direção	Não	Não	Sim
Número máximo de periféricos	16	1	32
Distancia máxima entre dispositivos	15m	15m	1200m
Velocidade máxima de transmissão	19.2Kbps	19.2Kbps	10Mbps
Estado *Idle*	+5V	-5V até -15V	+1.5V até +5V
Estado Ativo	0V	+5v até +15V	+1.5V to +5V

Observando a Tabela 3.2 e comparando as interfaces dos protocolos de comunicação ccTalk, RS-232 e RS-485, é possível analisar que a interface ccTalk apresenta valores semelhantes às interfaces RS-232 e RS-485, pelo que é possível concluir que a interface ccTalk torna-se num aperfeiçoamento do RS-232 e RS-485.

Tem-se então que o ccTalk, tal como o RS-232, são protocolos de comunicação para curta distância, no entanto, o ccTalk consegue fazer comunicação *multi-drop* através de um único cabo de ligação - Barramento de dado bi-direcional, alcança um maior número de periféricos. Já a interface RS-485, embora seja semelhante ao ccTalk e ao RS-232, é a interface mais indicada para comunicações a longa distância.

3.5 Cabos de comunicação que possam ser utilizados no ccTalk

O tipo de cabo não interfere na compatibilidade com o ccTalk, no entanto se for seguido um padrão pode ajudar a reduzir os problemas no que toca ao encaminhamento de dados e comunicação entre dispositivos.

Há diversos tipos de cabos possíveis para diferentes situações, sendo a escolha influenciada pelas especificações do produto, tal como requisitos de energia, custo, robustez, entre outros. De modo a facilitar este processo, o tipo de cabo fica definido logo desde a fábrica consoante o tipo de dispositivo.

3.5.1 Barramento de dados bi-direcional e *Half-duplex*

De uma forma geral, o protocolo ccTalk faz uso de um *Barramento* de dados bi-direcional e Half-Duplex [39] que pode ser utilizado de várias maneiras; uma maneira simples de implementar o protocolo ccTalk seria o micro-controlador de cada dispositivo utilizar uma única porta I/O bi-direcional capaz de intercalar com uma transmissão e com uma receção. Neste caso, só é efetuada uma nova transmissão quando a receção de outro pacote é terminada, ou seja, tanto o *host* como o *slave* só podem transmitir um pacote de dados, quando a receção de outro pacote estiver terminada, caso esse dispositivo esteja a receber algum.

Caso algum dispositivo seja constituído por um microcontrolador que contenha uma UART integrada, terá os Barramento de dados de transmissão e receção separados, podendo ser combinados na interface eletrónica. Com a utilização de uma UART, o *software* tem que estar preparado para executar um *loop-back* local, isto é, os dados são recebidos e imediatamente transmitidos para a origem sem serem processados. Neste caso, o *software* pode fazer uso de um sinal de ativação ou desativação quando recebe uma mensagem, ou após receber a mensagem de transmissão, ignorá-la.

No protocolo ccTalk, todas as mensagens que são recebidas vão ser processadas, sendo esta uma operação rápida num *slave* e não afeta de forma direta o desempenho nem o objetivo na comunicação.

3.6 Estrutura das mensagens no ccTalk

De uma forma geral, as mensagens encaminhadas através do protocolo ccTalk apresentam uma estrutura padrão do pacote, nomeadamente:

- **Endereço de destino** - O endereço de destino está presente em todos os tipos de pacotes, pois refere-se ao destino final do encaminhamento de um determinado pacote e sem esse parâmetro a mensagem não sabe para onde ir.

  Os endereços podem ocupar desde o *byte* 0 até ao 255, no entanto há endereços exclusivos, detalhadamente:

  - *Byte* 0 - Mensagem de *Broadcast*
  - *Byte* 1 - Endereço do *Master*
  - *Byte* 2 - Endereço do *Slave* para redes sem *Barramento multi-drop*
  - *Byte* 3 até 255 - Está disponível para todos os *slaves* que possuam *Barramento multi-drop*

- **Cabeçalho/Método** - Este campo está presente na estrutura do pacote e foi definido para satisfazer uma vasta gama de atividades no setor financeiro, mais especificamente nos aceitadores de moedas, validadores de notas, *hoppers* e dispensadores de troco. São *bytes* que podem ir deste o 0 até ao 255. Quando o valor do cabeçalho/método está a zero, trata-se de uma mensagem de resposta, por exemplo, *Acknowledgement* (ACK). No ccTalk, está definido que um *slave* não pode receber um cabeçalho nulo vindo do *master*.

  Caso um *slave* não reconheça um cabeçalho/método específico, este dispositivo não retorna qualquer tipo de informação ao *host* e nenhuma ação é executada por parte do *slave*.

- **Endereço de origem** - Como visto anteriormente, o *byte* 1 é utilizado como endereço de origem para o *host* e são poucas as circunstâncias em que este pode ser alterado. Quando um *slave* responde ao *host*, o endereço de origem é definindo no *slave* como 1.

- **Número de *bytes* de dados** - Este campo representa o número de *bytes* de dados da mensagem, e não o número total de *bytes* existentes no pacote, é um requisito obrigatório em todos os tipos de mensagens. É possível variar dentro de um intervalo entre 0 e 255 *bytes*, no entanto, em versões anteriores do ccTalk havia teorias que o número de *bytes* não devia de ultrapassar os 252 pois ajudava no *design* de pequenos dispositivos que faziam uso Microcontroladores PIC e beneficiariam em não ter mais que 252 *bytes* de dados.

  Contudo, nos dias de hoje, os microcontroladores são muito mais poderosos e possuem mais espaço de memória; e como resultado disso, é possível a transferência de um total de 255 *bytes* de dados num único pacote.

  Quando este campo apresenta o valor zero, significa que a mensagem não tem *bytes* de dados e o comprimento total do pacote de mensagens será de 5 *bytes*, tratando-se do valor mínimo de *bytes* permitido. O valor de 255 significa que o tamanho total do pacote será de 260 *bytes*.

- ***Data*** - Esta secção é designada para o conteúdo da mensagem em si, permitindo a existência de vários *bytes* de dados, compreendidos entre 0 e 255 *bytes*. Os códigos permitidos para caracterizar os dados são: Binary Coded Decimal (BCD) e American Standard Code for Information Interchange (ASCII).

[Endereço de Destino][Método][Endereço de Origem][Número Bytes de Dados][Dados][Checksum]

Figura 3.4: Estrutura da mensagem ccTalk

Na Figura 3.4, é apresentado um esquema da estrutura da mensagem, respeitando as definições e requisitos apresentados anteriormente. É de extrema importância que as mensagens respeitem esta estrutura para o bom funcionamento da comunicação.

3.6.1 Verificação de erros nas mensagens do ccTalk

Ambos os métodos apresentados em baixo são utilizados e eficientes para detetar erros. Uma soma de verificação simples de erros de 8 *bits* bits já é suficiente para detetar a maioria dos erros no protocolo de comunicação ccTalk. No entanto, a verificação de erros segundo o CRC de 16 *bits* também é utilizada para conseguir detetar erros como os descritos em cima.

No capítulo seguinte, 3.6.2, irão ser apresentadas as possíveis formas de estruturar as mensagens de dados, no que toca à verificação de erros e criptografia de *bytes*. Para tal, o ccTalk suporta a verificação simples de erros e o Cyclic Redundancy Check (CRC) para tratar da verificação de erros. E quanto à criptografia, é através de Data Encryption Standard (DES), podendo este tipo de linguagem fornecer uma criptografia dupla, especificamente, criptografia a nível de protocolo que envolve todos os pacotes de mensagens e a criptografia a nível de comando, sendo que esta apenas protege os dados que que são determinados por certos comandos do ccTalk.

3.6.1.1 Verificação de Erros Simples

A verificação de erros simples consiste no somatório dos *bytes* da mensagem, de modo que o seu resultado final mais o valor do *checksum*, seja 256. Caso contrário, estamos perante um erro.

Na hipótese de um *slave* receber uma mensagem destinada a outro dispositivo, este deve ignorar o pacote e a soma de verificação de erro.

Este método de verificação de erros é capaz de detetar todos os erros de *bit* único e também é capaz de detetar a maioria dos erros de *bit* duplo e por isso, este método não é recomendado para ser utilizado por canais que apresentem algum ruído, pois há a possibilidade de corromper algum *bit* da mensagem.

Exemplo:

- Mensagem: [1] [0] [2] [0]
- Soma de verificação: [253] pois:

$$1 + 0 + 2 + 0 + 253 = 256$$

3.6.1.2 Verificação de Erros CRC

A verificação de erros através do método CRC consiste na divisão polinomial de 16 *bits* e, como o próprio nome indica, este tipo de verificação utiliza códigos de redundância cíclica, mais concretamente o CRC-16 e o CRC-CCITT. Este processo é baseado na divisão polinomial, isto porque a divisão é muito mais eficiente do que a adição no que toca à deteção de erros. A capacidade de processamento necessária para cálculos de CRC é maior face às somas de verificação de adição, no entanto, é necessário mais código.

A verificação de erros segundo o método CRC traduz-se na verificação de erros CRC de 16 *bits* baseada num registo CRC com valor inicial de 0x0000 e no polinómio

$$x^{16} + x^{12} + x^5 + 1$$

Este método apresenta as suas vantagens, nomeadamente, detetar todos os erros de *bit* único e duplo e também é capaz de detetar todos os números ímpares de *bits* de erros.

3.6.2 Diferentes tipos de pacotes no ccTalk

1. Pacotes de mensagens padrão com verificação de erros simples

   - **Mensagem com N dados de *bytes*** - Neste tipo de mensagens o pacote está estruturado da forma como se pode ver na tabela 3.3. Quando inicializada a comunicação com o objetivo de uma mensagem deste tipo, são gerados dois pacotes com esta estrutura, um dos pacotes é encaminhado para o *master* e o outro para o *slave*; após este processo, o *master* recebe uma resposta enviada pelo *slave*. Esta resposta vinda do *slave* pode ser uma mensagem de confirmação (ACK), uma mensagem de fluxo de dados, ou outro tipo de resposta. Esta sequência não depende da direção de encaminhamento do pacote, consequentemente, esta não se altera consoante a direção.

Tabela 3.3: Sequência do pacote de uma mensagem simples

Sequência - Mensagem simples
Endereço de destino
Número de *bytes* de dados
Endereço de origem
Cabeçalho
Data 1
Data 2
...
Data N
Verificação de erros

   - **Mensagem sem dados de *bytes*** - Caso se pretenda uma mensagem sem *bytes* de dados, a estrutura mantêm-se igual à tabela 3.3, sendo que fica a zero o

campo de número de *bytes* de dados e também não há "[Data 1], [Data 2], ... , [Data N]".

- **Mensagem de confirmação (ACK)** - Este é um tipo de mensagem de confirmação e é um pacote ACK que não requer criptografia. Como já foi mencionado na secção 3.6, caso o cabeçalho do pacote esteja com valor nulo, tratando-se de um pacote de resposta, e não contenha nenhum *byte* de dados, estamos perante a uma mensagem ACK por parte do *slave*.

  Uma mensagem de *acknowledge* tem a seguinte estrutura:

  Tabela 3.4: Sequência do pacote de uma mensagem *acknowledge*

Mensagem *acknowledge*
Endereço de destino
[0]
Endereço de origem
[0]
Verificação de erros

  – **Mensagem Not Acknowledged (NAK)** - Estamos perante um tipo de confirmação negativa ou uma mensagem de não reconhecimento. Uma mensagem NAK é enviada pelo dispositivo de destino, neste caso, por um *slave*. Quando o *host* recebe este tipo de mensagem, volta a enviar o mesmo pacote até que o *slave* seja capaz de responder com uma mensagem ACK; em caso extremos de falhas de comunicação, a transmissão desse pacote é interrompida por completo.
  No caso do ccTalk, este tipo de mensagem pode ser útil quando, por exemplo, o *slave* falha a execução de um determinado pedido, mesmo que este tenha recebido o pacote do *host* sem falhas. Por exemplo, também há comandos de dispensa de dinheiro que utilizam a mensagem NAK para indicar que um pagamento falhou.
  No entanto, as mensagens NAK podem não ser a melhor solução, pois sendo o ccTalk uma rede *multi-drop*, há a possibilidade de responder a *host* inválido ou mal endereçado e isto pode prejudicar a largura de banda.
  Face a isto, o protocolo ccTalk opta por adicionar a informação no cabeçalho da mensagem NAK relativamente ao sucesso do comando pedido, deste modo, comparando com a mensagem NAK genérica, fica mais específica a ação executada e torna-se mais útil para um controlador *host*.
  A tabela 3.5 permite-nos observar um exemplo de uma mensagem NAK em que se pode verificar que o cabeçalho é diferente de zero. O cabeçalho com valor 5, no ccTalk, representa "despensa de moedas do *hopper*", sendo este tipo uma resposta NAK clássica.

Tabela 3.5: Sequência do pacote de uma mensagem *not acknowledge*

Mensagem *not acknowledge*
Endereço de destino
[0]
Endereço de origem
[5]
Verificação de erros

- **Mensagem de ocupado** - Como o próprio nome indica, é uma mensagem que informa que aquele periférico está ocupado, ou seja, é uma mensagem que informa o *host* que o *slave* está ocupado e que não está capaz de responder. O *host* está preparado para enviar novamente o pacote após um intervalo de tempo pré-definido, até que o *slave* esteja livre e capaz de responder.

2. Pacotes de mensagens padrão com verificação de erros a partir CRC

- **Mensagem com N dados de *bytes*** - Este tipo de pacotes é idêntico ao modelo anterior com a exceção que a verificação de erros é realizada através de somas de verificação CRC utilizando 16 *bits*. Quanto à estrutura do pacote, há alterações, nomeadamente o campo do endereço de origem é substituído pela parte interior da soma de verificação (CRC-16 LSB) e, no final, está o campo responsável por finalizar a verificação (CRC-16 MSB), como se pode confirmar na tabela 3.6. Neste caso, todos os dispositivos *slaves* respondem de forma automática ao endereço de origem, que está implícito como endereço padrão do *host*.Quando a criptografia é utilizada no nível do protocolo, o endereço do *host* é sempre igual a 1.

Tabela 3.6: Sequência do pacote de uma mensagem simples com verificação CRC

Sequência - Mensagem simples com CRC
Endereço de destino
Número de *bytes* de dados
CRC-16 LSB
Cabeçalho
Data 1
Data 2
...
Data N
CRC-16 MSB

3. Pacote de mensagem encriptada e com verificação de erros CRC

- **Mensagem com N dados encriptados** - Neste tipo de pacote os dados estão encriptados e a criptografia é utilizada na parte superior da soma de verificação de CRC, ou seja, todos os *bytes* de dados são encriptados logo no primeiro *byte* de soma da verificação CRC. Neste caso, o campo do endereço de destino e o valor que corresponde ao número de dados a ser transmitido permanecem igual na estrutura, tal como se pode ser na tabela 3.7, de modo a tornar o encaminhamento seguro e garantir que não existirão confusões de destinos quando o pacote está no *Barramento*. Assim, a mensagem encriptada será ignorada por todos os periféricos que não correspondam ao endereço destino.

Tabela 3.7: Sequência do pacote de uma mensagem encriptada com verificação CRC

Sequência - Mensagem encriptada
Endereço de destino
Número de *bytes* de dados
Byte encriptado 1
Byte encriptado 2
...
Byte encriptado N

3.6.3 Mensagem de *Broadcast* no ccTalk

A mensagem de *broadcast* pode ter diversos propósitos, no entanto, o principal objetivo é enviar uma mensagem para todos os dispositivos que se encontrem ligados à rede mas também pode ser para quando o *master* necessita de enviar uma mensagem a um determinado dispositivo mas o *master* não sabe qual o endereço de destino, entre outros.

No caso do ccTalk, o endereço de destino reservado para mensagens de *broadcast* é o zero. Assim, quando o *master* pretende encaminhar um pacote do qual não sabe o endereço de destino, envia o pacote e no campo de endereço de destino coloca zero e a mensagem é encaminhada para todos os periféricos presentes na rede. Por consequência, todos os periféricos vão responder ao *master* relativamente a este pacote, pelo que pode gerar colisões. Este tipo de mensagens só pode ocorrer em redes *multi-drop*, pois, assim que ocorrem colisões, o *software* está preparado para ignorar algumas atividades que possam acontecer no *Barramento* num determinado tempo fixo. Portanto, as mensagens de *broadcast* são utilizadas maioritariamente para garantir uma correspondência de endereço num *Barramento* de dados.

No entanto, a utilização das mensagens de *broadcast* podem trazer alguns problemas, nomeadamente quando o *master* envia uma mensagem encriptada para todos os dispositivos da rede, pois é possível que alguns periféricos não estejam capazes de descodificar esse tipo de mensagens.

No ccTalk, o pacote gerado pelo *master* não contém qualquer tipo de informação relativamente ao pacote estar encriptado ou não, por isso, quando uma mensagem encriptada é enviada para o endereço de *broadcast*, só os periféricos habilitados a descodificar a mensagem, segundo as suas próprias regras de descodificação, é que vão conseguir responder com mensagem de reconhecimento.

Uma mensagem codificada enviada pelo *master* destinada a todos os periféricos, *broadcast*, tem uma chave única de codificação que pode não funcionar em todos os dispositivos ligados à rede. Há três tipos de impasses que poderão ocorrer ao enviar uma mensagem encriptada para todos os dispositivos:

1. **Periférico capaz de descodificar mensagens encriptadas** - Ocorre o processo dito normal, o *master* envia em mensagem em modo *broadcast* e os *slaves* que correspondam à chave de codificação descodificam a mensagem e respondem com uma mensagem de confirmação.

2. **Periférico incapaz de descodificar mensagens encriptadas** - Neste caso, a chave de codificação do pacote não vai corresponder a nenhum *slave* e, quando se proceder à soma de verificação de erros CRC, todos os periféricos que não correspondam a essa chave vão falhar e não respondem com uma mensagem de confirmação ao *master*.

3. **Periférico capaz de descodificar mensagens encriptadas mas que recebe uma mensagem simples** - Neste cenário, o *slave*, ao receber uma mensagem simples, vai descodificar a mensagem, mesmo não sendo necessário esta ser descodificada, podendo resultar disso, erros na soma de verificação de erros CRC.

Para evitar situações deste tipo, é de se evitar o uso do endereço de *broadcast* nas situações descritas acima. No entanto, uma boa solução para combater estes problemas é possuir periféricos com endereços predefinidos.

3.6.4 Erro na comunicação do ccTalk

O *host* ao transmitir um pacote de dados para um *slave*, este pacote normalmente vem acompanhado de um pedido para o *slave* executar alguma operação. No entanto essa operação pode ser impossível de ser executada, pois pode ter ocorrido um erro, detetado através da soma de verificação de erros, ou faltar dados para concretizar um determinado pedido. Quando estes fenómenos ocorrem, o *slave* e o *host* têm o seguinte comportamento:

- *Slave* - Não efetua qualquer tipo de ação e o seu *buffer* de entrada é limpo

- *Host* - Este, como não recebe nenhuma resposta vinda do *slave*, tem a possibilidade de reenviar o pacote consoante as configurações definidas no *host*, nomeadamente, tempos fixos ou aleatórios de espera da resposta.

3.6.5 Camadas do protocolo ccTalk

Na Figura 3.5, é possível ver como as diferentes camadas do protocolo são implementadas no ccTalk no que toca a enviar mensagens de dados.

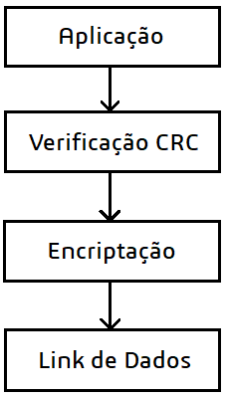


Figura 3.5: Camadas do protocolo ccTalk

Tomando o exemplo de enviar uma mensagem para um recetor de moedas, no *master* é gerado um pacote, que consiste no endereço de destino, o número de *bytes* que estão para ser enviados, o cabeçalho e os *bytes* de dados.

Uma soma de verificação de erros CRC de 16 *bits* é calculada e adicionada à estrutura do pacote.

Posteriormente, a mensagem é encriptada e verificada pela a UART, que é responsável por adicionar os *bits* de *start* e *stop* de transmissão. A mensagem é recebida pelo periférico, neste caso o recetor de moedas, e executa a mesma operação no sentido contrário de modo a confirmar ao *master* a receção da mensagem.

3.7 Intervalos de tempo entre diferentes operações

Nesta secção serão apresentados os intervalos de tempo necessários para executar uma nova operação, nomeadamente entre *bytes* e também entre novos comandos e a resposta a esse pedido. Embora não haja regras ou normas, há certos parâmetros a serem cumpridos.

3.7.1 Intervalo de tempo entre *bytes*

O ccTalk está preparado para, assim que um dispositivo recebe um pacote com *bytes* de dados, esse dispositivo ter que aguardar durante 50ms para estar apto a receber outro pacote com *bytes* de dados. É uma característica fundamental do ccTalk e deve ser cumprida.

O *software* de comunicação série do ccTalk, ao transmitir um pacote de dados, deve garantir o menor atraso possível, respeitando este intervalo de tempo entre *bytes* e também

garantir que os dados em série, que não estão para ser processados, não fiquem no *buffer* do recetor.

Como vimos anteriormente, no capítulo da taxa de transmissão suportada pelo ccTalk, 3.3, com uma taxa de transmissão de 9600 *baud*, cada *byte* necessita de 1.04 ms para ser transmitido e, portanto, o intervalo de tempo entre *bytes* deve ser menor que 2 ms para maximizar a largura de banda no canal de dados. Esta característica tem que ser tomada em consideração, pois num canal de dados *multi-drop*, onde vários dispositivos diferentes estão conectados ao mesmo *Barramento* de dados, as mensagens nestes dispositivos necessitam de ser processadas dentro do seu tempo pré-definido.

3.7.2 Intervalo de tempo entre um novo comando e a resposta

O intervalo de tempo entre a emissão de um comando e a sua resposta não tem um valor fixo, é decidido pelo próprio comando e pela aplicação que desempenha essa ação. Há alguns comandos que retornam dados imediatamente após terem recebido o pedido, normalmente demora 10 ms, enquanto outros comandos aguardam que alguma operação seja executada, como por exemplo, uma resposta ACK.

O *host* deve estar preparado para ter em consideração estes intervalos de tempo e atrasos de acordo com o pedido enviado e com a resposta recebida. Quanto aos *slaves*, estes devem estar bem descritos, a nível de *software*, para serem o mais rápido possível a responder e minimizar o atraso na rede.

3.8 Limitações dos periféricos *slave* para uso do ccTalk

Os dispositivos *slave* têm que possuir determinadas características no que toca à capacidade de processamento, memória e espaço RAM. Caso o *slave* tenha baixo desempenho nestas características, pode haver algumas restrições, no entanto podem ser reduzidas utilizando algumas particularidades do *slave*.

O Buffer presente na porta de entrada do *slave* é relativamente pequeno, entre 1 e 10 *bytes*, e mensagens demasiado grandes enviadas pelo *host* podem não ser recebidas ou armazenadas. O Buffer do *slave* armazena os dados recebidas até que o Buffer de entrada se encontre cheio.

Mesmo havendo suporte no protocolo ccTalk para mensagens de tamanho variável, o *slave* pode não possuir características para lidar com mensagens desta dimensão, assim o *slave* admite que o pacote proveniente do *host* possui um cabeçalho específico com um certo número de *bytes* de dados.

Capítulo 4

Solução Proposta

Após analisar as soluções existentes no mercado, no capítulo 2, quais os dispositivos ou periféricos utilizados no sistema, na secção 2.1 e conhecido o protocolo de comunicação ccTalk apresentado no capítulo 3, procedeu-se ao desenvolvimento do controlador propriamente dito.

A solução proposta passou por determinar os componentes necessários para o controlador, definir a arquitetura do circuito elétrico e por fim desenvolver cenários reais em ambiente fictício com que o *Money Link* terá que saber lidar.

4.1 Definição dos Componentes para o Controlador *Money Link*

Como foi referido no capítulo 2, o *Paylink* é o controlador de numerário utilizado nas portagens de Portugal, acontece que, como é um modelo limitado a A-to-Be, teve a necessidade de agregar ao *Paylink* mais um equipamento de modo a expandir algumas funcionalidades, como por exemplo aumentar o número de portas disponíveis. Este equipamento denomina-se de *board* 12449.

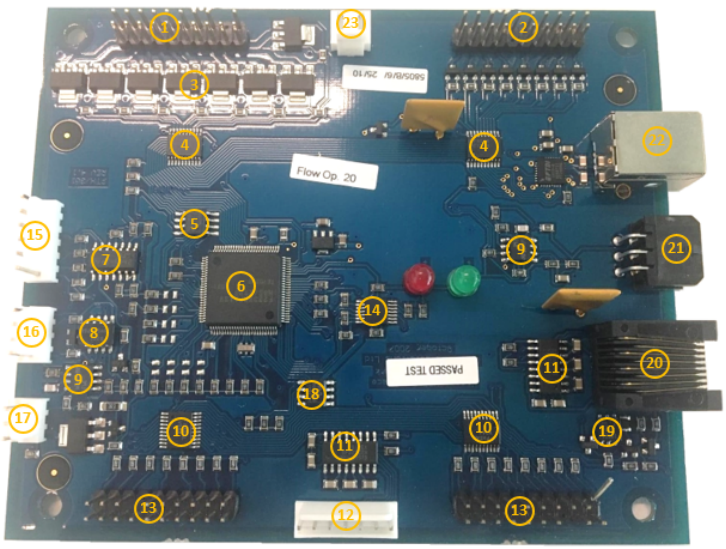
O KS1000 é igualmente um controlador utilizado nas portagens em operação nos Estados Unidos da América e também um controlador direcionado à vertente numerária mas com mais funcionalidades e com maior capacidade.

O *Money Link* será uma otimização destes três dispositivos, pelo que faz todo o sentido comparar todos os equipamentos e só posteriormente fazer um levantamento dos componentes para o *Paylink*. Na Tabela 4.1 pode-se verificar os três modelos e os respetivos componentes.

Nas Figuras 4.1 e 4.2, é possível ver com mais detalhe os circuitos elétricos dos modelos em estudo, *Paylink* e KS1000, respetivamente.

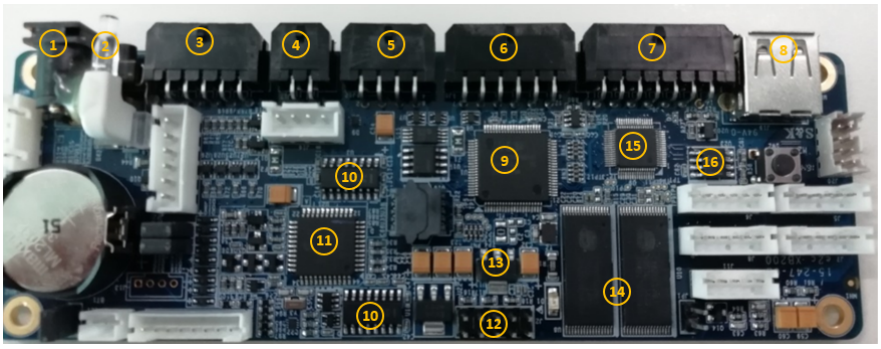
Tabela 4.1: Comparação dos componentes dos três equipamentos em analise, *Paylink*, *board* 12449 e KS1000

Paylink	**KS1000**	***Board* 12449**
Alimentação: 12V DC	Alimentação: 9V a 36V	Atuador para solenoide
16 Portas de entrada	PC	12 Portas de entrada
8 Portas de saída de alta potência, cada uma com 2.5A no máximo	6 Portas para comunicação série	6 Portas de saída de alta potência
8 Portas de saída de baixa potência	2 Portas para comunicação ccTalk	4 Portas de saída de baixa potência
1 Porta para comunicação MDB Master	4 Portas de entrada	Uma carta de expansão para comunicar através de ccTalk
1 Porta para comunicação MDB Slave	4 Portas de saída de potência	
1 Porta Serial meter	7 Interruptores	
1 Porta RS232 para comunicar com a impressora	1 Porta USB 2.0	
1 Porta para comunicação ARDAC5	2 Portas de *ethernet*	
1 Porta para comunicação ccTalk		
1 Porta para comunicação ID003		
1 Porta USB		
Dimensões: 142x162x25.5 (mm)	Dimensões: 280x130x60 (mm)	Dimensões: 122x101.9 (mm)



1 – 8 Saídas de alta potência
2 – 8 Saídas de baixa potência
3 – Transístores NPN 1A 60V
4 – Flip-Flop *Tristate* – Saída para 15 outputs
5 – Microcontrolador a operar com o *Serial Meter*
6 – Processador da *Board*
7 – 4 Comparadores
8 – *Optocopler*
9 – 2 Compradores
10 – Transístor PNP 3A 60V
11 – *Driver* RS-232
12 – Porta para comunicar através do RS-232
13 – *Switches*
14 – NAND Gate
15 – Porta *Serial Meter*
16 – Porta MDB *Master*
17 – Porta MDB *Slave*
18 – Processador EEPROM de 64Kbit I2C
19 – Silicon *Serial Number*
20 – Porta para comunicar através do ARDAC5
21 – Porta para comunicar através do ccTalk
22 – Porta USB
23 – Alimentação DC a 12V

Figura 4.1: Circuito do *Paylink* com os respetivos componentes constituintes



1 – Alimentação DC entre 9V a 36V
2 – Botão *Reset*
3 – Porta GPIO
4 – Porta SPKR
5 – Porta *Intrusions*
6 – Porta Série USB
7 – Porta MB série
8 – Porta USB
9 – Processador NXP LPC1754
10 – MAX323
11 – Processador PIC18f4611
12 – Transístor FET – 64jk N05a
13 – Transístor FET – 67fx N12a
14 – Memórias SRAM
15 – CPLD - *Complex Programmable Logic Device*
16 – *Voltage* supervisor – *Reset Control*

Figura 4.2: Circuito do KS1000 com os respetivos componentes constituintes

4.1.1 Componentes do *Money Link*

Considerando os controladores mencionados e descritos acima, e tendo em vista a execução de um novo controlador com características semelhantes, optou-se por implementar um sistema totalmente novo com as seguintes especificações:

- Alimentação de 12V a 24V
- Modulo Kinetis A-to-Be
- 1 Porta *Ethernet*
- Processador EEPROM 128k
- 2 Portas para comunicação ccTalk
- 3 Portas para comunicação Série
  - 1 Porta para comunicação série TX/RX
  - 1 Porta para comunicação RS232 sem *Handshake*
  - 1 Porta para comunicação RS232 com *Handshake*
- 1 Porta para comunicação MDB Master
- 1 Porta para comunicação MDB Slave
- 1 Porta para comunicação ID003
- 1 Porta para comunicação I2C
- 1 Porta USB *Host*
- 1 Porta USB Dispositivo
- 16 Portas de entrada
- 8 Portas de saída de potência

- 8 Portas de saída de baixa potência
- 1 *Dip switch* com interruptores
- 2 LEDs de estado - Tri-color

Infelizmente, não foi alcançável o desenvolvimento do desenho do circuito bem como a sua produção física, pois o responsável por efetuar este tipo de tarefas na A-to-Be não obteve disponibilidade no período de execução e, uma vez que não faz parte de uma das minhas áreas, eu teria um papel de observadora e aprendiz. No entanto, não foi totalmente desvantajoso, pois foi encontrada uma solução física já existente para elaborar testes e deste modo concluir com sucesso as funções pretendidas. É possível encontrar na secção seguinte 4.2 uma breve demonstração do sistema implementado.

4.2 Ambiente de Desenvolvimento

No que diz respeito ao ambiente de desenvolvimento, serão apresentados dois cenários.

O primeiro que será num ambiente de desenvolvimento para fase de testes preliminares, ou seja, é a elaboração do projeto através de dispositivos e mecanismos que não os reais. O segundo contexto, e a fase final da dissertação, envolve a utilização do periférico real de modo a testar o código que fora elaborado na fase anterior.

Para tal, procedeu-se ao desenvolvimento prático do sistema, com o objetivo de se produzir todos os cenários de simulação e produzir assim as respostas e reações a esperar do controlador e dos dispositivos.

Como este projeto se encontra numa fase muito embrionária, apenas se habilitou para a utilização de um único periférico, um dispensador de moedas, um dos mais simples, para o controlador, apresentado numa secção mais adiante 4.2.4. Para tal, desenvolveu-se o código num IDE 4.2.1 com base nas regras do ccTalk e do periférico a testar.

Nesta secção, vai ser possível analisar todos os equipamentos, mecanismos e ferramentas utilizadas para o desenvolvimento do projeto.

Em primeiro lugar, será apresentada a versão executada no ambiente para fases de testes iniciais, tal como se pode observar no esquema 4.3, em que tudo começa num computador que, através de uma IDE de desenvolvimento de *software* representam o controlador. O IDE fica responsável por enviar e processar as mensagens enviadas para a *board* através da porta UART, posteriormente a mensagem é encaminhada para o conversor TTL, por sua vez este dispositivo transforma a mensagem de TTL para RS 232 e vice-versa. Também o conversor está ligado ao computador por cabo USB, cuja porta USB está associada a um terminal de leitura, no caso deste projeto, é o *termite*, que serviu como Dispositivo virtual. A mensagem é recebida no *termite* como se fosse um Dispositivo e, de seguida, o utilizador tem que inserir manualmente a resposta hipotética que o Dispositivo responderia, esta mensagem é encaminhada no sentido inverso, ou seja, primeiro pelo

conversor e posteriormente pela *board* FRDM K64F até chegar por fim ao IDE de desenvolvimento; nesta fase, o IDE está capaz de interpretar, verificar o *checksum* e e decidir a próxima função para o Dispositivo.

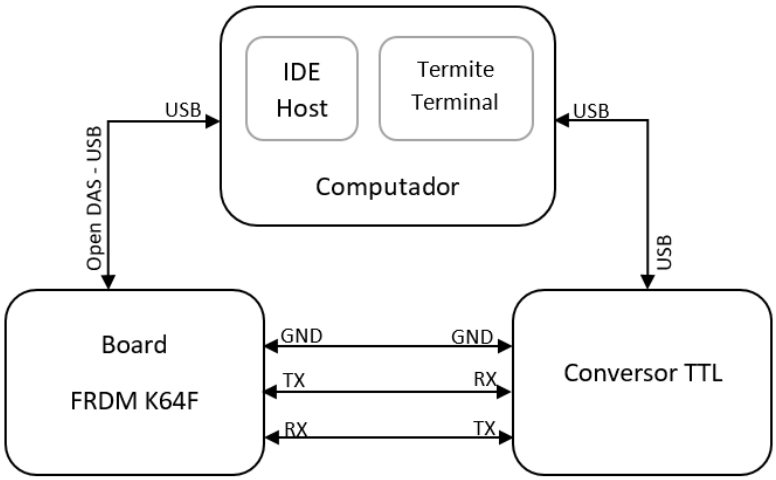


Figura 4.3: Esquema da montagem do ambiente de desenvolvimento para testes iniciais

Na fase final do projeto, o ambiente de desenvolvimento sofre algumas alterações mas apenas na vertente física, o *software* mantém-se o mesmo, deste modo é possível testar no periférico o código desenvolvido inicialmente e analisar todos os cenários em vida real. No esquema seguinte 4.4, observa-se as alterações realizadas para a fase final da dissertação, nomeadamente, surge a interface ccTalk a substituir o conversor TTL e esta liga-se ao Dispositivo, deixando assim de ser necessário recorrer ao *termite*.

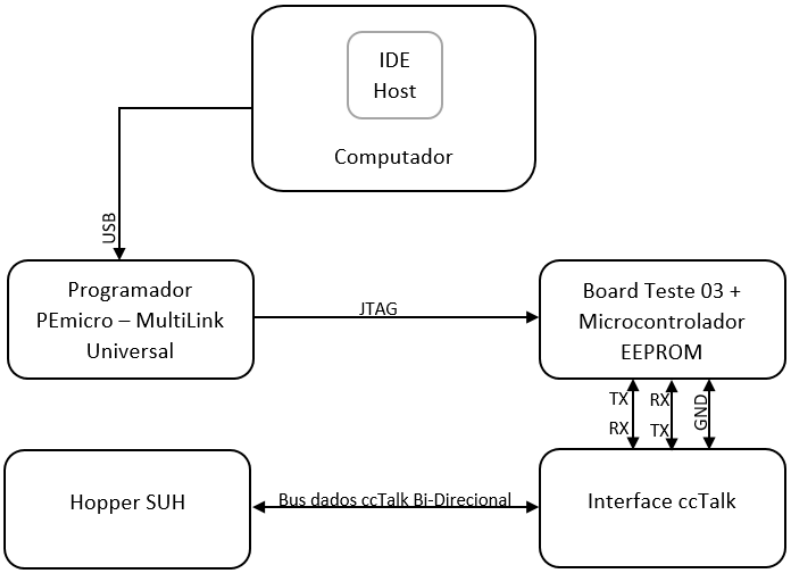


Figura 4.4: Esquema da montagem do ambiente de desenvolvimento de produção

Posto isto, nas secções seguintes serão abordados todos estes elementos físicos e virtuais que foram utilizados ao longo da dissertação, nomeadamente:

- IDE de desenvolvimento MCUXpresso
- *Board Kinetis* FRDM K64F
- Conversor TTL
- Interface ccTalk
- *Serial Universal Hopper*
- Microcontrolador EEPROM e *Board* Teste 03
- Programador PEMicro *MultiLink* Universal

4.2.1 MCUXpresso IDE

O IDE de desenvolvimento de *software* utilizado nesta dissertação foi o MCUXpresso IDE pertencente à entidade NXP *Semiconductors*. Foi o ambiente de desenvolvimento de *software* recomendado pela A-to-Be, pois, para além de ser a ferramenta utilizada por eles e por isso ser mais fácil o esclarecimento de dúvidas que foram ocorrendo ao longo do processo, é também possível encontrar vários fóruns de dúvidas de utilizadores que desenvolvem das mais simples até aos mais complexos sistemas.

O MCUXpresso possui exemplos de código aberto, *middleware* e exemplos de funcionalidades que servem como referência para facilitar no desenvolvimento de *software*.

O facto deste IDE ser desenvolvido pela mesma entidade que desempenhou a *board* semicondutora, FRDM K64F, facilitou a configuração da placa, pois bastou fazer o *download* do *Kit* de desenvolvimento de *software* (SDK) correspondente à placa FRDM K64F, adicionar ao programa MCUXpresso e, posteriormente, o MCUXpresso já estava capacitado para atuar em conjunto com a placa FRDM K64F.

Como já se havia decidido que a porta a utilizar para a comunicação seria a UART3, bastou ir à secção da configuração do *pinout*do processador e identificar a UART3, como se pode verificar na Figura 4.5, bem como definir o *baud rate* a 9600 e definir o modo de interrupções da CPU.

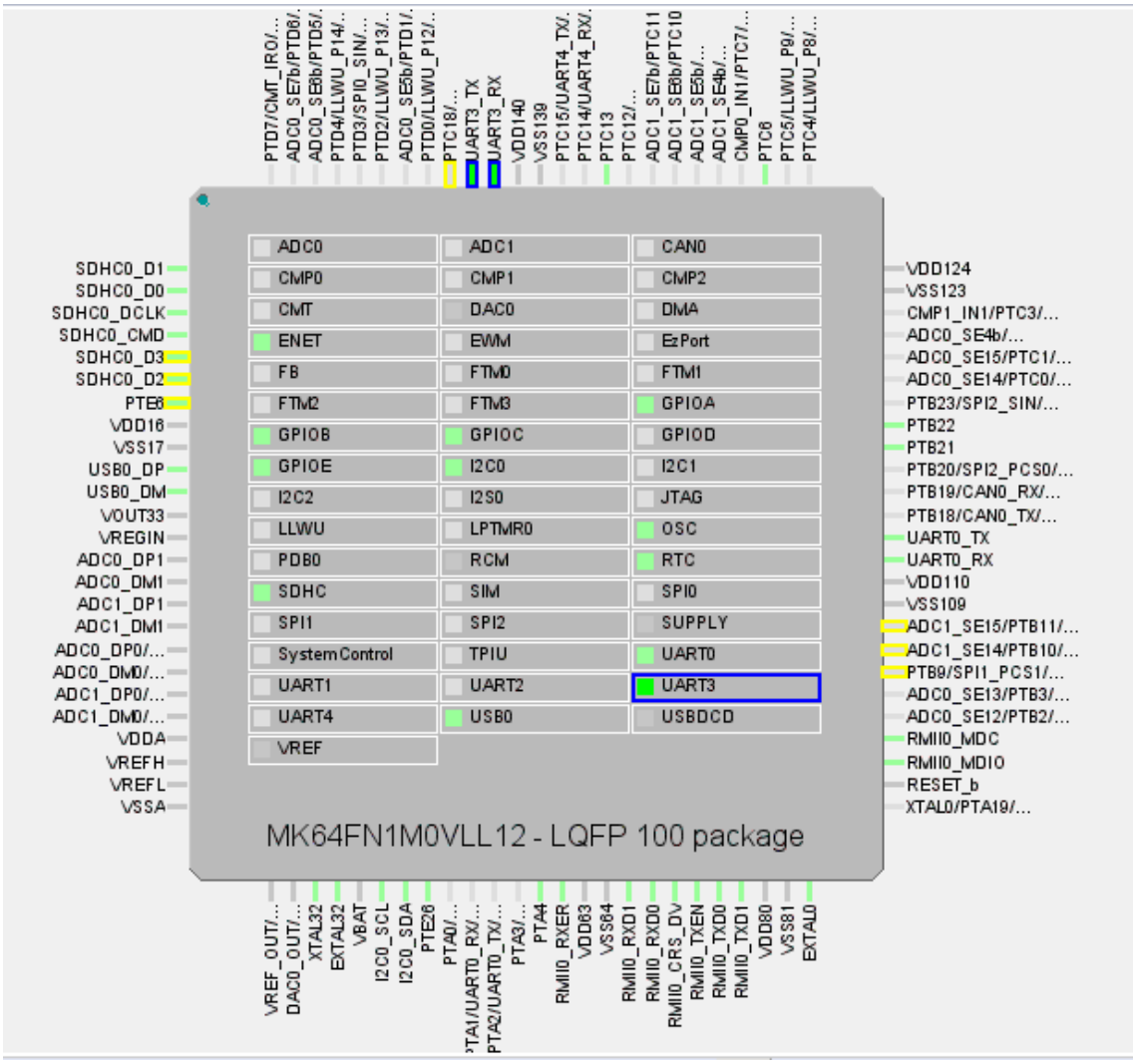


Figura 4.5: Seleção e configuração dos pinos da UART3

Após configurar a *board* através do SDK, o IDE já está preparado para se dar o inicio do desenvolvimento do software e dos testes. Posto isto, iniciou-se a programação do controlador em linguagem C.

Na minha pessoal opinião, é um IDE bastante fácil e prático de utilizar, o facto de ter acesso ao SDK permite uma rápida configuração dos pinos, *baud rate*, definição da suspensão da tarefas da CPU para responder a outras com maior prioridade, o que torna ainda mais prático e fácil de desenvolver código, assim não é necessário configurar o *clock* nem os pinos de forma manual. Também dispõe de vários fóruns de esclarecimento de dúvidas e até o próprio site da NXP possui vários formulários, vídeos exemplificativos, exemplos de pequenos trechos de código mais comuns, explicação detalhada de todas as *boards* e as suas possíveis configurações.

4.2.2 *Board Kinetis FRDM K64F*

A *board* utilizada nesta fase de testes corresponde à FRDM K64F, neste caso serviu como semicondutor para a comunicação e o controlo do periférico, substituindo a versão final do controlador *Money Controls*.

A FRDM-K64F *Freedom Development Platform da Freescale*, representado na Figura 4.6, é um semicondutor também produzido pela NXP e trata-se de uma plataforma de desenvolvimento de custo muito baixo para microcontroladores da *Kinetis*. [33] [38]

A utilização desta *board* foi também recomendada pela A-to-Be. Embora este dispositivo tenha muitas outras funcionalidades, no contexto desta dissertação, apenas se deu importância em escolher uma *board* simples e de baixo nível que fosse capaz de efectuar a comunicação via UART.

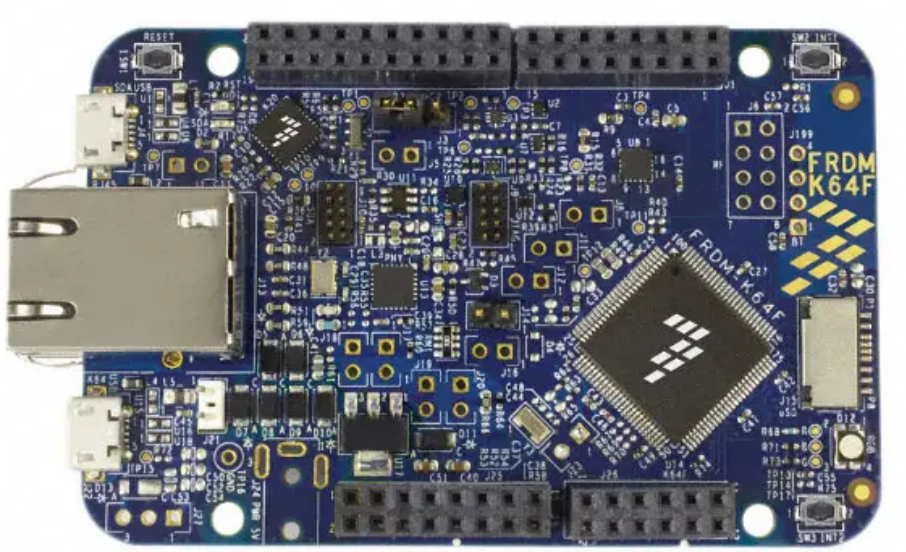


Figura 4.6: Plataforma de desenvolvimento FRDM-K64F *Freedom*

Quanto ao processador embutido na *board*, é um microcontrolador baseado em ARM Cortex®-M4 de 120 MHz com *Floating-Point Unit* (FPU) e faz parte da família de microcontroladores da *Kinetis*. Os produtos da categoria K64 são otimizados para aplicações sensíveis, no entanto, exigem alguma densidade de memória e baixa eficiência no processamento de energia, também fazem uso da ligação USB ou *Ethernet* de baixa energia e até 256 KB de SRAM incorporada, o *design* USB sem cristais permite reduzir o custo do sistema e também o espaço utilizado na placa. Estes dispositivos possuem as capacidades e escalabilidade abrangentes dos microcontroladores da desenvolvidos pela *Kinetis*.

No que diz respeito ao *hardware*, FRDM K64F é um equipamento compatível com os pinos R3 Arduino™, permitindo assim uma vasta gama de opções de *boards* de expansão.

É uma plataforma que aceita uma variedade de periféricos permitindo uma rápida prototipagem. Também possui um acelerómetro digital de 3 eixos e magnetómetro para criar recursos completos eCompass, um Led tricolor e botões de pressão para interação

direta com o utilizador e *feedback*. Há a possibilidade de expandir a memória utilizando um cartão microSD. Em relação às opções de ligação, estas podem ser feitas através da porta *Ethernet* na placa e distribuidores de fios para uso com módulos de montagem adicional com Bluetooth e rádio 2.4GHz. A plataforma de desenvolvimento FRDM-K64F tamém dispõe de uma entrada OpenSDAv2, o adaptador de código aberto, em série e de depuração incorporado no *hardware Freescale* executando um *bootloader* de código aberto. Este circuito oferece muitas opções para a comunicação em série, programação *flash* entre outras funcionalidades.

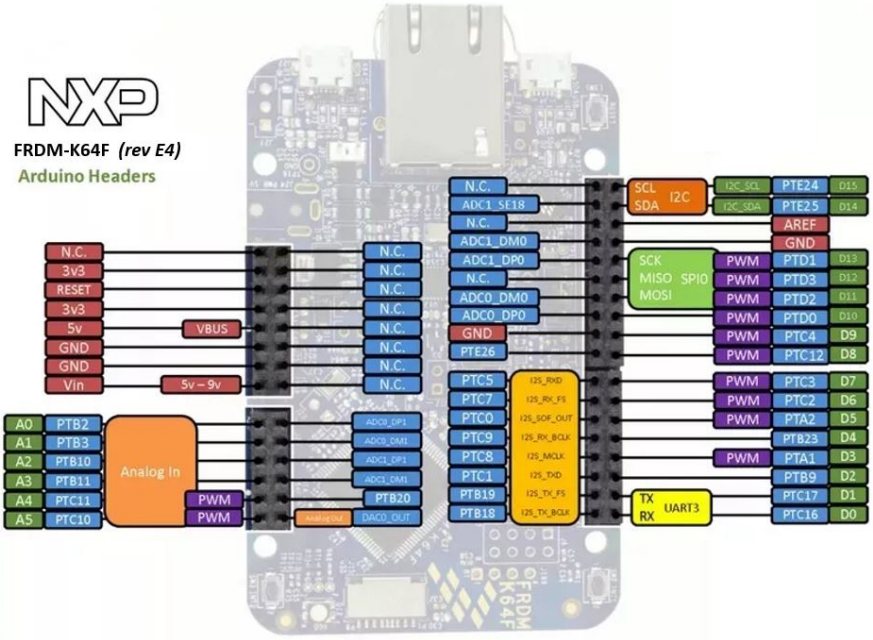


Figura 4.7: *Pinout* FRDM K64F

No contexto desta dissertação, esta placa FRDM K64F serviu apenas como semicondutor para comunicar através do protocolo de comunicação cTalk, como foi visto na secção 3, entre o *host* e o Dispositivo, substituindo o modelo final, o *Money Link*. Visto que esta *board* servirá apenas para transmitir dados, decidiu-se utilizar portas UART que, após analisar várias configurações , permitiu concluir que as portas mais indicadas seriam o Recetor (RX) e Transmissor (TX) pertencentes à UART3, como se pode ver na Figura 4.7. Para implementar este simples sistema, bastou ligar estas 2 portas, PTA16 e PTA17, correspondendo, respetivamente, ao RX e TX, e uma ligação ao *ground*.

Como já foi visto anteriormente, tanto a comunicação RS-232 como o ccTalk necessitam de um conversor para transformer o sinal USB em sinais série, no caso do ccTalk o desenho do circuito, bem como a sua montagem, será apresentado na secção 4.2.3. Por isso, nesta fase de testes preliminaries, utilizou-se um conversor de RS232 para TTL, como se pode ver na Figura 4.8. Este conversor é um dos mais simples que existe no mercado, tornando-se fácil de utilizar para a conversão do sinal. Esta placa tem o único objetivo de transformar RS232 para TTL e de RS232 para TTL, TX e RX. [14]

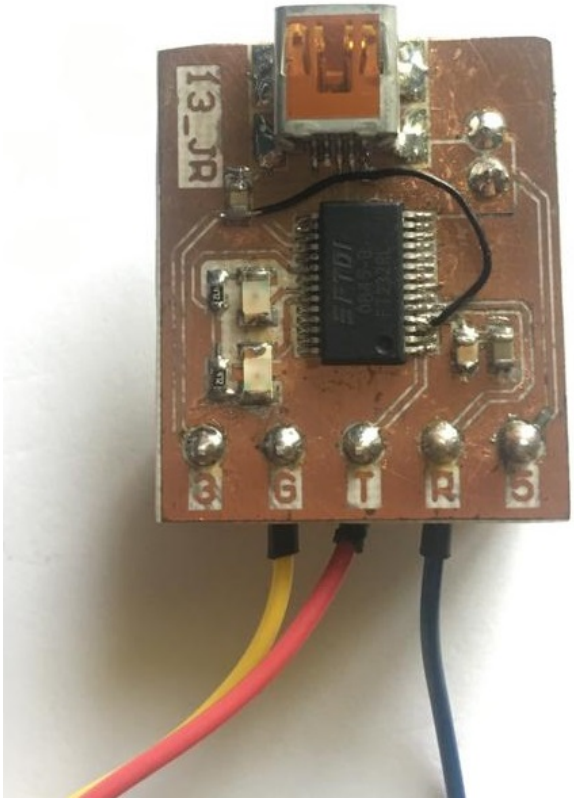


Figura 4.8: Conversor de RS232 para TTL

4.2.3 Interface ccTalk

A interface ccTalk é um dos elementos essenciais para a execução deste projecto, uma vez que é a interface responsável por fazer a ponte entre o controlador *Money Link* e o periférico em questão. Neste caso, a interface foi desenvolvida segundo o circuito apresentado na Figura 4.9, pois é o exemplo *standard* demonstrado nos documentos oficiais do protocolo de comunicação ccTalk. [9] [12]

Quanto aos valores dos níveis lógicos do ccTalk, variam entre 0V a 1V para o estado ativo e entre 3.5V a 5V para o estado inativo, pelo que para os microcontroladores de 5V é possível implementar uma interface de baixo custo. Já foi referido anteriormente que as mensagens transmitidas e recebidas são encaminhadas por um único Barramento de dados bi-direcional, no entanto há outro canal de 0V denominado de COM.

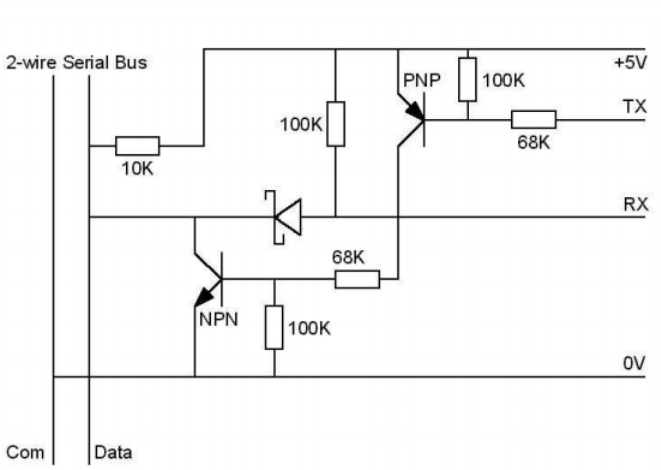


Figura 4.9: Desenho do circuito eléctrico da interface ccTalk [9]

Na Figura 4.9 está representado o desenho do circuito elétrico da interface ccTalk e esta é constituída por dois transístores, um transístor de coletor aberto e outro transístor PNP, e por um díodo. Nas primeiras versões da interface do ccTalk apenas era utilizado um transístor PNP, no entanto não era viável uma vez que a tensão no Barramento de dados em alguns periféricos descia para os 4V e, deste modo, a segurança não era garantida.

Por isso, estes três componentes trabalham em conjunto e também com uma resistência *Pull Up* para garantir a tensão correta e a segurança necessária para a comunicação. O circuito dispõe de um transístor de coletor aberto para accionar o Barramento de dados e um receptor directo protegido por um díodo.

Nesta fase do projeto, surge uma adversidade, pois o ccTalk comunica a 5V e as *boards* de teste comunicam a 3.3V, pelo que foi necessário adaptar o circuito do ccTalk mencionado em cima, na Figura 4.9, para ajustar as tensões, e foi necessário adicionar um conversor de sinal. Nas Figuras 4.10 e 4.11 é possível analisar os circuitos eléctricos da interface ccTalk para enviar mensagens e receber respetivamente, e é possível verificar que as diferenças entre estes circuitos e o circuito recomendado pelo protocolo ocorrem no conversor de sinal posto imediatamente após o sinal TX.

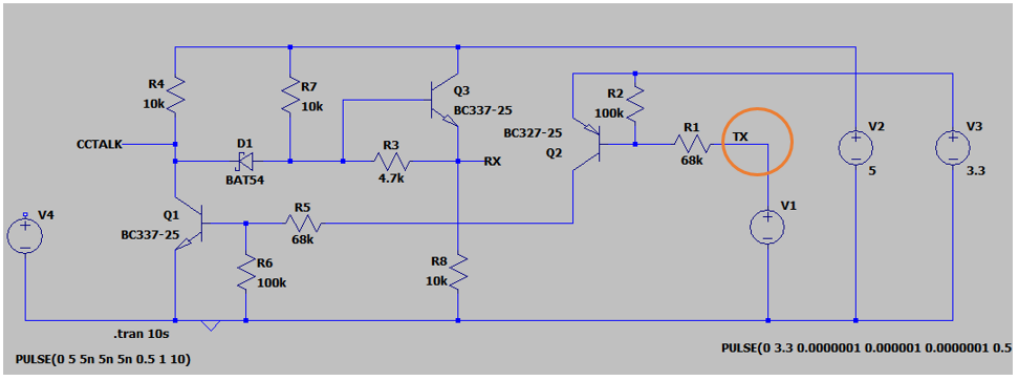


Figura 4.10: Circuito da interface ccTalk quando uma mensagem é enviada

Na Figura 4.10, está representado o desenho do circuito elétrico da interface ccTalk, bem como, do conversor de sinal, sendo que neste primeiro cenário, está representada a comunicação no sentido do *host* para o o Dispositivo, estando a saída designada pelo ccTalk.

No segundo cenário, este para receber a mensagem vinda do Dispositivo, o sentido inverte-se, consequentemente, a fonte de tensão tem origem no ccTalk até ao Rx como se pode verificar pelo circulo cor-de-laranja na Figura 4.11.

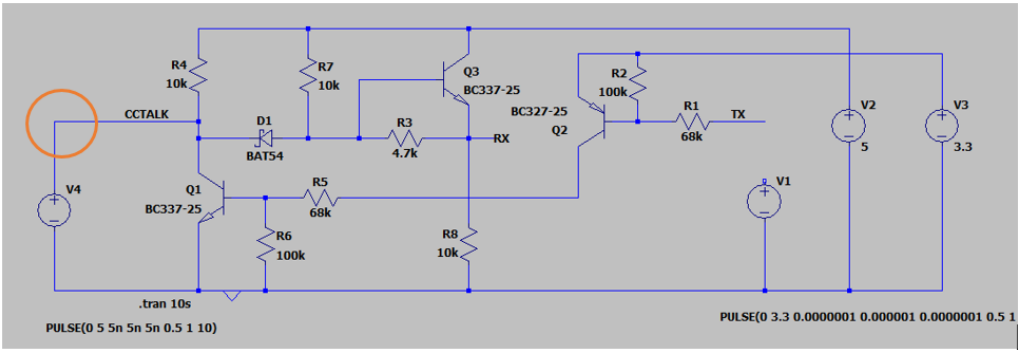


Figura 4.11: Circuito da interface ccTalk quando uma mensagem é recebida

Quando as mensagens são enviadas pelo *host* para o Dispositivo passando pelo Barramento de dados bi-direcional do ccTalk, é retornada pelo cabo Rx uma mensagem de eco, tal como se pode ver na Figura 4.12, que é o resultado obtido ao testar a interface ccTalk quando é enviada uma mensagem do controlador. É possível verificar que tanto o V(Tx) como o V(Rx) estão a transmitir ou a receber mensagens, no entanto, trata-se sempre da mesma mensagem, pois o Rx encaminha uma mensagem de eco de retorno.

Também se consegue observar que a tensão no ccTalk corresponde a 5V e a tensão tanto no Tx como no Rx é de 3.3V, pelo que desta forma foi possível confirmar o bom funcionamento do conversor de sinal.

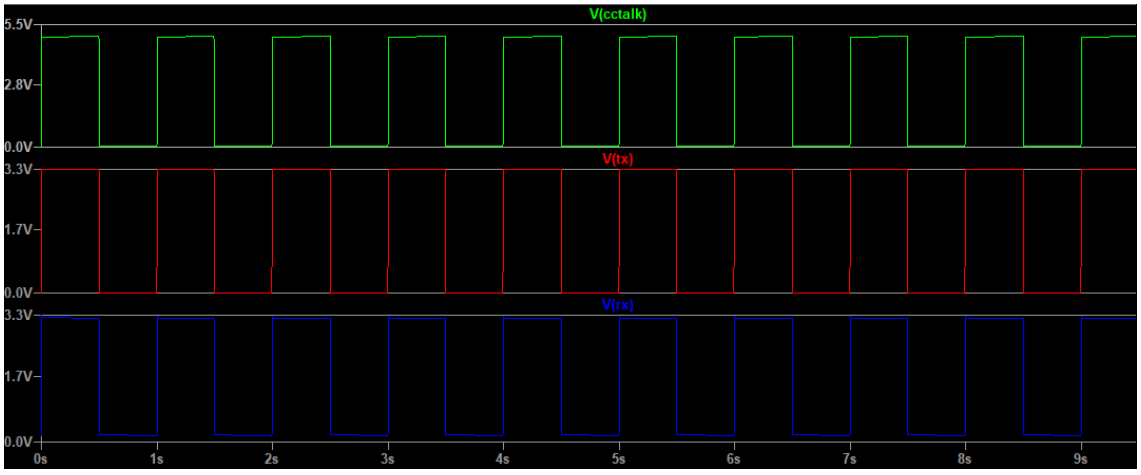


Figura 4.12: Observação no osciloscópio do output do circuito da interface ccTalk ao enviar uma mensagem

Na Figura 4.13 está representada a interface a encaminhar uma mensagem vinda do Dispositivo para o controlador. Enquanto a mensagem atravessa a interface ccTalk, o canal Tx deteta um ligeiro ruído, no entanto não apresenta qualquer tipo de interferência no sistema.

Mais uma vez, verifica-se que a tensão no ccTalk está a 5V e no Rx está a 3.3V.

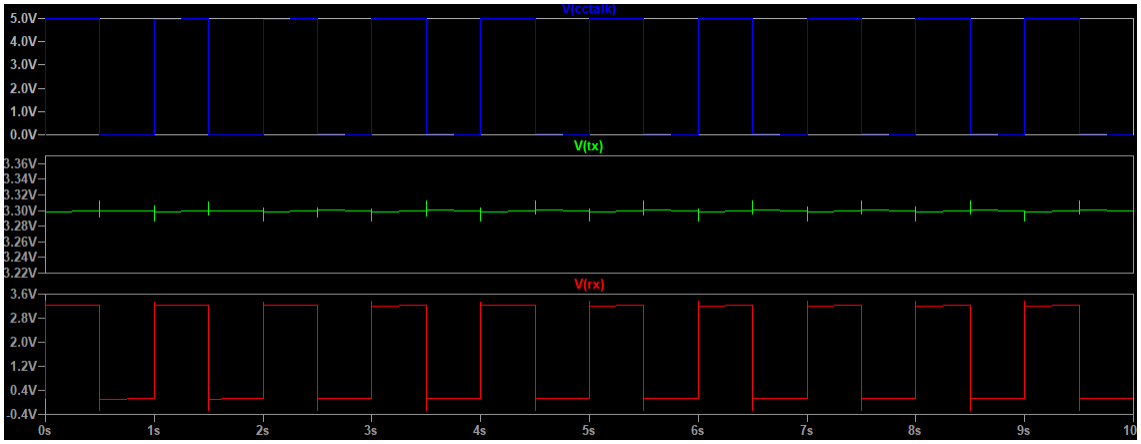


Figura 4.13: Observação no osciloscópio do output do circuito da interface ccTalk ao receber uma mensagem

4.2.4 SUH - Serial Universal Hopper

Para a fase de testes num dispositivo real, optou-se por utilizar um *hopper* também conhecido por funil, neste caso, o *Universal Hopper* com interface em Serie, SUH. Este dispositivo é produzido pela empresa *Money Controls* e é também o *hopper* utilizado pela A-to-Be e nas suas máquinas já em operação a nível nacional e internacional.

A interface em série utilizada pelo periférico é o ccTalk, pois, como se tem referido até aqui, é o protocolo de comunicação mais indicado para o controlo de dispositivos e é de baixa velocidade na indústria de transacções financeiras, mas uma das principais vantagens do ccTalk é o equilíbrio entre a simplicidade e segurança.

Um *hopper* é como um dispensador de moedas, cuja função é libertar moedas, por exemplo, dar o troco ao cliente. No caso deste dispositivo, é constituído por uma pista de moedas que é acionada por mecanismos do motor. À medida que a pista se move, vai colher as moedas que estão na parte inferior da caixa e encaminha-as para a porta de saída de moedas do dispositivo. Na porta de saídas do *hopper*, encontram-se os sensores óticos, nomeadamente transmissores infravermelhos e detetores fotográficos. Quando a moeda passa pela porta de saída do dispositivo, o raio de luz do sensor infravermelho é fragmentado e é gerado um sinal de saída da moeda. Este *hopper* em questão tem uma capacidade entre 700 a 3500 moedas, dependendo da dimensão do dispositivo, sendo que existe um compartimento com um limite máximo para cada tipo de moeda. [44]

Figura 4.14: *Hopper* SUH

Este dispositivo também está apto para o mecanismo de criptografia de 64 bits, de modo a garantir que não haja a distribuição de moedas de forma incorreta. A chave de encriptação é secreta, no entanto há procedimentos de criptografia simples e possíveis de executar, como se pode ver utilizando os comando mencionados a seguir:

1. *Pump RNG* - O controlador envia para o *hopper* um conjunto de números aleatórios, 8 bytes. Este comando não é obrigatório, no entanto é recomendado para aumentar o número de possíveis chaves de criptografia que são encaminhadas ao logo do *bus* de dados e que possam ser captadas por um *hacker*. Basicamente, utilizando este comando, apenas irá aumentar o número de possíveis chaves de encriptação tornando mais difícil a um *hacker* encontrar a chave real.

2. *Request cipher key* - O pedido de uma nova chave de encriptação deve ser feito antes da distribuição das moedas, pois após a sua distribuição a chave de encriptação é alterada e não volta a ser a mesma que era inicialmente. Este tipo de comando pode ser repetido no caso de haver um erro na comunicação, sendo que a chave será retransmitida em vez de ser reinventada. De seguida, combina-se a chave encriptada com o número de moedas a dispensar, este bloco surge no final do pacote da mensagem. A combinação da chave com o número de moedas é feita com o auxílio do Cryptographic Mapping Function (CMF), com o propósito de inverter todos os *bytes* da mensagem.

3. *Dispense hopper coins* - Sempre que o comando para dispensar moedas é acionado, a chave de encriptação é alterada e por isso a dispensa de moedas não pode ser efetuada exceto se o algoritmo (CMF) for conhecido, mas para isso comprometíamos a segurança das moedas e do Sistema, pelo que a *Money Controls* incorporou no sistema uma mecanismo não documentado para alterar o CMF sem que haja risco de segurança e alterações no sistema a nível de *hardware*.

O SUH também oferece um outro recurso de segurança, o código PIN. Nunca fase inicial, a opção da utilização do PIN está inativa e não é obrigatório o seu uso. Uma vez ativado, só é possível distribuir moedas com a utilização do código PIN e, mesmo sendo desligado ou feito um *reset* ao dispositivo, é necessário a introdução do PIN.

Caso o *hopper* se desligue acidentalmente, como por exemplo por falha de energia ou utilizando o comando de parar de emergência, o SUH possui uma memória não volátil (EEPROM) destinada ao armazenamento dos contadores de moedas, deste modo, quando ocorre este fenómeno, a situação pode ser reposta e as moedas residuais são dispensadas após o *hopper* voltar a ligar. Todo o controlo e manuseamento da moedas é feito pelo *host* e não pelo Dispositivo, pois através de determinados comandos e pedidos enviados pelos *host* é possível aceder à memoria do Dispositivo e posteriormente só o *host* tem a capacidade de decidir se deve dispensar ou não mais moedas.

Na tabela 4.2, estão representados os comandos desenvolvidos para operar com o SUH.

Tabela 4.2: Lista de comandos realizados para operar com o *hopper* SUH

Cabeçalho	**Função**
Header 254	*Simple poll*
Header 253	*Address poll*
Header 252	*Address clash*
Header 251	*Address change*
Header 250	*Address random*
Header 247	*Request variable set*
Header 246	*Request manufacturer id*
Header 245	*Request equipment category id*
Header 244	*Request product code*
Header 242	*Request serial number*
Header 241	*Request software revision*
Header 236	*Read opto states*
Header 219	*Enter new PIN number*
Header 218	*Enter PIN number*
Header 217	*Request payout high / low status*
Header 216	*Request data storage availability*
Header 215	*Read data block*
Header 214	*Write data block*
Header 192	*Request build code*
Header 172	*Emergency stop*
Header 171	*Request hopper coin*
Header 169	*Request address mode*
Header 168	*Request hopper dispense count*
Header 167	*Dispense hopper coins*
Header 166	*Request hopper status*
Header 165	*Modify variable set*
Header 164	*Enable hopper*
Header 163	*Test hopper*
Header 161	*Pump RNG*
Header 160	*Request cipher key*
Header 004	*Request comms revision*
Header 003	*Clear comms status variables*
Header 002	*Request comms status variables*
Header 001	*Reset device*

Para dispensar moedas é necessário executar uma sequência mínima de comandos para que o processo seja levado até ao fim e de forma correta; essa sequência é apresentada de seguida:

1. *Enable hopper* - Responsável por ativar o *hopper*
2. *Request cipher key* - Solicita a chave de encriptação
3. *Dispense hopper coins* - Comando que efetivamente dispensa as moedas para o exterior

4. *Request hopper status* - Revê o estado do depósito do *hopper*

Contudo, há uma sequência recomendada pelos produtores do *hopper*, que vai desde o momento em que encontra o endereço do dispositivo, caso o controlador não conheça, passando pela inicialização do *hopper* com a definição dos requisitos para o motor, até à fase de dispensa das moedas pelo *hopper*, havendo ainda uma verificação completa de erros ou moedas que possam não ter sido expelidas. Esse procedimento está descrito na secção de implementação e validação, mais adiante.

Este procedimento é uma relação entre o controlador e o dispositivo *hopper*, no entanto, todas as decisões e pedidos são tomados pelo controlador para o periférico e o controlador está apto a agir consoante as respostas vinda do Dispositivo.

4.2.5 Microcontrolador EEPROM e *Board* Teste 03

O microcontrolador a utilizar tem o nome de *Kinetis*, representado na figura 4.15, e faz parte de um módulo já outrora desenvolvido pela A-to-Be. Neste módulo é possível encontrar várias funcionalidades extras e já implementadas, como por exemplo introdução de um cartão SSD, entre outras funções.

Este módulo é constituído por um processador EEPROM 128k com 144 pinos. Consequentemente, começou-se por verificar quais os pinos que já estavam atribuídos e a sua função, como por exemplo a alimentação, *reset*, etc.

Após saber quais os pinos livres, procedeu-se à atribuição de funções aos mesmos, tendo em consideração os requisitos necessários e a finalidade já predefinida para os pinos, ou seja, um pino pode ser capaz de realizar várias funções diferentes consoante a sua configuração. O trabalho aqui consistiu em determinar qual a função para um determinado pino, como por exemplo, o pino 136 ficou definido como UART0 para o protocolo RS232 com *handshake*, no entanto, este pino também poderia ser utilizado como PTD7, pois o pino 136 está configurado para ser capaz de realizar três a quatro funções diferentes, dependendo só da sua configuração.

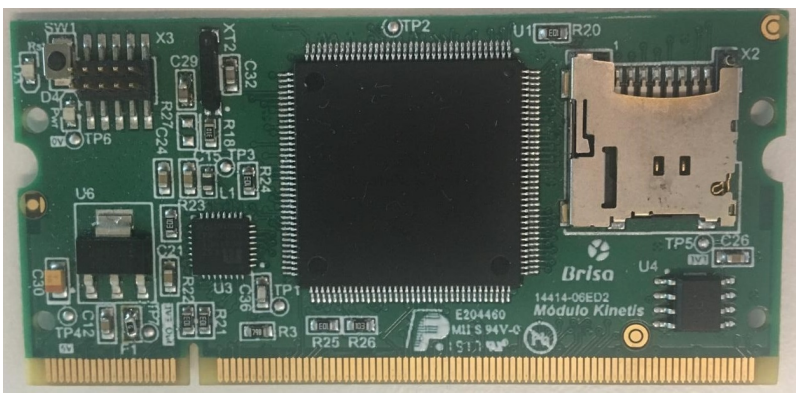


Figura 4.15: Módulo *Kinetis*

Quanto ao circuito e à produção do *Money link*, não foi possível realizá-los a tempo da entrega desta dissertação. No entanto, não foi prejudicial para o desenvolvimento da mesma, pois para substituir foi-me atribuída uma *board* de teste capaz de integrar o microcontrolador mencionado em cima.

Apesar de não ser o *Money Link*, não teve impacto, pois para esta fase do projeto, em que o maior foco fica no desenvolvimento do protocolo ccTalk para a comunicação com um periférico, apenas foi necessário que a *board* tivesse uma entrada para a interface ccTalk, sendo que esta *board* apresenta essa principal característica e também tem o adaptador para o microcontrolador, permitindo montar o sistema em ambiente de desenvolvimento de produção para testes, tal como está apresentado na secção 4.2.

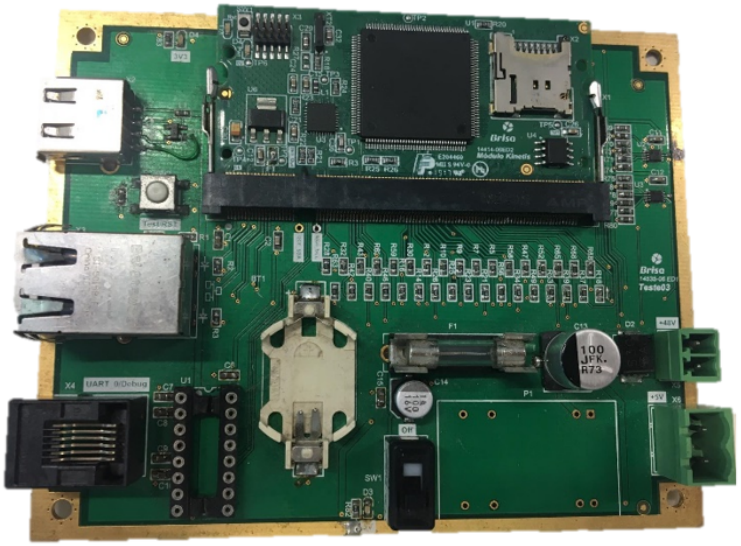


Figura 4.16: *Board* Teste 03 utilizada para testes substituindo o *Money Link*

4.2.6 Programador PEMicro *MultiLink* Universal

Este tipo de equipamento tem fortes características e funcionalidades, visto que servem maioritariamente como programadores autónomos, no entanto, têm outras utilidades como efetuar *debug* e testes.

Mais concretamente, o PEMicro, é um equipamento que acomoda uma vasta gama de processadores, particularmente ARM Cortex e NXP®, logo fez todo o sentido utilizar este programador para programar a *board* de teste utilizada neste projeto.

Este tipo de equipamento torna-se muito versátil, pois dispõem de armazenamento *on-board* de imagens de programação, bem como que suportam energia à placa e ainda suporte manual ou programação automática.

Neste caso, o inicio da programação realiza-se automaticamente ao ligar o PEMicro ao PC e através do SDK do microcontrolador.

Capítulo 5

Implementação e Validação

Nesta primeira fase de desenvolvimento do *software*, começou-se por se desenvolver a comunicação entre os dispositivos no ambiente de desenvolvimento para testes iniciais, como está representado a nível de *hardware* na Figura 5.1.

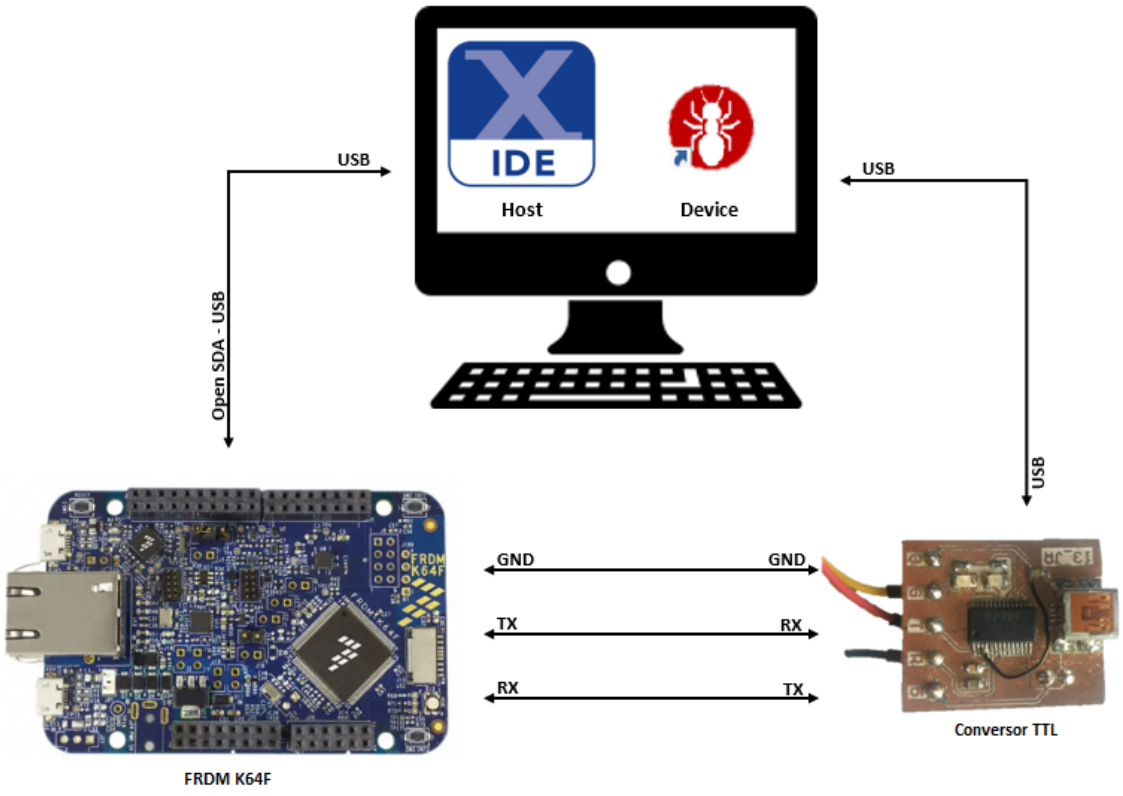


Figura 5.1: Esquema da montagem do ambiente de desenvolvimento inicial

Já se havia concluído que se viria a comunicar através da UART3, assim como já se havia definido a CPU com interrupções para parar determinada tarefa e executar outra com prioridade maior.

Posto isto, foi necessário perceber como são enviadas as mensagens do *host* para o *device*, a mensagem tem uma determinada estrutura mas o conteúdo dos diferentes campos são valores numéricos, por isso, optei por fazer um vector de inteiros, em que cada *byte* corresponde a um campo da mensagem, e tratar de cada campo da mensagem de forma individual. Ou seja, tanto a mensagem enviada como a mensagem recebida têm um vetor associado, como se pode ver em seguida:

- Mensagem enviada pelo *host*:
  - vetor_to_send[0] = id_destino
  - vetor_to_send[1] = num_bytes_dados
  - vetor_to_send[2] = id_origem
  - vetor_to_send[3] = metodo
  - vetor_to_send[4 + num_bytes_dados] = data_to_send
  - vetor_to_send[5 + num_bytes_dados] = checksum
- Mensagem enviada pelo *device*:
  - vetor_to_receive[0] = id_destino
  - vetor_to_receive[1] = num_bytes_dados
  - vetor_to_receive[2] = id_origem
  - vetor_to_receive[3 + num_bytes_dados] = data_to_recieve
  - vetor_to_receive[4 + num_bytes_dados] = checksum

Como os dados a enviar não são variáveis constantes, depende do número de *bytes* enviados, foi necessário fazer ciclos *for* para encontrar e definir o valor para a variável do número de *bytes* e, consequentemente, as posições do vetor também não são fixas. Por sua vez, o *checksum* também ficará na posição do vetor imediatamente após o término dos dados.

Na imagem seguinte, é apresentado um exemplo de uma mensagem enviada pelo host, neste caso é o comando 218 para inserir o número PIN. Nesta situação, será o *host* a enviar a mensagem para o *device*, logo o ID de origem é o do *host* que é igual a 1, como já foi visto anteriormente, e o ID de destino é o ID do *device* e, como se viu na secção do SUH, 4.2.4, o endereço indicado para o *device* será 3. No nível superior da Figura 5.2, observa-se a estrutura da mensagem e no nível inferior está apresentada a mensagem efectivamente enviada segundo a correspondência de cima.

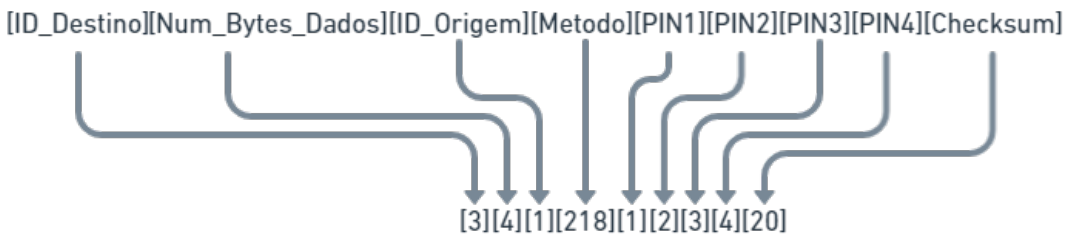


Figura 5.2: Exemplo da mensagem enviada do *host* para o *device* para inserir o código PIN

Entretanto, o controlador, enquanto espera pela resposta do *device* ao comando 218, pode estar a fazer outras tarefas. No entanto, quando o *hopper* reagir ao comando 218, a CPU sofre uma interrupção e fica a escutar a resposta do periférico. Nesta etapa, o ID de origem e de destino trocam, uma vez que agora é o *device* a enviar a mensagem de retorno para o *host*, mantendo-se na mesma a estrutura da mensagem. Também como se pode observar na Figura 5.3, não surge o método, pois o controlador, embora esteja a monitorizar outros dispositivos, a *board* saberá que, quando o *hopper* responder, será ao método 218.

Neste cenário, a resposta do *hopper* ao comando 218 é uma mensagem de ACK, conforme se pode observar na Figura 5.3 a qual será a resposta ao *device* ao pedido do controlador.

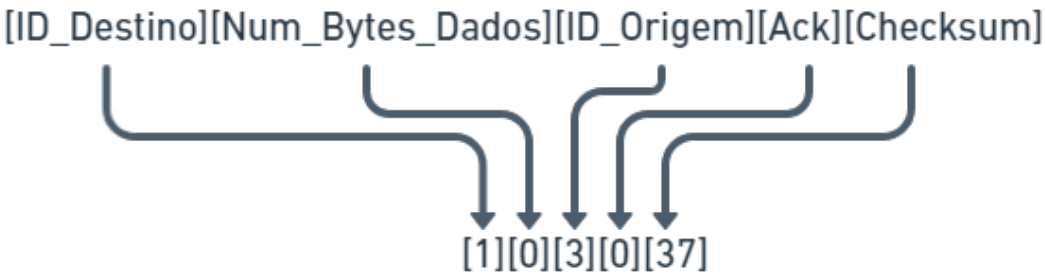


Figura 5.3: Exemplo da mensagem enviada do *device* para o *host* em resposta ao comando 218

Quanto às mensagens recebidas pelo *hopper*, também optei por utilizar um vetor de números inteiros para uma maior facilidade em desmembrar as mensagem e possibilitar a separação de vários valores, fazendo-os corresponder a uma única entidade. Por outras palavras, quando é recebida uma mensagem vinda do *device*, vem sob a forma de vários números inteiros seguidos, então, tendo um vetor de números inteiros, torna-se mais simples separar valores e corresponder as várias posições do vetor com o valor correto a um campo da estrutura da mensagem.

Uma vez compreendida a forma de estruturar as mensagem enviadas e recebidas, procedeu-se ao desenvolvimento de todos os comandos executados pelo *hopper*, apresentados na Tabela 4.2, da secção do *Serial Universal Hopper*.

Quanto ao código em si, temos vários ficheiros, nomeadamente:

- Metodos.h - É uma biblioteca de domínio público, onde é apresentada uma listagem de todos os comandos que os *devices* são capazes de executar com os respectivos valores de 1 a 255, como se pode verificar na figura seguinte 5.4 um pequeno trecho deste ficheiro

```
enum ccTalk_metodo {
    CCTALK_METHOD_RESET_DEVICE = 1,
    CCTALK_METHOD_REQUEST_COMMS_STATUS_VARIABLES = 2,
    CCTALK_METHOD_CLEAR_COMMS_STATUS_VARIABLES = 3,
    CCTALK_METHOD_REQUEST_COMMS_REVISION = 4,
    CCTALK_METHOD_READ_BARCODE_DATA = 129,
    CCTALK_METHOD_REQUEST_INDEXED_HOPPER_DISPENSE_COUNT = 130,
    CCTALK_METHOD_REQUEST_HOPPER_COIN_VALUE = 131,
    CCTALK_METHOD_EMERGENCY_STOP_VALUE = 132,
    CCTALK_METHOD_REQUEST_HOPPER_POLLING_VALUE = 133,
    CCTALK_METHOD_DISPENSE_HOPPER_VALUE = 134,
    CCTALK_METHOD_SET_ACCEPT_LIMIT = 135,
```

Figura 5.4: Metodos.h - Ficheiro que define todos os comandos a serem executados pelo *hopper*

- Host_functions.c - É o ficheiro que possui todos os comportamentos que o *host* deve tomar, basicamente é um ficheiro que no futuro não será de sofrer alterações, mesmo que se mude de periférico, uma vez que o controlador tem o mesmo comportamento para todo o tipo de periféricos, apenas mudando o ID do *device*. Deste modo, é um ficheiro global que apresenta todos os comportamentos do controlador face ao método em questão.

- Device_functions.c - Trata-se de um ficheiro idêntico ao anterior, mas neste caso interpreta as mensagens recebidas do *device*. Este ficheiro poderá sofrer alterações caso se altere para outro tipo de periférico, pois, apesar do comando ser o mesmo, o tipo de resposta poderá ser diferente, por consequência o controlador terá que se adaptar.

- Verificação_erros.c - É o ficheiro que contém as funções responsáveis pela verificação de erros.

- Final_Test.c - Este é o ficheiro principal, contém a função responsável pelas interrupções quando é recebida uma mensagem, também dispõe da função principal para o *host* enviar a mensagem com a estrutura definida. Assim como também possui a função *main*, onde é iniciada toda a ação, primeiro pela inicialização do *hardware* da *board*, ou seja, *baudrate*, os pinos do dispositivo, a frequência do *clock* e o *debug* da consola. Após a inicialização do controlador, é dado inicio ao programa em si,

que fica num ciclo infinito a produzir uma determinada sequência de métodos. Sendo que há uma função dentro do *main* que está sempre a escutar possíveis mensagens que possam vir do Dispositivo, mas na verdade um Dispositivo apenas vai responder aquando uma ordem vinda do controlador e respondendo ao mesmo método enviado pelo controlador.

A sequência de métodos que descreve todo o comportamento que o *host* deve ter em consideração de modo a executar o procedimento completo da dispensa de moedas, está representado na Figura 5.5. Sendo esta a sequência recomendada pelos fabricantes do SUH, como foi visto na secção do *hopper*, 4.2.4.

Tendo sido esta a sequência recomendada para a execução desta dissertação, passou-se ao desenvolvimento destes procedimentos, bem como, o estudo dos seus resultados.

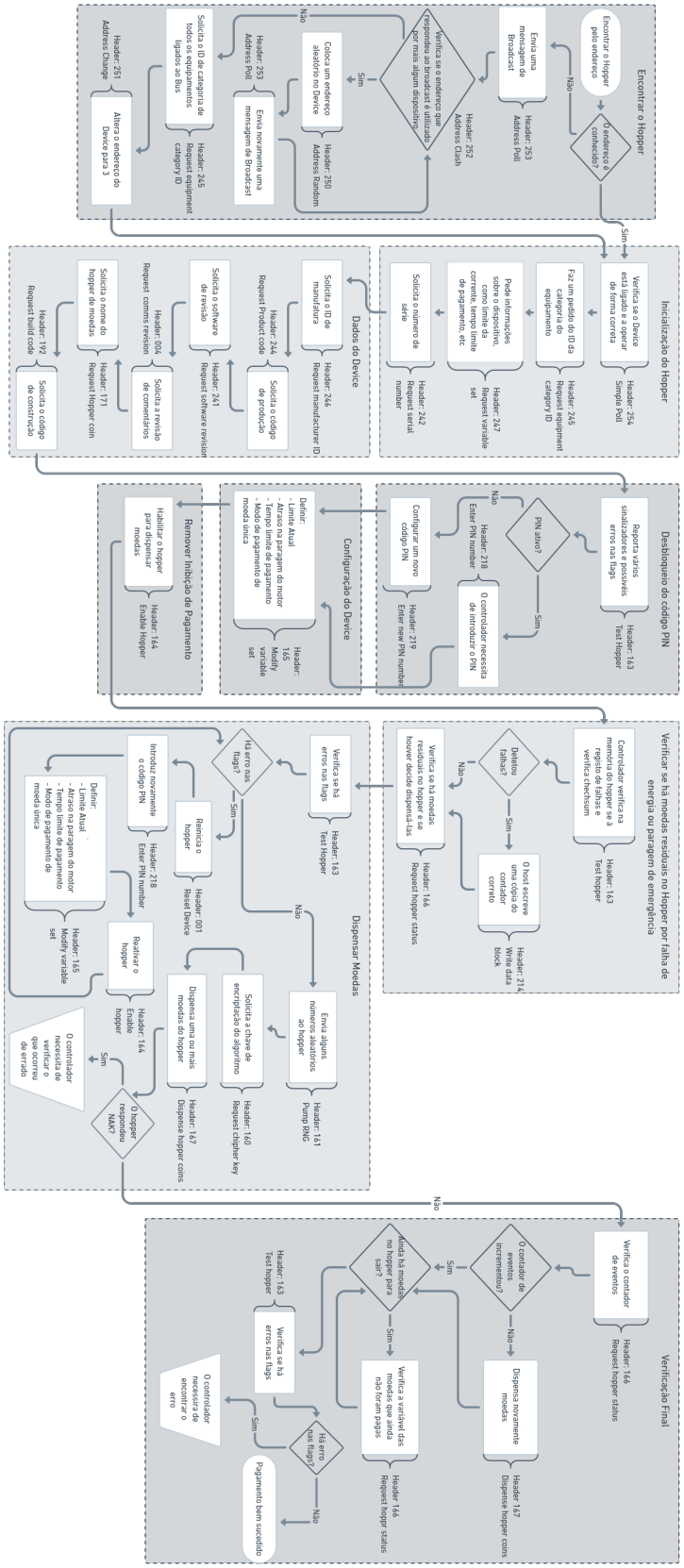


Figura 5.5: Fluxograma que representa um procedimento completo realizado entre o controlador e o *hopper*

Na Figura 5.6, é possível analisar a máquina de estados referente a um dos procedimentos constituintes da sequência completa que foram tomados, sendo neste caso, o pedido dos dados do *device*. Apesar de nesta máquina de estados estar apenas representado o dados do periféricos, todo o fluxograma pode ser descrito na máquina de estados, visto ter sido essa uma das sequências de teste efetuadas na elaboração deste projeto.

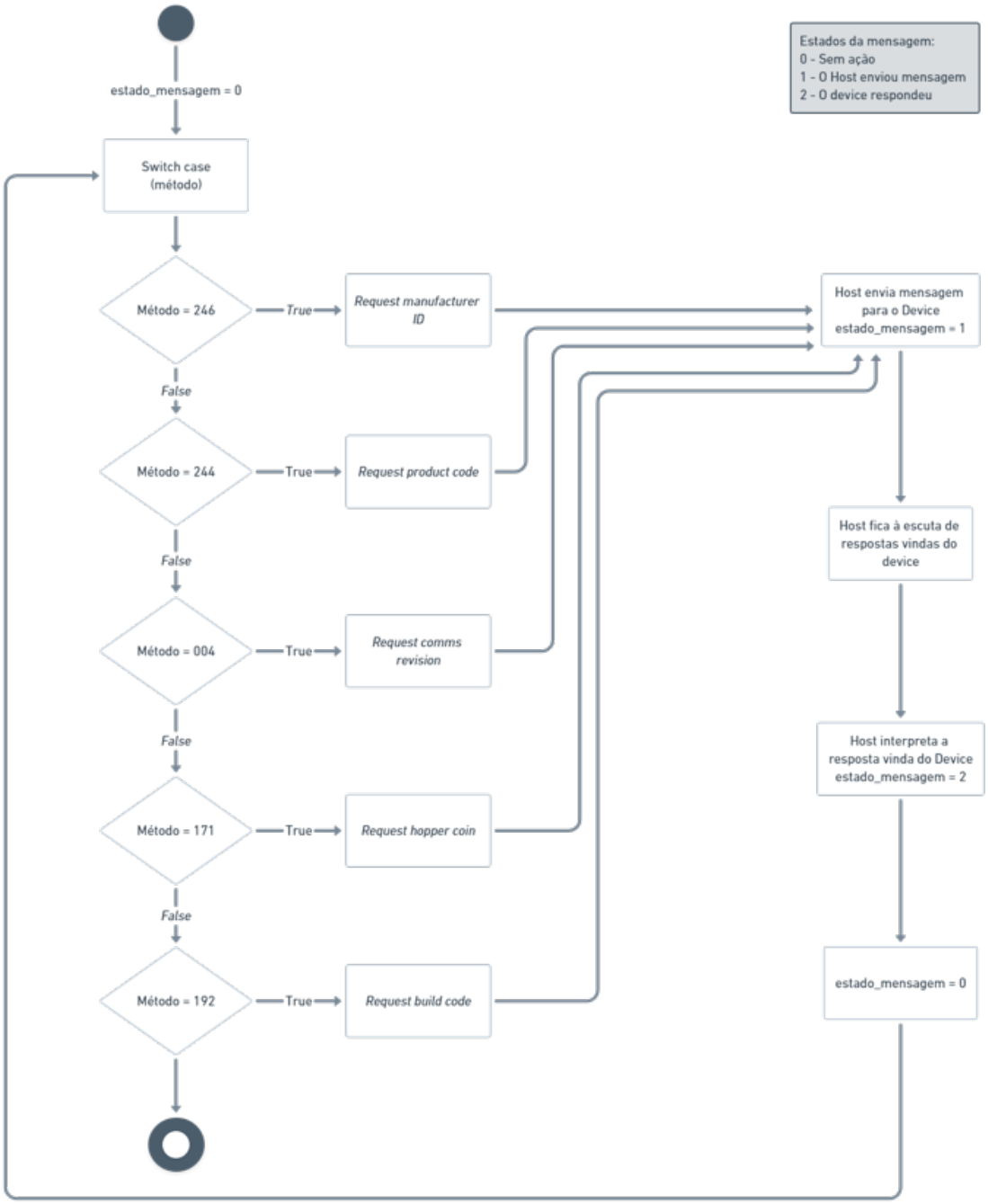


Figura 5.6: Máquina de estados da sequência de procedimento realizado pelo *host* afim de pedida os dados do *device*

No código, esta máquina de estados encontra-se dentro da função *main* que, por sua

vez, possui uma função *switch case*, sendo esta responsável por passar ao próximo método da sequência e por atualizar os estados da mensagem. Neste exemplo, foi dada apenas uma pequena demonstração de como está implementada a função *main* e, caso se pretenda alterar a sequência, basta adicionar novos métodos ou alterar os que já lá estão, pois, através dos métodos, o controlador saberá como agir e saberá o que responder.

Os resultados desta pequena combinação foram testados no ambiente de desenvolvimento implementado para a fase de testes iniciais, através da board FRDM K64F e são apresentados na seguinte imagem 5.7. Como foi dito na secção do ambiente de desenvolvimento 4.2, nesta fase do projeto, foi testado em ambiente de desenvolvimento de testes preliminares, pelo que os resultados apresentados em baixo são os obtidos na consola do IDE.

```
Metodo a executar: Request manufacturer id
O Host enviou a seguinte mensagem:
[3][0][1][0][5]

O Device respondeu a seguinte mensagem:
[1][14][3][M][o][n][e][y][][C][o][n][t][r][o][l][s][44]
ID de fabrica: Money Controls

-------------------------------------------------------

Metodo a executar: Request product code
O Host enviou a seguinte mensagem:
[3][0][1][0][7]

O Device respondeu a seguinte mensagem:
[1][4][3][S][U][H][1][161]
Codigo do produto: SUH1

-------------------------------------------------------

Metodo a executar: Request software revision
O Host enviou a seguinte mensagem:
[3][0][1][0][14]

O Device respondeu a seguinte mensagem:
[1][9][3][S][U][H][1][-][V][x][.][y][43]
O pedido de revisao foi efetuado: SUH1-Vx.y

-------------------------------------------------------

Metodo a executar: Request comms revision
O Host enviou a seguinte mensagem:
[3][0][1][0][89]

O Device respondeu a seguinte mensagem:
[1][3][3][1][3][2][186]

A mensagem recebida sera do tipo: [ccTalk level] [major revision] [minor revision]
A mensagem recebida:
cTalk Level = 1
Major Revision = 3
Minor Revision = 2
Que significa: the first issue level of ccTalk specification 3.2

-------------------------------------------------------

Metodo a executar: Request hopper coin
O Host enviou a seguinte mensagem:
[3][0][1][0][107]

O Device respondeu a seguinte mensagem:
[1][6][3][C][C][V][V][V][I][78]
A mensagem recebida:[C][C][V][V][V][I]

-------------------------------------------------------

Metodo a executar: Request build code
O Host enviou a seguinte mensagem:
[3][0][1][0][95]

O Device respondeu a seguinte mensagem:
[1][8][3][L][e][v][][H][i][L][o][66]
'Lev HiLo' para sensor de nivel alto e baixo
```

Figura 5.7: esultado obtido no terminal no contexto do ambiente de desenvolvimento inicial

Posto isto, também se realizou a sequência de comandos recomendada pelos autores do *device* SUH - embora tivessem que ser ajustados alguns dos comandos enviados pelo *host* e interpretação das mensagens recebidas, verificou-se que o sistema funcionava na *board* de teste FRDM K64F.

Para concluir a fase de testes, passou-se aos testes num ambiente de produção, nesta fase já com o dispensador de moedas SUH, tal como foi explicado na secção do ambiente de desenvolvimento 4.2. Para tal, programou-se a *board* de teste através do programador PEMicro. Por sua vez, a mensagem é encaminhada pela interface ccTalk até por fim chegar ao SUH para este responder e a mensagem de resposta ser transmitida no sentido inverso. Tal como se pode verificar na Figura 5.8 a montagem do sistema a nível de *hardware*.

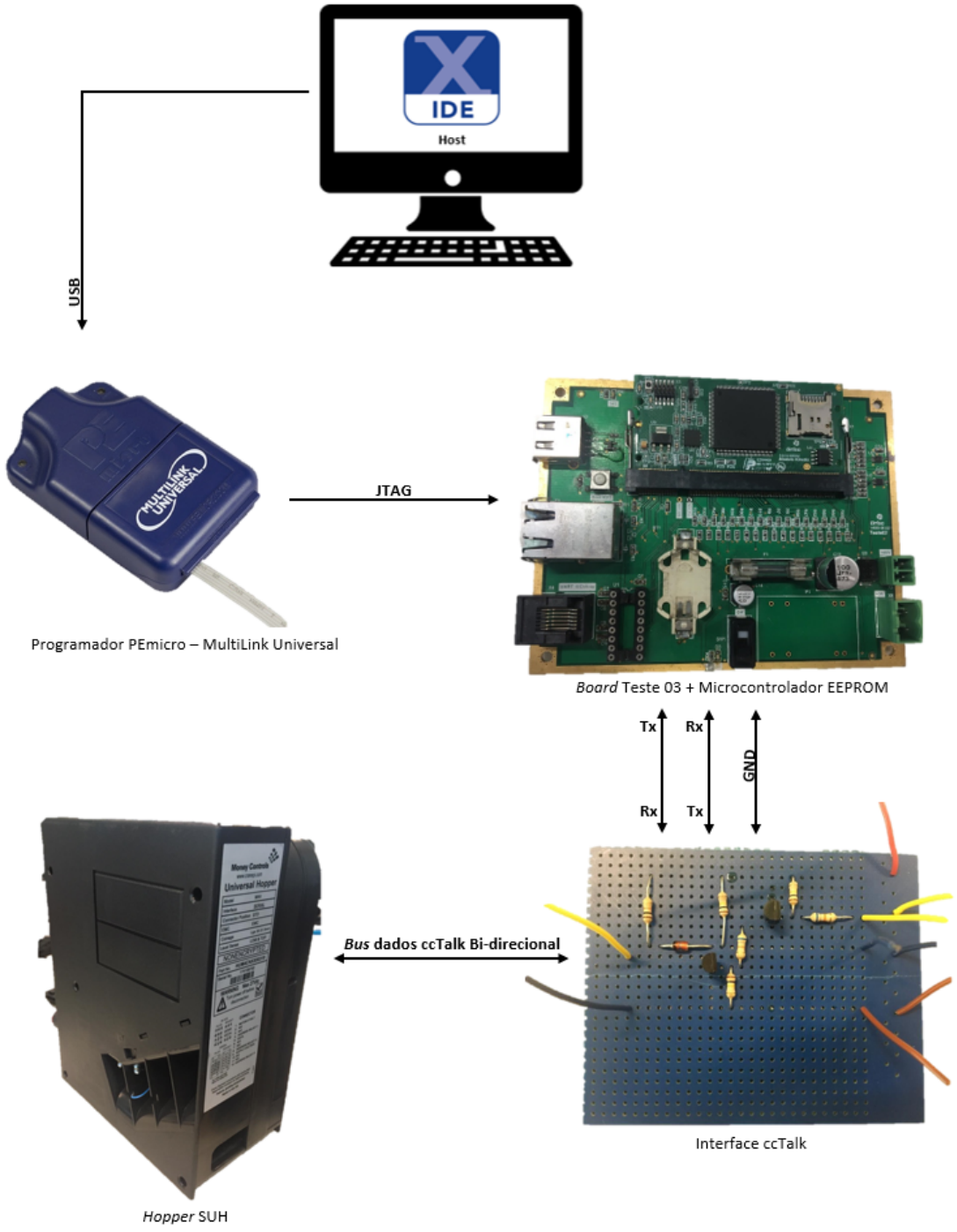


Figura 5.8: Esquema da montagem do ambiente de desenvolvimento de produção

Como nesta fase de teste já não é possível utilizar o terminal *termite*, e uma vez que o sistema já dispõe de periférico físico, sendo que este não apresenta nenhum *display*, não é possível perceber de forma plausível os resultados da comunicação como foi apresentada na figura 5.7, no entanto, através da utilização do osciloscópio ligado de forma correta à interface ccTalk, é possível determinar as mensagens enviadas e recebidas e comprovar o seu bom funcionamento.

Então, analisando o seguinte caso, em que o controlador manda uma mensagem a pedir a categoria do *device*, cujo o número do método é 245, percebe-se então que o *host* deveria de enviar e receber as seguintes mensagens:

Figura 5.9: Mensagem enviada pelo *host* e a respetiva resposta ao comando 245

Segundo a Figura 5.9 apresentada anteriormente, podemos converter para os seguintes dados da mensagem enviada:

Tabela 5.1: Conversão dos valores da mensagem enviada pelo *host*

Vetor_to_send[]	**Campo da mensagem**	**Decimal**	**Hexadecimal**	**Binário**
0	ID_Destino	3	0x03	00000011
1	Num_bytes_dados	0	0x00	00000000
2	ID_Origem	1	0x01	00000001
3	Metodo	245	0xF5	11110101
4	Checksum	7	0x07	00000111

O único método para testar o funcionamento e interpretar as mensagens enviadas e recebidas, é através da observações de valores em binário no osciloscópio. Na Figura seguinte, 5.11, está representa a mensagem enviada pelo *host*.

Figura 5.10: Resultado observado no osciloscópio da mensagem enviada pelo controlador

No resultado obtido no osciloscópio, são visíveis duas ondas, sendo a onda amarela a mensagem efectivamente enviada e a azul a mensagem de eco recebida, no entanto, já se esperava este tipo de resultado, tal como foi visto na secção da interface do ccTalk 4.2.3.

Comparando os valores em binário da mensagem apresentados na tabela 5.1 com os valores apresentados no resultado observado no osciloscópio, pode-se verificar que os resultados estão escritos na ordem certa, mas de forma contrária, ou seja, por exemplo, o 3 em binário é escrito 00000011, no entanto o osciloscópio apresenta-o de forma contrária, isto é, 11000000; este fenómeno é uma consequência de estar a utilizar UART, pois a UART escrever por ordem os valores, desde o *bit* menos significativo até ao mais significativo, convertendo assim a ordem dos bits, no entanto está correto.

Também se pode verificar que entre cada campo da mensagem são enviados dois *bits* de separação, *Start Bit* e *Stop Bit*, isto é, um mecanismo de sincronismo da interface ccTalk, visto isto ser um sistema assíncrono, o facto de apresentar estes dois *bits* é uma forma do sistema ficar harmonioso, caso contrario, a mensagem poderia ficar confusa no que toca aos campos da estrutura da mensagem.

Este cenário verifica-se para todos os campos da mensagem e portanto conclui-se que a mensagem é enviada com sucesso.

Analisando a comunicação completa, ou seja, a mensagem enviada e a mensagem recebida, também se verificam os resultados pretendidos.

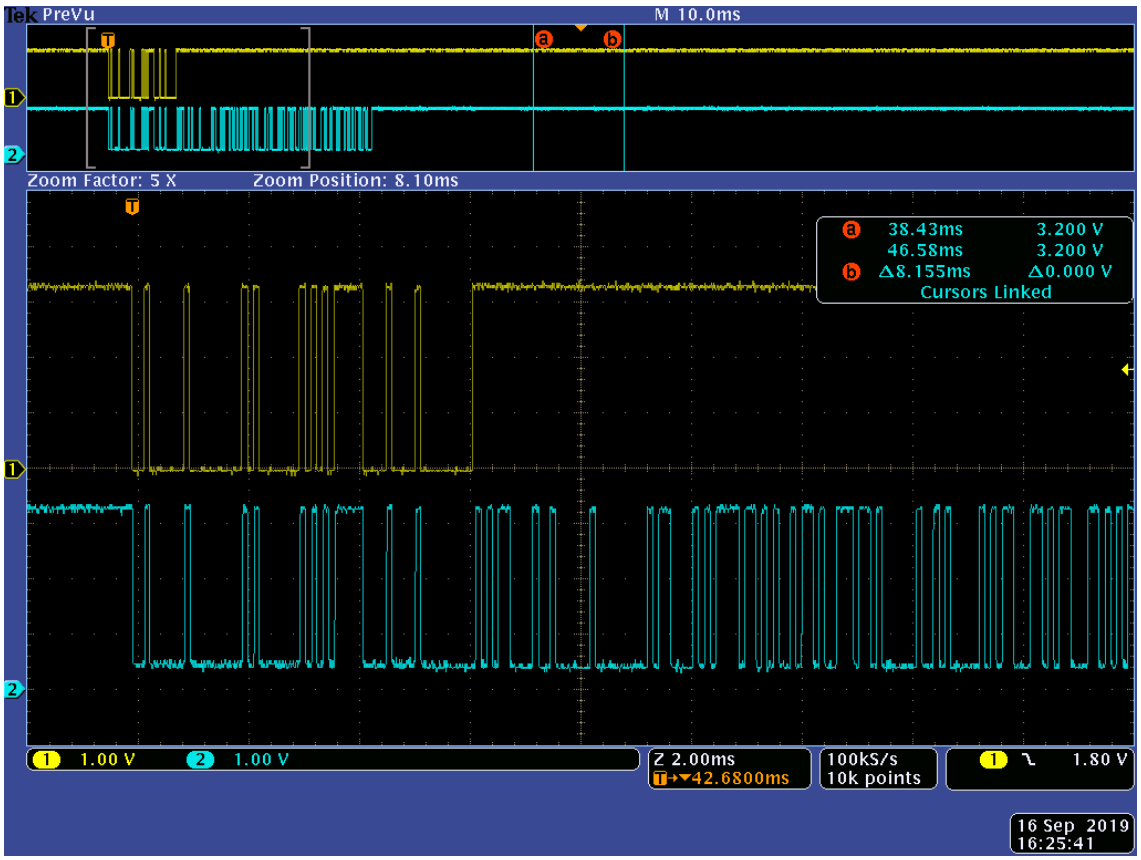


Figura 5.11: Resultado observado no osciloscópio da mensagem enviada e recebida ao comando 245

Para concluir, a análise dos resultados e a sua validação foram fundamentais para melhorias do *software*, pois muitas das mensagens recebidas pelo *device* estavam a ser mal interpretadas, pelo que foi necessário testar o sistema em ambiente de produção para efectivamente perceber como era enviada a mensagem do *device*. Estas correções foram realizadas através da observação no osciloscópio de todas as mensagens recebidas, bem como a sua alteração à posteriori do device_functions.c.

Caso, numa próxima fase, se pretenda estender este *software* a outro tipo de periférico, mantendo-se o protocolo de comunicação, o ccTalk, será apenas necessário adaptar o ficheiro device_functions.c, uma vez que poderão mudar meramente a interpretação de alguns comandos.

Como é o exemplo do comando 245, no caso de se utilizar o *hopper* SUH a resposta será "*Payout*", no entanto o comando 245, num aceitador de moedas, a resposta seria "*Coin Acceptor*", ou seja, a mensagem enviada do controlador para o *device* seria igual, no entanto a resposta diferia e, consequentemente, o número de *bytes* de dados, sendo que este tipo de comando não teria qualquer influência no código. Claro que há outros comandos que teriam que sofrer grandes alterações, pois o *hopper* SUH tem o objetivo de dispensar moedas, enquanto que um aceitador de moedas teria que recebê-las e, por isso, os comandos mais específicos teriam que sofrer alterações, mas apenas na sua interpretação.

Capítulo 6

Conclusões e Trabalhos Futuros

Para concluir, é feita uma reflexão relativamente às dificuldades sentidas ao longo da realização desta dissertação. Com base na análise desses aspetos, serão propostos trabalhos futuros e possíveis melhorias ao sistema.

6.1 Conclusões

No decorrer desta dissertação houve algumas alterações ao plano inicial, uma vez que a A-to-Be propôs o desenvolvimento de um controlador completo com várias interfaces físicas e capaz de comunicar, interagir, monitorizar e suportar vários protocolos de comunicação, passando por tarefas como desenho e produção do circuito eléctrico, bem como desenvolver o *software* que suportasse as especificações mencionadas anteriormente. E, por fim, ainda era pedida a elaboração de uma aplicação genérica para os sistemas *host* que poderiam utilizar o controlador.

No entanto, houve tarefas que não foram possíveis de realizar, devido a incompatibilidades e adversidades que foram surgindo. Por consequência, e de acordo com as exigências do projeto inicial, optou-se, em conjunto com a A-to-Be, por só se habilitar o controlador a operar com um único dispositivo, o SUH, e apenas um único protocolo de comunicação, o ccTalk. No entanto, o sistema desenvolvido é bem capaz de interagir com outro tipo de periféricos e, caso numa próxima etapa do projeto seja conveniente passar a esta fase, apenas será necessário adaptar um dos ficheiros desenvolvidos em *software*.

Foi consequência da demorada compreensão do protocolo de comunicação ccTalk, dado ser um protocolo bastante extenso, completo e complexo. Outro fator que impossibilitou a conexão do controlador a outros periféricos diferentes foi o surgimento de alguns problemas na *board* FRDM K64F, pois de alguma forma desapareceu o *firmware* da placa, o que impossibilitou a realização de mais testes e o avanço para outros objetivos enquanto

este problema não ficasse resolvido; após ter ficado corrigido, optou-se por testar em ambiente real com *board* Teste 03 e apenas com o *hopper* SUH.

Quanto ao desenho e à produção do controlador *Money Link*, não foi alcançável por motivos de incompatibilidade entre projetos dos colaboradores da A-to-Be. No entanto, A autora efetuou uma tabela confidencial que servirá como auxilio quando se for realizar o desenho do circuito elétrico bem como a sua produção, pois a tabela apresenta todas as configurações e definições dos pinos, potência, portas de entrada e saída, interfaces físicas, entre outros parâmetros a serem utilizados no *Money Link*.

No entanto, tendo em conta as tarefas efetuadas, estas vão ao encontro da maioria das especificações exigidas pela A-to-Be, embora algumas não tenham sido realizadas a nível visível, todavia foi feita a preparação, estudo e levantamento da maioria das tarefas que podem ser executadas em trabalhos futuros. Bem como para esta fase inicial do projeto correspondeu aos critérios exigidos para a execução desta dissertação.

Porém, excluindo todos os percalços encontrados no decorrer desta dissertação, foram correspondidas às expectativas, visto ter-se conseguido programar um controlador que por sua vez comunicou e monitorizou um periférico através do protocolo de comunicação série ccTalk. Consequentemente, testar com sucesso o sistema em ambiente de desenvolvimento de testes inicias, assim como em ambiente de produção. Deste modo, verificou-se que o sistema está capaz de, neste caso, dispensar moedas e monitorizar todas as suas funções.

Para concluir, de forma geral, a execução desta dissertação foi conseguida e observada por diversas vezes, bem como melhorada ao longo da execução da mesma, porém é notória a necessidade de continuidade do projeto, pois estando nesta fase de inicialização ficaram algumas das funcionalidades por realizar bem como completar outras tantas onde já realizei alguns estudos. Importa destacar ainda que a fasquia foi posta num nível elevado no que toca ao número de tarefas face ao período disponível para a execução deste projeto na totalidade.

6.2 Trabalhos Futuros

Como já foi dito na secção das conclusões 6.1, este projeto está numa fase muito embrionária, tendo sido a autora a dar início a todo o processo, pelo que ficaram algumas tarefas para execução numa próxima fase.

Na opinião da autora, deveria começar-se pelo desenvolvimento do desenho do circuito eléctrico bem como a sua produção física, após isto, facilitaria muito a quem fosse desenvolver as próximas interfaces com os respetivos protocolos de comunicação e para diferentes periféricos.

Tendo o controlador *Money Link* já finalizado, segue-se o desenvolvimento de todos os protocolos de comunicação suportados por todas as interfaces compatíveis com a *board* mencionadas na secção onde foi feito o levantamento dos componentes do *Money Link*, na secção 4.1, ou seja, protocolo tais como ccTalk, ID003, MDB, I2C, entre outros para todos os tipos de periféricos que operem com numerário.

Assim, concluídas estas duas tarefas tem-se o controlador *Money Link* totalmente apto a operar com todo o tipo de periféricos, independentemente do protocolo que comunicação que utilizarem.

É de recordar que o *Money Link* só tem o objectivo de operar com periféricos que monitorizem e façam a gestão de dinheiro em numerário.

Bibliografia

[1] Aardvark. *Products: Milan / Paylink Interface*. 2019. URL: http://www.aardvark.eu.com/products/milan/index.htm.

[2] Aardvark. *Products: Milan / Paylink Interface - Technical Details*. 2019. URL: http://www.aardvark.eu.com/products/milan/techdetails.htm.

[3] Aardvark. *Services: Custom Development*. 2019. URL: http://www.aardvark.eu.com/services/index.htm.

[4] Aardvark. *Support and Downloads: Milan / Paylink Interface*. 2019. URL: http://www.aardvark.eu.com/products/milan/support.htm.

[5] M. C. Aardvark. "PayLink Technical Manual". Em: United Kingdom, UK, 2009. URL: http://www.aardvark.eu.com/downloads/documents/PayLinkTechnicalExtract.pdf.

[6] M. C. Aardvark. "Milan/Paylink Application Program Interface Manual". Em: United Kingdom, UK, 2013. URL: http://www.aardvark.eu.com/downloads/documents/MilanPaylinkProgrammersManual.pdf.

[7] M. C. Aardvark. "Milan/Paylink System Manual". Em: United Kingdom, UK, 2016. URL: http://www.aardvark.eu.com/downloads/documents/MilanPaylinkSystemManual.pdf.

[8] G. Azkoyen. "Technical Information: HOPPER U – II". Em: *HOPPER U- II*. 2007. URL: http://www.coinmech.com/images_products/images_supporting/1715.pdf.

[9] A. Barson. "ccTalk Generic Specification - Part 3". Em: England: Crane Payment Solutions Ltd., 2013. URL: https://cctalktutorial.files.wordpress.com/2017/10/cctalkpart3v4-7.pdf.

[10] A. Barson. "ccTalk Serial Communication Protocol - Generic Specification". Em: England: Crane Payment Solutions Ltd., 2013. URL: https://projekte.turmlabor.deâĂžcctalkPart1v4.6.pdf.

[11] *Brisa - Rede em Operação e Concessionárias*. 2019. URL: https://www.brisa.pt/pt/Empresa-e-EstratÃĬgia/O-Grupo-Brisa/Rede-em-OperaÃğÃčo.

[12] *ccTalk tutorial , testing the cable with some basic commands*. 2019. URL: https://cctalktutorial.wordpress.com/2015/09/10/111/.

[13] *Conceitos Gerais de Comunicação Serial*. 2015. URL: http://digital.ni.com/public.nsf/allkb/32679C566F4B9700862576A20051FE8F?OpenDocument#485.

[14] *Conversor de RS232 para TTL*. 2019. URL: https://multilogica-shop.com/conversor-de-rs232-para-ttl.

[15] *COTEC,Portugal - BRISA AUTO-ESTRADAS DE PORTUGAL, SA*. 2019. URL: http://www.cotecportugal.pt/pt/quem-somos/associados/brisa-auto-estradas-de-portugal-sa.

[16] *Crane Payment Innovations - CPI*. 2019. URL: https://www.cranepi.com/en.

[17] *e²c. KS1000*. 2017. URL: https://etwoc.com/wp-content/uploads/2017/02/KS1000-Smart-Controller-Overview-Jan.-2017_public.pdf.

[18] *e²c. CC100*. 2019. URL: https://etwoc.com/products/cc100-2/.

[19] *e²c. CC200*. 2019. URL: https://etwoc.com/wp-content/uploads/2016/12/CC200-Product-Brief.pdf.

[20] *e²c. CC200 e Periféricos*. 2019. URL: https://etwoc.com/cc200-smart-device-hub_new/.

[21] *e²c. e²c*. 2019. URL: https://etwoc.com/e2c-platform-technology-solutions/.

[22] *e²c. e²c*. 2019. URL: https://etwoc.com/cash-control-series-smart-hubs/.

[23] *e²c. iSocket*. 2019. URL: https://etwoc.com/wp-content/uploads/2012/04/iSocket-Product-brief-1.0.pdf.

[24] *e²c. KS1000*. 2019. URL: https://etwoc.com/ks1000-embedded-computer/.

[25] *e²c. KS1000*. 2019. URL: https://etwoc.com/wp-content/uploads/2017/02/69-0010-KS1000-Product-Brief.pdf.

[26] *Grupo Azkoyen*. 2019. URL: https://www.azkoyen.com/en/.

[27] HWe Hemisphere West Europe. *Bank Note Acceptance*. 2019. URL: https://www.hweurope.com/knowledge-centre/bank-note-acceptance.html.

[28] HWe Hemisphere West Europe. *Bank Note Dispensing*. 2019. URL: https://www.hweurope.com/knowledge-centre/bank-note-dispensing.html.

[29] HWe Hemisphere West Europe. *ccTalk Interface*. 2019. URL: http://www.hweurope.com/docs/interface/hwe-cct-brochure.pdf.

[30] HWe Hemisphere West Europe. *Coin Acceptance*. 2019. URL: https://www.hweurope.com/knowledge-centre/coin-acceptance.html.

[31] HWe Hemisphere West Europe. *Credit Card Acceptance*. 2019. URL: https://www.hweurope.com/knowledge-centre/credit-card-acceptance.html.

[32] *JCM Global*. 2019. URL: https://emea-en.jcmglobal.com/products/vendingkiosk/.

[33] *Kinetis K64F Sub-Family Data Sheet*. 2016. URL: https://www.nxp.com/docs/en/data-sheet/K64P144M120SF5.pdf.

[34] B. Kuruvilla. *NPN Transistor: Definition and Equations*. 2019. URL: https://study.com/academy/lesson/npn-transistor-definition-equations.html.

[35] A. Lombardi. *How Does a Cash Recycler Work? 5 Simple Answers*. 2017. URL: https://www.cashtechcurrency.com/blog/how-does-a-cash-recycler-work-5-simple-answers.

[36] A. Lombardi. *Note Recyclers 101: What Are They and How Do They Work?* 2017. URL: https://www.cashtechcurrency.com/blog/note-recyclers-101-what-are-they-and-how-do-they-work.

[37] "MDB Converter Manual". Em: 2016. URL: https://www.abrantix.com/files/cto_layout/img/products/MDB2PC/Abrantix%20MDB%20Converter%20Manual.pdf.

[38] *NXP Semiconductors Plataforma de desenvolvimento FRDM-K64F Freedom*. 2019. URL: https://pt.mouser.com/new/freescale-semiconductor/freescale-frdm-k64f-dev-board/.

[39] B. Pereira. *Sistemas Simplex, Half-duplex e Full-duplex*. 2012. URL: https://brunomigg.wordpress.com/2012/09/27/sistemas-simplex-half-duplex-e-full-duplex/.

[40] E. Schumaki. *O que é Bit de paridade?* 2016. URL: https://espacotechinfromatica.wordpress.com/2016/01/14/o-que-e-bit-de-paridade/.

[41] L. F. Sebastião. "Vamos todos ser portageiros". Em: *Público* (2011). ISSN: 0872-1548. URL: https://www.publico.pt/2011/03/22/jornal/vamos-todos-ser-portageiros-21609735#gs.HXGGktcK.

[42] A. E. Solutions. "Milan/Paylink product description manual". Em: United Kingdom, UK, 2019. URL: http://www.aardvark.eu.com/downloads/brochures/milanbrochure.pdf.

[43] H. W.E.U. P. Specialists. "Flexible, Multi-function Coin Hopper". Em: *X5 Flexi Coin Hopper*. 2019. URL: https://www.hweurope.com/docs/coin/hwe-x5hopper-brochure.pdf.

[44] *SUH - Serial Universal Hopper Technical Manual*. 2019. URL: https://www.eurocoin.co.uk/universal-hopper-manual-serial.

[45] VendSoft. *MDB Protocol for Simplifying Your Vending Business*. 2016. URL: https://www.vendsoft.com/mdb-vending-machine-protocol.

[46] *Via Verde*. 2007. URL: https://pt.wikipedia.org/wiki/Via_Verde.