UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

REGISTRADOR DE EVENTOS SOBRE UMA PLATAFORMA COM MICROCONTROLADOR PARALLAX PROPELLER E INTERFACE WEB.

Autor: Valter Gabriel Paes Santiago

Orientador: Prof. Dr. Evandro L. L. Rodrigues

São Carlos

2015

Valter Gabriel Paes Santiago

REGISTRADOR DE EVENTOS SOBRE UMA PLATAFORMA COM MICROCONTROLADOR PARALLAX PROPELLER E INTERFACE WEB.

Trabalho de Conclusão de Curso apresentado
à Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia de Computação

ORIENTADOR: Prof. Dr. Evandro L. L. Rodrigues

São Carlos
2015

Santiago, Valter Gabriel Paes
S194r REGISTRADOR DE EVENTOS SOBRE UMA PLATAFORMA COM MICROCONTROLADOR PARALLAX PROPELLER E INTERFACE WEB / Valter Gabriel Paes Santiago; orientador Evandro Luis Linhari Rodrigues. São Carlos, 2015.

Monografia (Graduação em Engenharia de Computação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2015.

1. Controle de Acesso. 2. SPIN. 3. Parallax Propeller. I. Título.

FOLHA DE APROVAÇÃO

Nome: Valter Gabriel Paes Santiago

Título: "Controle de acesso a um ambiente sobre uma plataforma com microcontrolador, RFID, cartão SD e interface WEB"

Trabalho de Conclusão de Curso defendido em 27/11/2015.

Comissão Julgadora:	**Resultado:**
Prof. Associado Evandro Luís Linhari Rodrigues (Orientador) - SEL/EESC/USP	Aprovado
Prof. Dr. Marcelo Andrade da Costa Vieira SEL/EESC/USP	Aprovado
Mestre Alex Antonio Affonso Doutorando - SEL/EESC/USP	Aprovado

Coordenador do Curso Interunidades Engenharia de Computação pela EESC:

Prof. Dr. Maximiliam Luppe

Dedicatória

Dedico este trabalho à minha família. Meu pai, minha mãe e meu irmão são pessoas que jamais deixaram de me apoiar, mesmo em frente à maior dificuldade que eu estivesse enfrentando. Tê-los como exemplo me faz crescer como pessoa, como adulto que sou, e só posso desejar que eu consiga transmitir aos meus filhos a mesma consciência do mundo que eles me apresentaram.

Valter Gabriel Paes Santiago.

Agradecimentos

Gostaria de agradecer à minha família e amigos pelo apoio. Gostaria, também, de agradecer especialmente à Jéssica, minha namorada, companheira insubstituível de todas as horas, por seu apoio e sua persistência indispensável comigo diante das minhas dificuldades. Por último e não menos importante, fica meu agradecimento a todos os professores que contribuíram para a minha graduação, principalmente ao professor e orientador Evandro. Durante o período da Universidade é muito difícil dar o devido valor a tudo que nos é ensinado, porém, logo que começa nosso caminho pelo mercado de trabalho, percebemos o quanto é importante todo esse conhecimento e passamos a sentir um orgulho enorme de fazer parte desta escola.

Valter Gabriel Paes Santiago.

"Só é lutador quem sabe lutar consigo mesmo."

Carlos Drummond de Andrade

Resumo

Neste trabalho foi desenvolvida uma plataforma que realiza o registro de acesso a um ambiente utilizando um microcontrolador *Parallax Propeller* juntamente com um sensor RFID, um cartão SD para o armazenamento de arquivos e uma interface *Web* para a recuperação dos dados armazenados no sistema remotamente. Este trabalho também contém um estudo sobre o microcontrolador *Parallax Propeller*, visto que possui uma arquitetura peculiar, com a presença de oito núcleos independentes e não possuir interrupções. Foram utilizadas as linguagens SPIN e linguagens *Web* em geral. O projeto funcionou como o esperado, possibilitando o registro e a recuperação tanto local como remotamente. Este trabalho também possibilita a familiarização com a arquitetura do microcontrolador usado, juntamente com sua linguagem nativa e seu ambiente de desenvolvimento, servindo como ferramenta para futuros desenvolvedores sobre esta plataforma.

Palavras-Chave: Controle de Acesso, *SPIN*, *Parallax Propeller*.

Abstract

In this work a platform that keeps records of access to an environment was developed using a microcontroller Parallax Propeller along with a RFID sensor, a SD card for storing files and a Web interface to remotely recover the data stored in the system. This work also contains a study of the microcontroller *Parallax Propeller*, since it has a peculiar architecture, with eight independent cores and no interruptions. SPIN and Web languages in general were used. The project worked as expected, enabling the recording and recovery both locally and remotely. This work also enables familiarization with the architecture of the microcontroller used, along with their native language and its development environment, serving as a tool for future developers on this platform.

Keywords: Acces Control, SPIN, Parallax Propeller.

Lista de Figuras

Siglas

Web	*Word Wide Web*
RFID	*Radio Frequence IDentification*
RTC	*Real Time Clock*
SPI	*Serial Peripheral Interface*
I2C	*Inter Integrated Circuit*
RAM	*Random Access Memory*
ROM	*Read Only Memory*
SD Card	*Secure Digital Card*
LCD	*Liquid Crystal Display*
MAC	*Media Access Control*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
SVN	*Subversion (version control system)*
PASM	*Propeller Assembly)*

Sumário

Capítulo 1

Introdução

Uma característica atual de praticamente qualquer ambiente formal, seja de trabalho ou de qualquer outra atividade, é de que exista alguma maneira de se ter controle sobre o fluxo de pessoas, individualizando cada uma delas, junto com seus respectivos horários de acesso e outros dados que sejam necessários.

Existem vários métodos para a auto-identificação como código de barras, reconhecimento óptico de caracteres, biometria ou cartões inteligentes. Além disso, existem os dispositivos RFID (Radio-Frequency IDentification- ou Identificador por Rádio Frequência), que são tratados como uma revolução tecnológica atual por terem condições de ser empregados nas mais diversas atividades a preços muito competitivos e garantindo segurança e facilidade de gerenciamento. Várias aplicações de RFID incluem: Transporte e logística, manufatura e processamento, segurança, animal *tagging* (registro e identificação posterior de animais por uma etiqueta subcutânea), gestão de resíduos, rastreamento postal, recuperação de bagagem aérea, gestão de pedágios, etc [5].

Atualmente, para manter pessoas não autorizadas sem acesso ao seu local físico, empresas têm implementado diversos sistemas utilizando RFID. Isso possibilita que essas empresas controlem o fluxo de funcionários, ao mesmo tempo que permitem ou restringem o acesso a determinados locais dentro da própria empresa, ou seja, determinam os locais onde cada trabalhador pode circular.

1.1 Objetivo

O objetivo deste trabalho é o desenvolvimento de uma plataforma de baixo custo capaz de registrar o acesso e o tempo de permanência de pessoas em uma sala utilizando um micro-

controlador *Parallax Propeller*.

Além da criação do mecanismo de controle, também é requisito do projeto uma implementação que permita a recuperação dos dados registrados, seja localmente, através do acesso ao Cartão SD, ou remotamente, através do acesso ao sistema via internet, tanto local como global.

1.2 Motivação

Este trabalho teve sua motivação em duas etapas diferentes descritas a seguir.

A primeira consiste no fato de que o aluno estava começando o estágio em uma empresa *StartUp*, com poucos colaboradores no momento porém com a expectativa de expansão. Com este cenário, fez-se necessário que um registro de acesso de pessoal e de horários fosse implementado. Como todo o material necessário estava disponível, o primeiro projeto de responsabilidade do aluno foi a criação do sistema proposto neste trabalho.

À princípio, o trabalho consistia apenas do acesso local aos dados armazenados no sistema, ou seja, não estava presente a interface *Web*. Posteriormente, foi incluída nesta segunda etapa esta funcionalidade para que os dados pudessem ser acessados remotamente, tornando o sistema mais abrangente e complexo.

Outra motivação é a implementação de um sistema num ambiente real que utilize o microcontrolador *Parallax Propeller* principalmente pela sua peculiaridade, pois, como não possui interrupções e possui vários núcleos, é possível responsabilizar sensores diferentes para núcleos diferentes, tendo uma abordagem bastante diferente do uso recorrente de interrupções em outros microcontroladores.

1.3 Organização do Trabalho

Este trabalho apresenta-se dividido da seguinte maneira:

- Capítulo 2 - Embasamento Teórico: apresentação dos conceitos e conhecimentos importantes necessários para o desenvolvimento deste trabalho;
- Capítulo 3 - Materiais e Métodos: descrição do projeto desenvolvido, com detalhamento de cada função e das aplicações;
- Capítulo 4 - Resultados: apresentação dos resultados obtidos com a utilização do sistema e casos de teste.

- Capítulo 5 - Conclusão: discussão final e validação dos objetivos propostos no início do trabalho.

Capítulo 2

Embasamento Teórico

O controlador de acesso é composto basicamente por duas partes principais. Uma plataforma onde está instalado o microcontrolador *Parallax Propeller* responsável por administrar os periféricos do sistema, como *Display* LCD, RTC, leitor RFID, etc. A outra parte consiste também de um microcontrolador *Parallax Propeller*, porém com a presença do controlador *Ethernet* W5100.

2.1 Microcontrolador *Parallax Propeller*

O microcontrolador utilizado neste sistema é o *Parallax Propeller* [6], apresentado na figura 2.1.

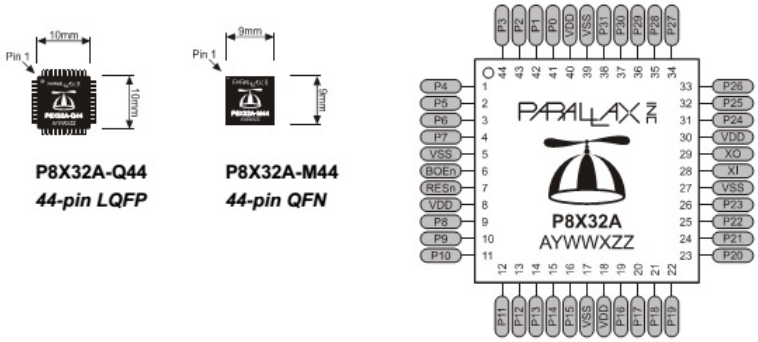

Figura 2.1: Microcontrolador *Parallax Propeller*[1].

Este microcontrolador, atualmente, é um concorrente direto da solução Arduino, tomando como base o preço, tendo um preço competitivo e sendo direcionado para as mais diversas aplicações.

Apesar dessa semelhança, outros aspectos deste microcontrolador o tornam único. Seu

conceito baseia-se em prover alta velocidade de processamento para sistemas embarcados enquanto mantém baixo consumo de energia. Além disso, por ter arquitetura de 32 *bits*, RISC, e contando com a presença de oito processadores, chamados aqui de *cogs*, pode realizar várias tarefas independentemente ou de maneira cooperativa, ainda assim mantendo a arquitetura simples e de uso praticamente intuitivo.

Outra característica bastante interessante do *Parallax Propeller* é que ele não possui interrupções, como grande parte dos outros microcontroladores. Sendo assim, para a realização de tarefas que seriam feitas através de interrupções é possível apenas direcionar um *cog* para que tome conta da tarefa enquanto os outros podem se manter realizando outras atividades ou mesmo desligados, economizando energia.

Existem 32 pinos de propósito geral (P0 até P31). A pinagem do microcontrolador é encontrada na figura 2.2, juntamente com a descrição de cada função respectivamente.

Table 1-1: Pin Descriptions

Pin Name	Direction	Description
P0 – P31	I/O	General purpose I/O Port A. Can source/sink 40 mA each at 3.3 VDC. Logic threshold is ≈ ½ VDD; 1.65 VDC @ 3.3 VDC. The pins shown below have a special purpose upon power-up/reset but are general purpose I/O afterwards. P28 - I^2C SCL connection to optional, external EEPROM. P29 - I^2C SDA connection to optional, external EEPROM. P30 - Serial Tx to host. P31 - Serial Rx from host.
VDD	---	3.3 volt power (2.7 – 3.3 VDC).
VSS	---	Ground.
BOEn	I	Brown Out Enable (active low). Must be connected to either VDD or VSS. If low, RESn becomes a weak output (delivering VDD through 5 KΩ) for monitoring purposes but can still be driven low to cause reset. If high, RESn is CMOS input with Schmitt Trigger.
RESn	I/O	Reset (active low). When low, resets the Propeller chip: all cogs disabled and I/O pins floating. Propeller restarts 50 ms after RESn transitions from low to high.
XI	I	Crystal Input. Can be connected to output of crystal/oscillator pack (with XO left disconnected), or to one leg of crystal (with XO connected to other leg of crystal or resonator) depending on CLK Register settings. No external resistors or capacitors are required.
XO	O	Crystal Output. Provides feedback for an external crystal, or may be left disconnected depending on CLK Register settings. No external resistors or capacitors are required.

Figura 2.2: Descrição da Pinagem do microcontrolador*Parallax Propeller*[1].

Um aspecto importante que pode ser observado neste microcontrolador é que ele trabalha em diferentes velocidades, variando de 5MHz a 80MHz, sendo possível administrar o

uso do processador e, consequentemente, o consumo de energia do sistema como um todo. Infelizmente não é possível determinar a velocidade de processamento em tempo de execução, porém, já é algo de extrema importância pois, conforme a complexidade da atividade, pode-se economizar muita energia com *clocks* menores.

Além disso, cada um dos processadores independentes possui uma memória *RAM* individual de 512 x 32 *bits*. O *chip* como um todo também possui uma memória compartilhada de 64KB, sendo 32KB de memória *RAM* e 32KB de memória *ROM*. Isso possibilita que diferentes *cogs* acessem a memória compartilhada para trocar informações ou fazer uso de um mesmo dado de maneira paralela. Este cenário está ilustrado nas figuras 2.3 e 2.4, juntamente com o diagrama de blocos do *chip* [1].

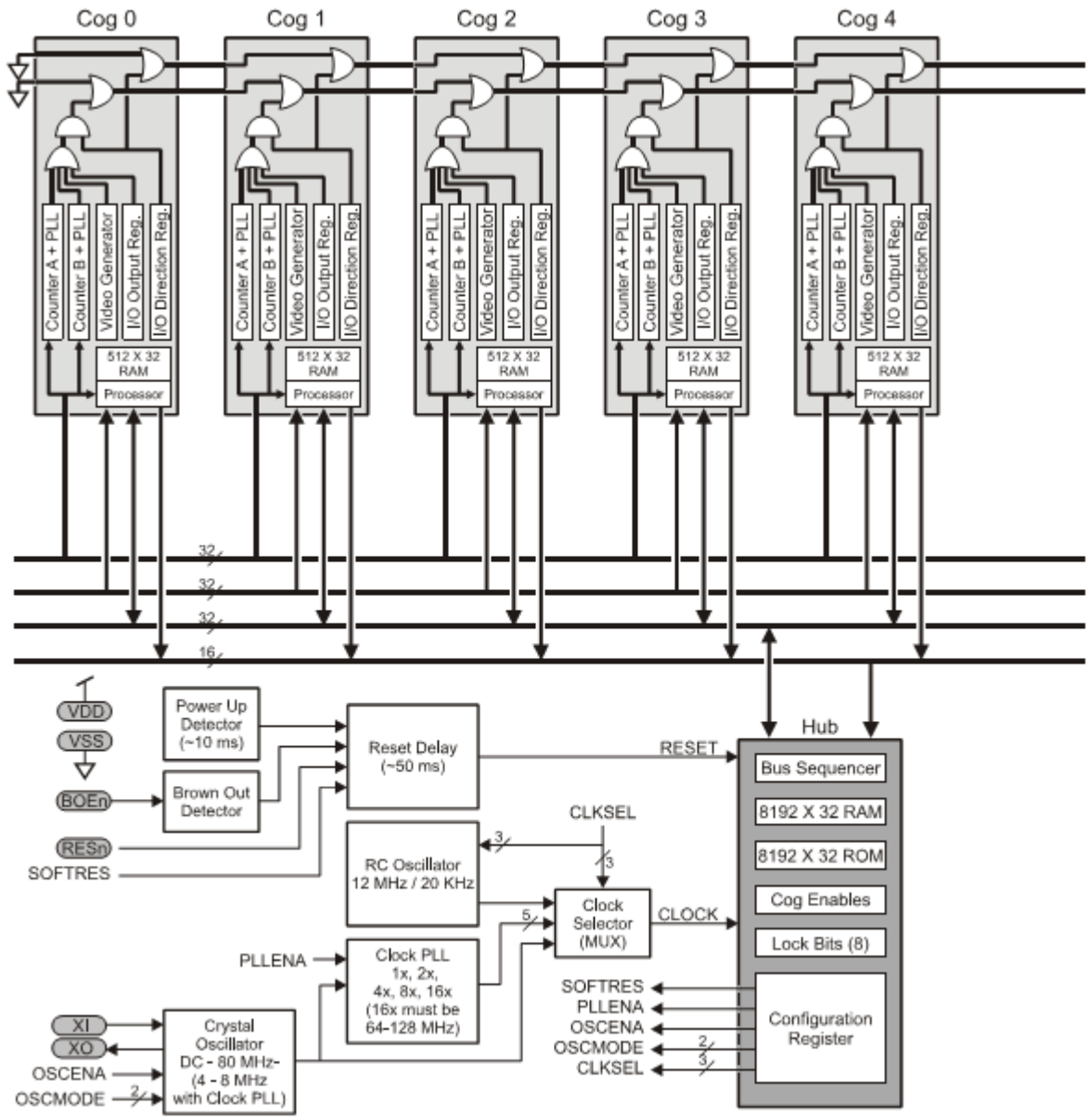

Figura 2.3: Diagrama de Blocos do *Propeller Chip* 1[1].

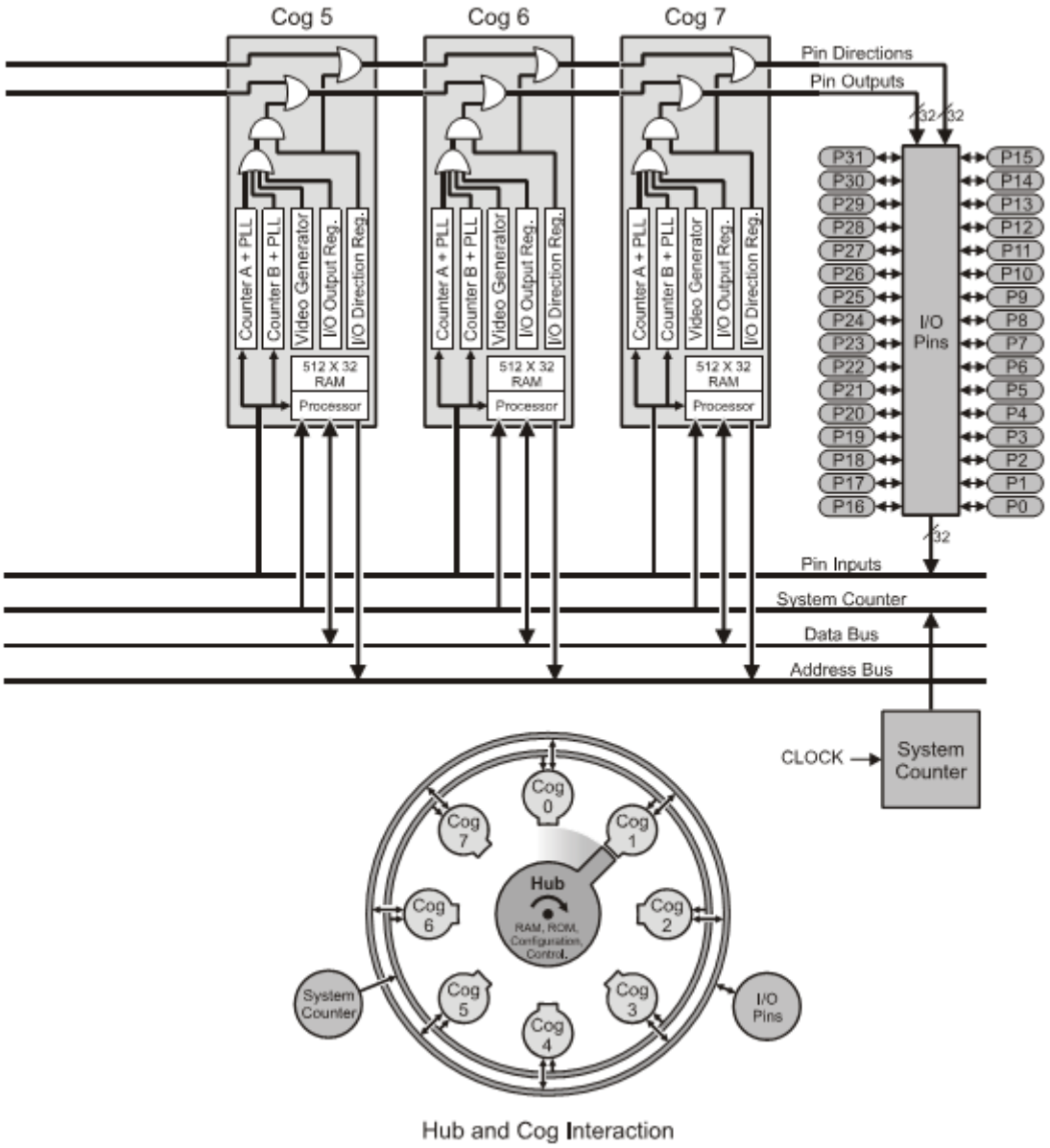

Figura 2.4: Diagrama de Blocos do *Propeller Chip* 2[1].

É importante salientar que o microcontrolador *Parallax Propeller* pode ser programado em três liguagens distintas. A primeira, e nativa, é a linguagem *SPIN*, que é utilizada neste trabalho e será apresentada na próxima seção. Além desta linguagem, também é possível programar em C, porém esta linguagem ainda não está completamente desenvolvida para ser aplicada junto ao microcontrolador. Sendo assim, a escolha pela primeira linguagem se tornou a mais sensata, tendo como meta que todo o sistema funcione com o mínimo de falhas. A última delas é a linguagem Assembly, que não foi utilizada por questões óbvias de complexidade e objetividade.

2.2 Linguagem *SPIN Propeller*

SPIN é uma linguagem interpretada de alto nível que se assemelha bastante a uma mistura da linguagem *BASIC* com *Python*. Logo de início se percebe a simplicidade desta linguagem, dividida em poucos blocos de programação, em que sistemas inteiros podem ser programados.

Um arquivo em linguagem *SPIN*, depois de salvo, pode ser utilizado em outros programas, de maneira análoga aos *includes* da linguagem C, ou aos *imports* do *Python*.

Estes blocos são:

- CON: Neste bloco são definidas as constantes globais, tal como a frequência do *clock* e outros aspectos do sistema;

- VAR: Aqui, as variáveis são definidas. Elas podem ter tamanhos diferentes como *byte*, *float*, *long*, *word*, etc. Também é aqui onde são declarados *arrays*, da mesma maneira como na linguagem C;

- PUB e PRI: Tanto nos blocos PUB como PRI se localizam os métodos, ou a programação em si. Todas as operações são realizadas nestes dois blocos. Eles se diferenciam quanto à privacidade. Blocos PUB podem ser acessados por outros

- OBJ: Neste bloco são declarados quais *objects* serão utilizados no programa. Como já foi exemplificado, este bloco funciona como um espaço de *includes* da linguagem C, ou aos *imports* do *Python*.

- DAT: O bloco DAT é usado para fazer *defines*, de maneira muito parecida como são feitos os *defines* da linguagem C. Outra funcionalidade deste bloco é que aqui também são armazenados os códigos em *Assembly* que podem ser utilizados pelo programa. Em geral, *drivers* de dispositivos de áudio ou vídeo utilizam programação em *Assembly* por uma questão exclusivamente de desempenho.

Na figura 2.5 se encontra um exemplo de como esses blocos podem ser conjugados para formar um arquivo *.spin*, que será compilado e transportado ao microcontrolador posteriormente. Este é um programa simples que transmite caracteres por interface Serial para um terminal no computador.

```
{{
    autor: Valter G P Santiago
}}

OBJ

    pst         : "Parallax Serial Terminal Plus"
    system      : "Propeller Board of Education"
    timing      : "Timing"
    sd          : "PropBOE MicroSD"

VAR

    byte serialWord[4]          'Word that will be sent from the terminal

PUB main | compass

    system.clock(80_000_000)

    pst.start(115_200)

    'sd.Mount(0)

    compass := random // 359
    serialWord [3] := 10

    repeat
        compass := compass + random // 4

        if compass > 359
            compass := compass - 360
        elseif compass < 0
            compass := compass + 360

        serialWord[0] := 48 + compass / 100 // 10
        serialWord[1] := 48 + compass / 10 // 10
        serialWord[2] := 48 + compass // 10

        pst.char(serialWord[0])
        pst.char(serialWord[1])
        pst.char(serialWord[2])
        pst.char(serialWord[3])

        timing.pause1ms(50)

PRI random | x

    x := cnt
    ?x

    return x
```

Figura 2.5: Exemplo da linguagem *SPIN*.

Como em muitas outras linguagens, *SPIN* tem uma quantidade grande de singularidades que não são objetos diretos de estudo deste trabalho e portanto não serão apresentadas aqui. Apenas pode-se deixar claro que esta é uma linguagem muito interessante para a programação de microcontroladores. Ela leva em consideração vários aspectos, como simplicidade e desempenho, fazendo com que o trabalho do dia a dia seja recompensado com uma grande satisfação, visto a fluidez e rapidez com que a adaptação a ela ocorre, além de permitir resolver problemas de maneira intuitiva, da mesma forma que outras linguagens de auto nível para plataformas *Desktop*.

2.3 RFID

RFID, acrônimo para *Radio-Frequency IDentification*, é uma tecnologia utilizada desde a Segunda Guerra Mundial, nos sistemas de radares de inúmeros países. Quando a única possibilidade de identificação de elementos distantes como aviões era o RADAR, foram implantados em aviões ingleses transmissores que davam respostas diferentes a este sistema, indicando-os como amigos. Deste modo, deu-se como implantado o primeiro sistema de identificação por rádio frequência [7].

Um sistema RFID possui basicamente de três componentes. Uma antena e um transceptor, que normalmente encontram-se no mesmo dispositivo, e um transponder ou uma etiqueta RF, que deverá conter a informação a ser transmitida. A antena usa ondas de RF para transmitir um sinal que ativa o transponder. Quando ativado, o transponder transmite dados de volta para o dispositivo de origem. Esta informação é usada para que alguma ação ocorra. Esta ação, no caso, pode ser simples como levantar uma cancela de um estacionamento ou muito complexa, como permitir que uma transação bancária ocorra.

Estes sistemas tem diversos alcances, também, dependendo da frequência de onda usada em cada um. Baixas frequências têm baixo alcance, em geral menos que um metro, porém altas frequências podem ter um alcance de até 50 metros. Apesar disso, baixas frequências são capazes de contornar objetos enquanto altas frequências não possuem a mesma capacidade.

Na figura 2.6 está um exemplo de uma etiqueta RFID, enquanto na figura 2.7 um sistema simples está ilustrado.

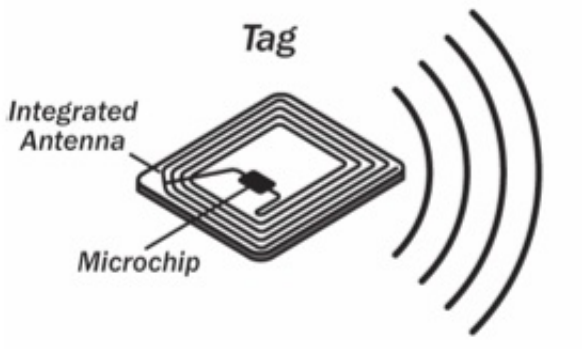

Figura 2.6: Etiqueta RFID[2].

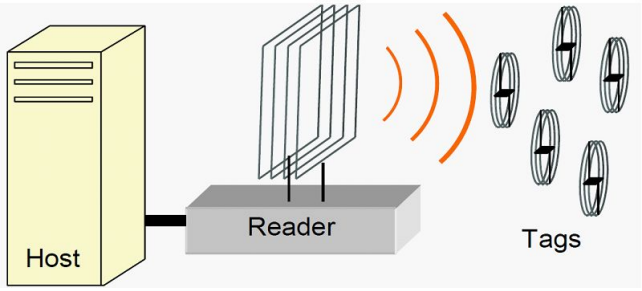

Figura 2.7: Sistema simples RFID[2].

Existem dois tipos de etiquetas RFID, são elas:

- Passiva: São energizadas pelo sinal de rádio emitido do leitor para transmitir a sua informação. Normalmente são gravadas permanentemente quando são fabricadas, como cartões de ônibus. Contudo, também podem ser regraváveis, como cartões de acesso em geral.

- Ativa: Estas etiquetas contam com uma alimentação própria para transmitir seu sinal, e, portanto, atingem uma distância consideravelmente maior, além de armazenar uma quantidade de informação muito maior.

Com uma abrangência tão grande deste sistema, nota-se sua presença nos mais variados cenários, desde radares, com alcance muito grande, até sistemas de pagamento via telefone celular, controle de estoque, substituição de códigos de barras, rastreamento de cargas ou de animais, identificação biométrica e assim por diante. Inclusive implantes em humanos já foram realizados, tornando-os capazes, por exemplo, de obter acesso a uma sala apenas pela passagem da mão por um leitor, ou mesmo de desbloquear o próprio celular sem a necessidade de um padrão ou uma senha [2].

2.4 *Spinneret Web*

Um dos *kits* mais interessantes contendo o microcontrolador *Parallax Propeller* certamente é o *Spinneret Web Server*. Neste conjunto, estão presentes um microcontrolador *Parallax Propeller*, juntamente com um controlador *Ethernet WIZnet* W5100, um *socket* para Cartão SD e um RTC com um capacitor de *backup*.

Nessas condições, é possível o desenvolvimento de uma plataforma web completa, visto que o controlador W5100 possui a implementação de toda a pilha *Ethernet*, enquanto o pa-

ralelismo do microcontrolador *Parallax Propeller* permite o desenvolvimento de um sistema embarcado para redes de aplicações robusto e confiável.

Na figura 2.8 é possível notar, também, que este *kit* é extremamente compacto, possuindo dimensões de 9,7 x 3,4 x 1,7 cm. Isso possibilita sua instalação nos mais diversos ambientes sem que sua presença seja notada de maneira desagradável.

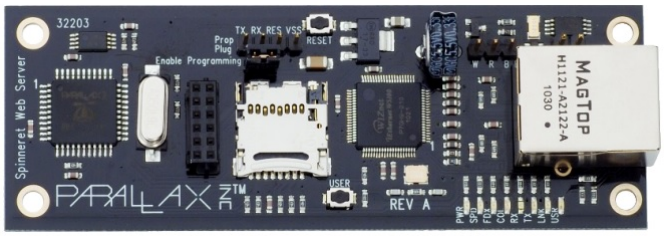

Figura 2.8: *Spinneret Web*[3].

Outra facilidade do *Spinneret Web* é que, junto com ele, vem uma câmera serial que pode ser facilmente instalada e com o *software* desta câmera incluído como exemplo. Apesar de não ter sido usada neste trabalho, poderia ser uma inclusão importante caso fosse necessário um registro fotográfico a cada evento ocorrido.

2.4.1 Controlador *Ethernet* W5100

Juntamente com o *Parallax Propeller*, o *Spinneret Web* possui um controlador *Ethernet WIZnet* W5100. Este é um controlador *Ethernet* compatível com internet, desenvolvido para sistemas embarcados onde facilidade de integração, estabilidade, performance e custo são fatores determinantes. Ele dispobiliza uma pilha TCP/IP funcional e bem implementada, integrado com MAC e PHY. Suporta TCP, UDP, IPv4, ICMP, ARP, IGMP e PPPoE e não necessita que se considere o controle *Ethernet* em si, permitindo que se programe apenas sobre o *socket*.

Para facilitar a integração, existem duas interfaces para conexão com este controlador. São elas, acesso direto pela memória, chamado simplificadamente de acesso direto, e acesso indireto por SPI [4]. Neste trabalho, o acesso indireto é utilizado pelo microcontrolador *Parallax Propeller* para se conectar ao *WIZnet* W5100.

Para sua ilustração, na figura 2.9 está representado o diagrama de blocos do controlador W5100, onde é possível notar sua interação tanto diretamente pela memória quanto indiretamente por SPI, juntamente com os protocolos implementados.

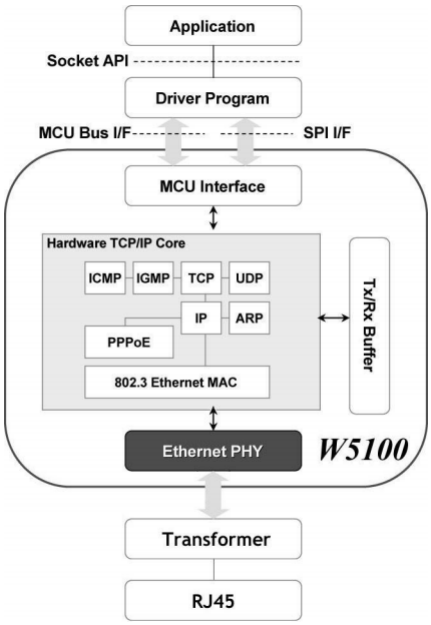

Figura 2.9: Diagrama de blocos do controlador W5100[4].

Capítulo 3

Materiais e Métodos

Na figura 3.1 a seguir, é mostrada a estrutura geral do projeto. O sentido das setas é análogo ao tratamento dado aos componentes, ou seja, se ele recebe uma entrada ou se envia algum dado ou instrução. Neste caso, é possível perceber que o LCD, por exemplo, apenas recebe informações do microcontrolador enquanto o leitor RFID manda informações para o mesmo. As funcionalidades do sistema estão apresentadas detalhadamente nas próximas seções deste capítulo.

O *hardware* do sistema de controle de acesso é formado inicialmente por uma caixa de plástico, onde estão instalados todos os componentes necessários para registrar entradas e saídas, horários e eventos. Estes dispositivos, como *Display LCD*, RTC, *Buzzer* e outros, estão conectados ao microcontrolador através dos seus pinos de propósito geral.

Além disso, é importante salientar que o sistema trabalha com um sistema de contagem regressiva de horas de trabalho. Isso significa que é estabelecido um parâmetro de tempo de trabalho diário e é feito o balanço dessas horas durante o mês. Ao final do mês, o tempo de trabalho que o trabalhador deixou de trabalhar, ou que trabalhou a mais é zerado e o balanço do mês fica armazenado no cartão SD. Outro aspecto importante é que o sistema só contabiliza o tempo trabalhado caso o trabalhador feche o ciclo, ou seja, que o número de entradas seja igual o número de saídas. Caso isso não aconteça, o balanço de horas do dia é zerado. Neste trabalho, o total de horas de trabalho diárias é de oito horas, seguindo o que é consenso de limite de tempo de trabalho diário.

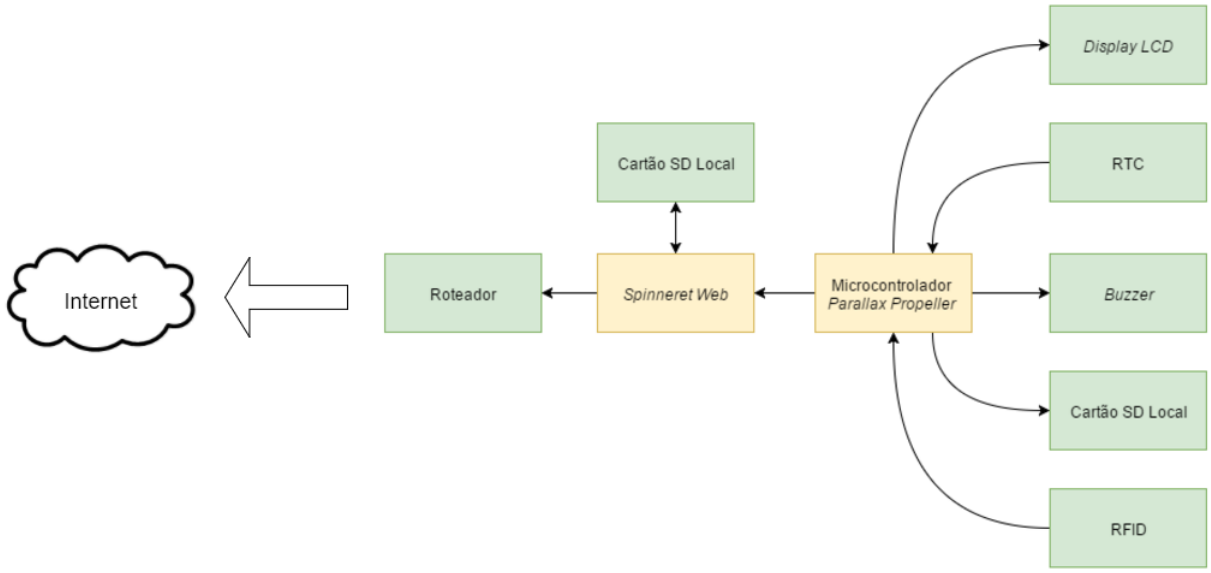

Figura 3.1: Arquitetura do sistema de controle de acesso.

3.1 Microcontrolador *Parallax Propeller*

Toda a programação deste microcontrolador foi realizada na linguagem *SPIN*, como descrito na seção 2.2. Aqui, todos os processos criados para integração de cada dispositivo ao sistema serão detalhadamente descritos. É importante destacar, ainda, que toda a criação do sistema foi feita sobre a IDE *Propeller Tool*, que será detalhada na subseção a seguir.

3.1.1 IDE *Propeller Tool*

A *IDE Propeller Tool* está na versão 1.3.2, de outubro de 2012 [8].

Desenvolvida especialmente para a programação para o microcontrolador *Parallax Propeller*, ela possui todos os aspectos necessários, tanto para programação quanto para a interação com o microcontrolador. Esta é uma IDE proprietária, mas que também disponibiliza uma versão *open source* para que alterações sejam feitas e *features* sejam incorporadas. Esta IDE é utilizada por ser a mais simples, visto que outros projetos de IDE para o microcontrolador foram desenvolvidas, porém, com uma complexidade desnecessária de uso. Fazem parte desta IDE:

- *Driver* USB: Utilizado para interação entre computador e microcontrolador, tanto para comunicação serial, quanto para fazer *upload* do arquivo compilado para que o microcontrolador realize alguma tarefa.

- *Propeller Manual*: Usado como referência para interação com o microcontrolador e com a linguagem SPIN.

- Esquemáticos: Ilustrando os esquemas de interação entre o microcontrolador e outros dispositivos que possam ser conectados e ele.

- Exemplos: Usados para que o usuário se habitue com a programação em SPIN e se sinta confortável para interagir com os mais diversos dispositivos. Inclusive tutoriais estão disponíveis para que o usuário se integre com facilidade à plataforma.

Além disso, um importante componente desta IDE é o *Parallax Serial Terminal* utilizado tanto para entrada e saída de dados feita entre o computador e o microcontrolador, como ferramente de *Debug* pois como o desenvolvimento desta ferramenta e mesmo deste microcontrolador ainda é recente, o grande revés desta IDE é a falta de um *Debugger* completo. Porém está prometido pelos desenvolvedores que esta ausência será suprida na segunda versão do *Parallax Propeller* [9], anunciada para ter seu lançamento em breve.

As figuras 3.3 e 3.2 são, respectivamente, a IDE *Propeller Tool* e o *Parallax Serial Terminal*.

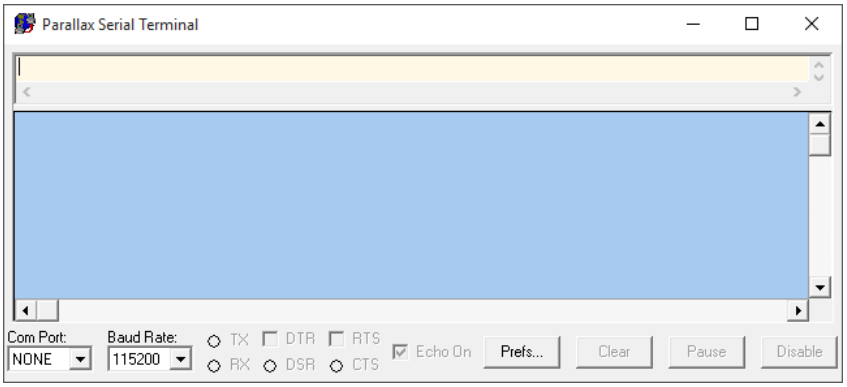

Figura 3.2: *Parallax Serial Terminal*.

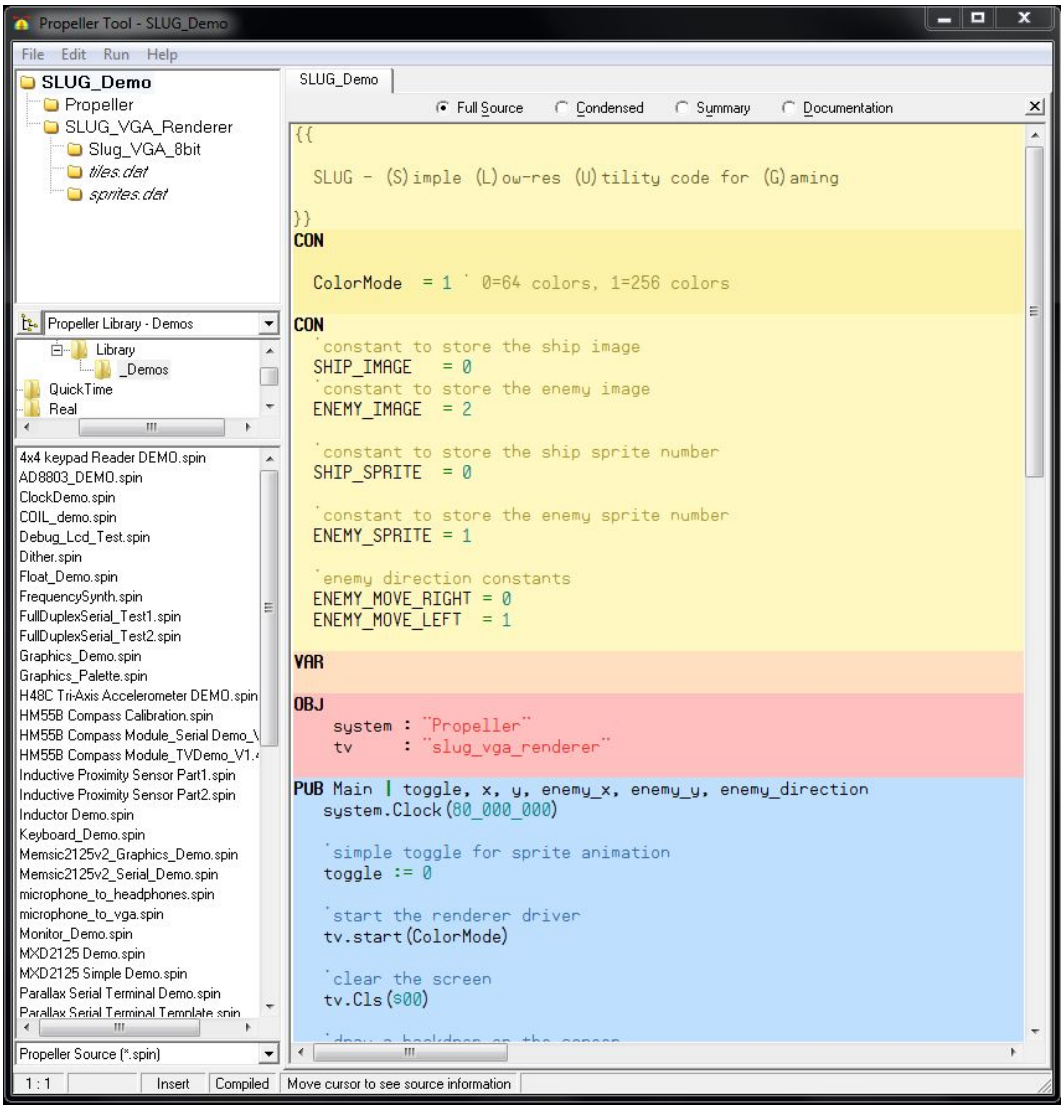

Figura 3.3: IDE *Propeller Tool*.

3.2 Cartão SD

O cartão SD é a base da informação deste controlador de acesso. Todos os dados de entrada, saída e horário registrados estão armazenados no cartão SD.

No sistema, o cartão SD é utilizado através de um driver FAT, que possui as operações básicas de gerenciamento de arquivo juntamente com operação para fazer *mount* e *unmout*. Ele está ligado ao microcontrolador através dos pinos 22 a 25, utilizando a interface SPI para realizar esta conexão. As operações principais utilizadas no cartão SD são as seguintes:

- *Mount*: Responsável por dar início ao processo de comunicação com o cartão SD. A partir desta operação, é possível realizar qualquer alteração nos arquivos existentes,

criar ou deletar arquivos;

- *Unmount*: Após a utilização do cartão SD, é recomendado fazer o *unmount*, para que as alterações realizadas sejam devidamente armazenadas, sem correr o risco de haver uma interrupção ou corrompimento de arquivo. Além disso, fazer *unmount* deixa bem localizado o fim de uma operação no cartão SD, sendo assim, uma boa prática de uso;
- *FileOpen*: Utilizada para abrir um arquivo no Cartão SD para edição. Neste caso, caso o arquivo não exista, ele será criado.
- *FileClose*: Utilizada para fechar um arquivo após a sua edição pelo sistema, normalmente após uma entrada ou saída de alguém do ambiente que se quer ter o controle de acesso.
- *Write*: Neste sistema, várias operações de gravação nos arquivos SD são realizadas. Entre elas, são gravados inteiros e caracteres. No *driver* são disponibilizadas operações de gravações dos mais diversos tipos de variáveis.
- *ListFiles*: É uma operação que faz a listagem dos arquivos no cartão SD. Esta operação é utilizada para fazer a operação de recuperação dos dados pela *Web*.

A gravação dos arquivos no cartão SD obedece algumas regras listadas a seguir:

- Novos arquivos são criados a cada mês com as iniciais em maiúsculo do nome das pessoas inseridas previamente no sistema, seguido do mês e do ano. Esses arquivos são do formato .txt;
- A cada evendo no sistema, entrada ou saída, é gravado o horário do evento com os parâmetros de hora, minuto e segundo. Considera-se que sempre depois de uma entrada existe uma saída, sendo impossível ter duas entradas seguidas no sistema;
- No primeiro evento de entrada do dia, também é gravada a data no início do cartão;
- Caso uma pessoa não encerre seu dia com uma saída, um asterisco (*) será gravado ao final da linha, indicando que a contagem de horas de trabalho do dia foi anulada.

Sendo assim, o formato final dos arquivos gravados no cartão SD é "NN_MM_YY.txt", onde:

- NN são as iniciais do nome das pessoas cadastradas;

- MM é o número do mês do ano;

- YY são os dois dígitos menos significativos do ano;

Além do nome do arquivo, o conteúdo interno de cada arquivo possui um cabeçalho, gravado também no momento em que o arquivo é criado, e nas linha seguintes os eventos de entrada e saída registrados. Este cabeçalho está mostrado a seguir:

#, date , start , timeOut, ... ,dayBalance ,totalBalance

Onde:

- "#": Índice da entrada do mês;

- "*date*": Data do dia;

- "*start, timeOut*: Repetidos várias vezes, indicam os horários de entrada e saída da sala;

- "*dayBalance*": Número que indica o total de horas trabalhadas no dia, este número é gravado em minutos;

- "*totalBalance*": Número que indica o balanço total do mês até aquele momento. Caso este número seja positivo, o trabalhador trabalhou mais do que o tempo de trabalho determinado. Caso seja negativo, o trabalhador está em débito de horas. Este número também é gravado em minutos.

3.3 *Display* LCD

O LCD utilizado neste sistema é um *Display* com apenas duas linhas, com 16 espaços em cada linha. Ele possui uma conexão paralela com o microcontrolador. O *driver* disponibilizado pelo fabricante é totalmente compatível com o microcontrolador, tornando simples a sua utilização, tendo de considerar apenas os aspectos lógicos quando se deseja apresentar uma mensagem.

Neste trabalho, o *Display* LCD deve mostrar uma mensagem de recepção na linha de cima, seguida pelo dia e pela hora na parte de baixo, como mostra a figura 3.4

Figura 3.4: *Display* LCD.

A cada evento do sistema, o *display* saúda a entrada ou a saída, seguido pelo nome da pessoa com o cartão correspondente, e mostra, posteriormente, o balanço total do mês, convertido para horas e minutos para tornar o controle pessoal mais intuitivo, seguindo a regra do sinal para horas positivas ou negativas.

3.4 *Buzzer*

A cada evento realizado no sistema, um som de confirmação é emitido. O responsável por este som é um *buzzer* simples, que emite um som na frequência de 2349,3 Hz, correspondente ao sétimo Ré de um piano.

Para que este *buzzer* seja ativado, um contador no microcontrolador é configurado para alternar o pino 10 do sistema, onde ele está conectado. Este som persiste por 50 milissegundos.

Para melhor compreensão, esta função está ilustrada na figura 3.5.

```
PRI buzzerConfirmation

    'Configuring the Counter A
    ctra[30..26] := %00100
    ctra[5..0] := buzzer
    frqa := 126_130
    'frqa := 2018080

    'Broadcast the signal for 50ms, system clock divided by 20 (clkfreq/20)
    dira[buzzer] := 1
    waitcnt((clkfreq/20)+cnt)
    dira[buzzer] := 0
```

Figura 3.5: Função responsável pelo *buzzer*.

3.5 RTC

Este sistema possui um *chip* separado para a contagem do tempo, que não é feita diretamente pelo microcontrolador *Parallax Propeller*. Neste caso, o *chip* utilizado é o DS1302,

que provê a contagem do tempo e possui total compatibilidade com o microcontrolador principal.

Ele é conectado ao microcontrolador através dos pinos 19 ao 21 e suas funções de acesso, consulta ao tempo e configuração de data e hora são totalmente providos pelo *driver* disponibilizado pelo fabricante.

As principais funções do RTC utilizadas neste trabalho são as seguintes:

- *config*: Responsável por colocar o RTC em modo de configuração;
- *setDatetime*: Responsável por determinar o dia e a hora, passados por parâmetro;
- *readTime*: que permite a recuperação da hora juntamente com os parâmetros de minutos e segundos;
- *readDate*: que permite a leitura da data em anos, mês e dia;

3.6 Leitor MFRC522 Mifare RFID

O leitor de RFID utilizado neste trabalho é o MFRC522 Mifare que opera na frequência de 13,56 MHz, por ser de baixo custo e já tradicional em várias plataformas que utilizam Arduino [10]. Este leitor é conectado ao microcontrolador através de uma interface SPI. Os pinos correspondentes são o 1, 2, 4, 5 e 7.

Na figura 3.6 é possível ver o leitor RFID MFRC522 conectado ao sistema. Para que uma ação seja feita, é necessário que se aproxime deste leitor uma etiqueta RFID que esteja previamente cadastrada no sistema. Quando isso acontece, o *buzzer* é acionado, desencadeando um evento de entrada ou saída.

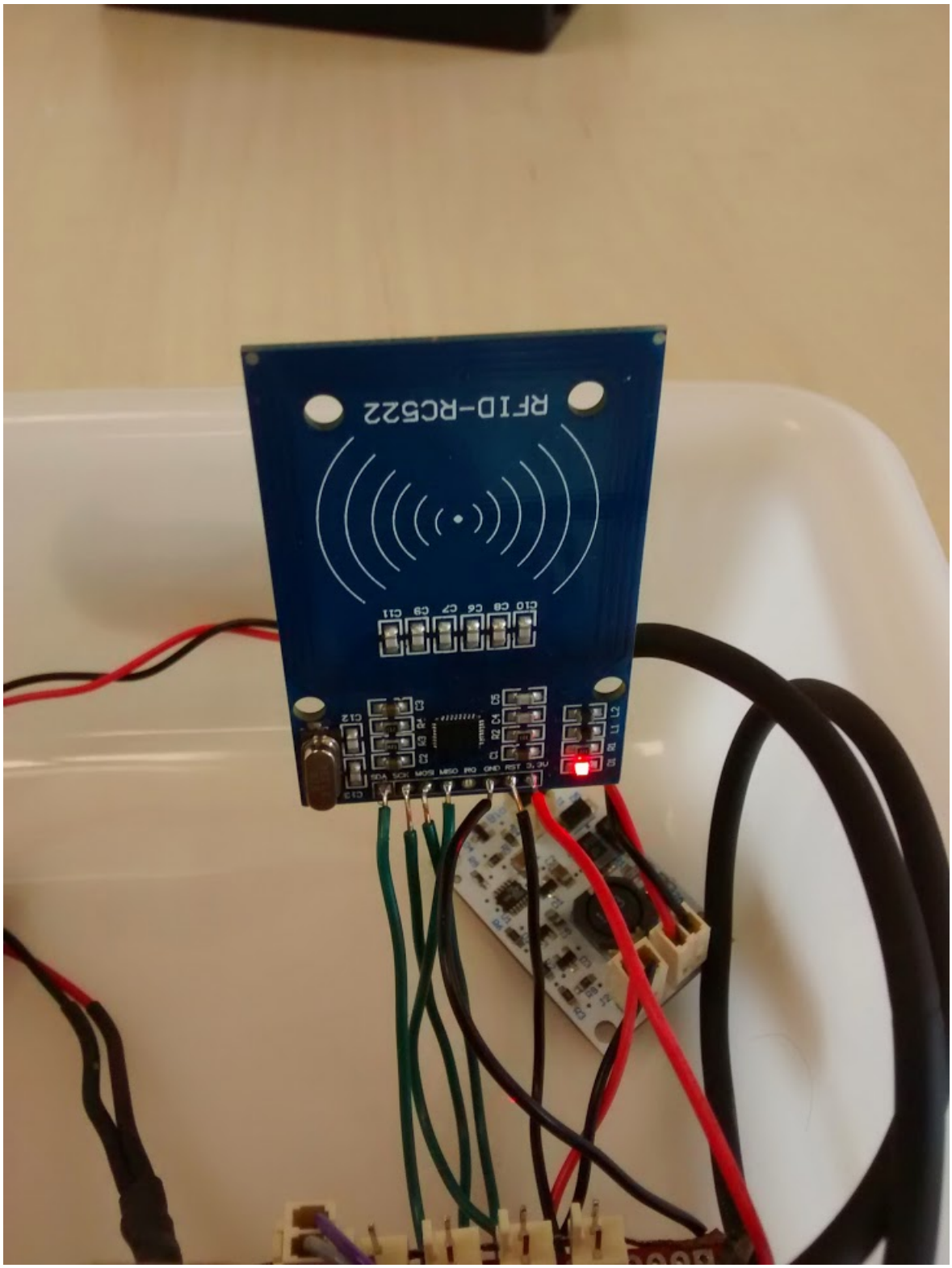

Figura 3.6: Função responsável pelo *buzzer*.

Nas figuras 3.7 e 3.8 estão ilustradas dois dispositivos de leitura RFID reconhecidos pelo leitor MFRC522. A primeira é um cartão comum, enquanto a segunda é uma *tag* RFID

estrategicamente posicionada em volta da câmera de um celular, permitindo que o portador realize uma ação de entrada ou saída no sistema simplesmente aproximando o próprio celular do leitor RFID.

Figura 3.7: Cartão com Etiqueta RFID.

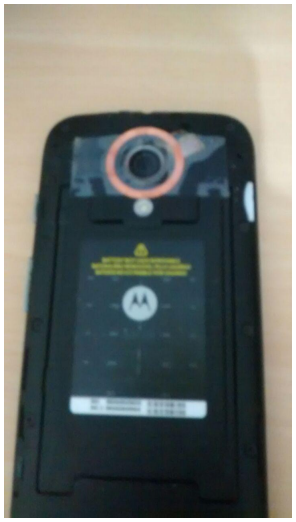

Figura 3.8: Celular com Etiqueta RFID posicionada em volta da câmera.

É importante deixar claro que caso um dispositivo não cadastrado se aproxime do sistema, ele não desencadeará nenhuma ação no controlador de acesso, ou seja, é sempre necessário um cadastro prévio para que um dispositivo interaja com o sistema.

Na figura 3.9, está ilustrado como os dispositivos estão cadastrados no sistema, através de

um bloco DAT da linguagem SPIN. É interessante notar que até mesmo um cartão de ônibus comum pode ser lido através do leitor MFRC522, porém, não é possível nenhuma mudança neste tipo de cartão, apenas a leitura do seu *Serial Number*.

```
DAT
'' Add serial numbers from previouly read cards.

serialNumber            long $7E208623          '' Ricardo
                        long $972A20A3          '' Ronaldo
                        long $09899B1A          '' ValterKeyRing
                        long $F7D51FA3          '' ValterCard
                        long $21C2C1AF          '' ValterBusCard
                        long $235331A2          '' Ana
                        long END_OF_SERIAL_NUMBER_LIST
```

Figura 3.9: Bloco DAT em SPIN de cadastro dos dispositivos lidos pelo sistema.

3.7 Principais Funções do Sistema

Para a ilustração das principais funções do sistema, alguns fluxogramas foram desenvolvidos. Eles existem desde antes da implementação deste sistema, servindo como base para o desenvolvimento. Apesar de simples, horas foram despendidas enquanto eles eram desenvolvidos e torna-se muito importante atentar para o fato de que muito tempo acabou sendo economizado, pois, na fase de projeto dessas funções, a maioria dos problemas de lógica de programação foram eliminados, sobrando apenas os problemas locais de implementação de cada uma das funções descritas.

Sendo assim, estes fluxogramas estão presentes nas figuras seguintes, juntamente com a explicação de cada uma dessas funções.

3.7.1 Função *main*

Esta função ilustra a base do sistema. Aqui, as funções mais abrangentes foram representadas. Elas ilustram todo o funcionamento do sistema tendo como base um panorama totalmente abstrato. Aqui está presente o *loop* principal do programa.

Sendo assim, em suma, o programa é composto por uma função que inicializa todos os outros dispositivos do sistema, se mantém em *stand by* e checa se algum dispositivo cadastrado foi aproximado do leitor RFID. Este procedimento está representado na figura 3.10.

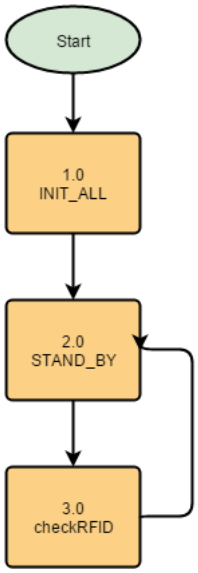

Figura 3.10: Função principal do sistema.

3.7.2 Função *INIT_ALL*

Esta função inicializa os diversos dispositivos que trabalharão juntamente com o microcontrolador. São eles o *display* LCD, o RTC e o *driver* do Cartão SD.

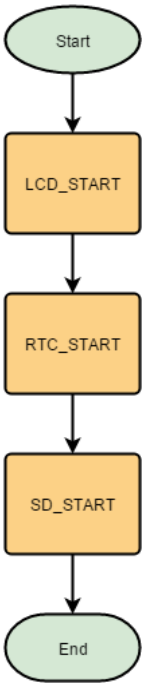

Figura 3.11: Função *INIT_ALL*.

3.7.3 Função *STAND_BY*

Esta função tem como principal atividade atualizar o *display* LCD do sistema, alterando-o a cada segundo. Além disso, esta é a função que faz a checagem de um novo dia e de um novo mês, que terão tratamento demonstrado a seguir.

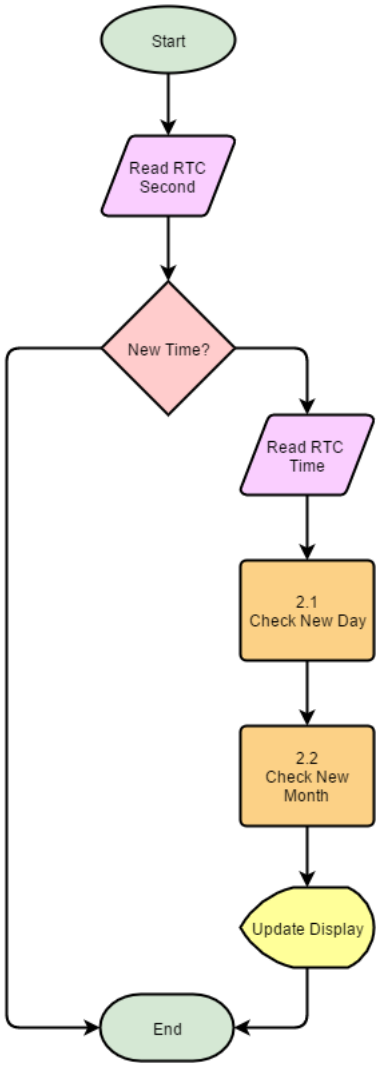

Figura 3.12: Função *STAND_BY*.

3.7.4 Função *checkNewDay*

A função *checkNewDay* é responsável por determinar de o dia é um dia de trabalho e, consequentemente salvar o balanço diário e total no cartão SD, além de fechar um ciclo de trabalho e preparar para o próximo dia. Um dia de trabalho é considerado para todos os trabalhadores cadastrados no sistema quando pelo menos uma pessoa dá entrada no sistema, ou seja, se uma pessoa entrar na sala, o dia é considerado como um dia de trabalho e caso

alguém não compareça, terá suas horas computadas como horas de trabalho a cumprir.

Além disso, é interessante deixar claro que um dia de trabalho é considerado como válido quando o número de entradas e saídas de uma pessoa é o mesmo, ou seja, a pessoa entrou e saiu do ambiente de trabalho. Caso a pessoa não tenha dado saída do sistema, o dia será invalidado e as horas não serão contadas no final do balanço.

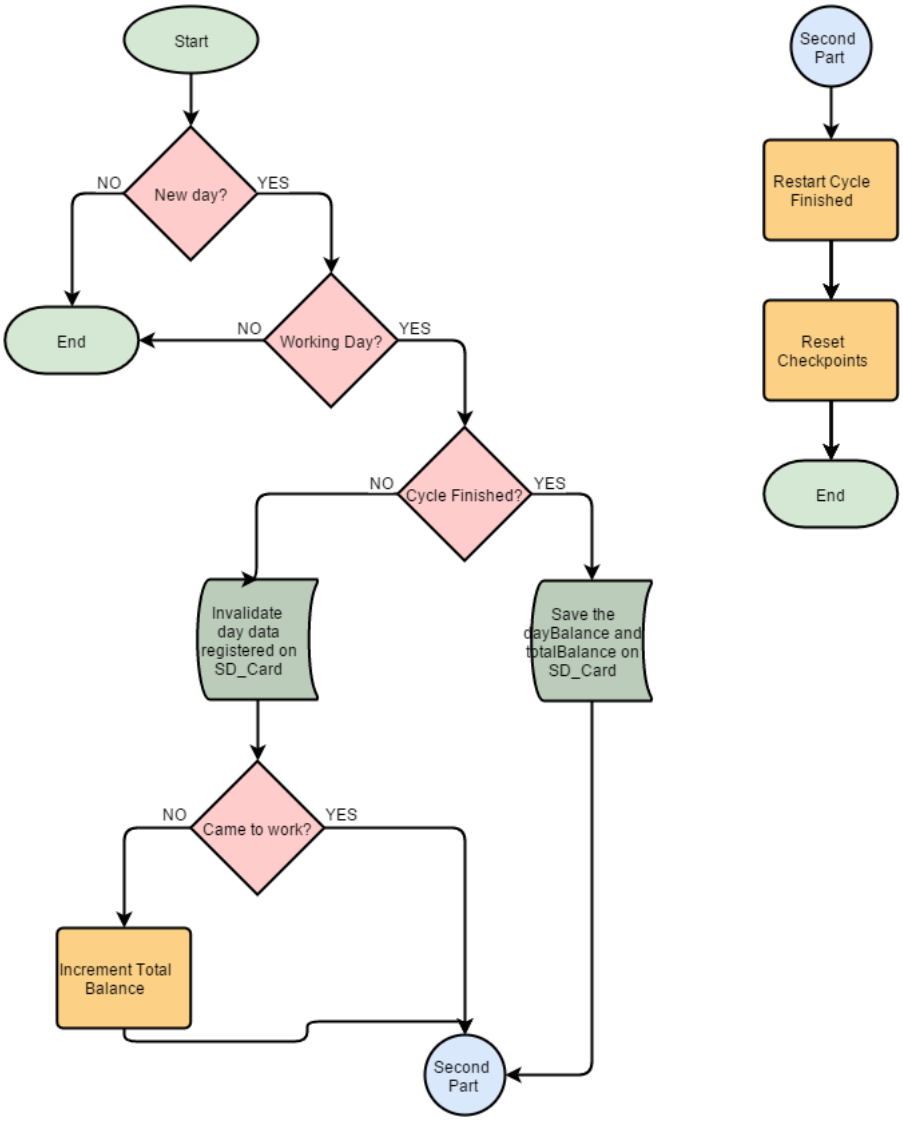

Figura 3.13: Função *checkNewDay*.

3.7.5 Função *checkNewMonth*

Esta função cria os novos arquivos no cartão SD correspondentes aos eventos registrados por cada usuário. Além disso, ela também zera os balanços, deixando o saldo do trabalhador gravado no último registro do mês anterior, conforme mostrado na seção 3.2.

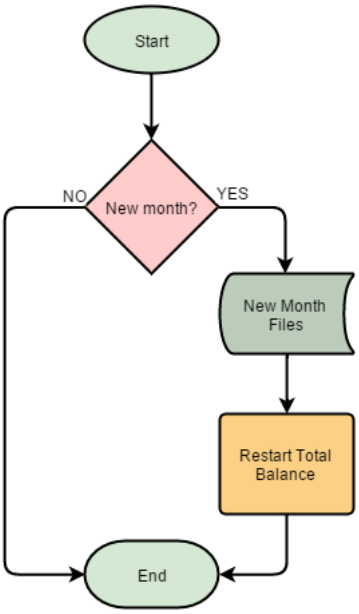

Figura 3.14: Função *checkNewMonth*.

3.7.6 Função *checkRFID*

Esta é a função responsável por determinar se houve algum evento relacionado com a leitura de um dispositivo RFID cadastrado. Caso isso seja detectado, a função RFID_ACTION é chamada.

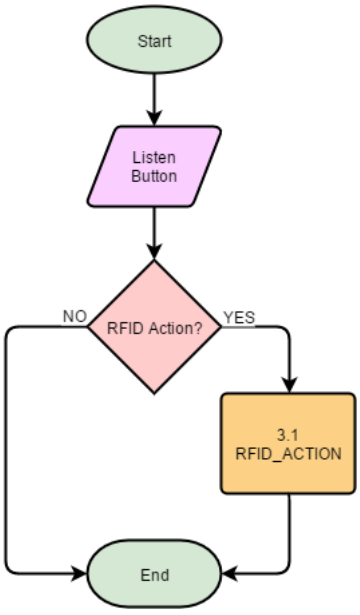

Figura 3.15: Função *checkRFID*.

3.7.7 Função *RFID_ACTION*

A função RFID_ACTION é certamente a mais importante do sistema, pois chama as funções de controle de tempo e realiza a gravação de dados no cartão SD. De acordo com a figura 3.16, quando um evento é registrado pelo sistema, primeiramente o *buzzer* é acionado, para que seja claro que o sistema está realizando uma ação. Posteriormente, uma mensagem é mostrada no *diaplay* LCD com uma saudação direcionada ao dono do dispositivo. Em seguida, o balanço total é calculado e o dado é gravado no cartão SD, seguido de uma confirmação no *display* LCD contendo o balanço atual de horas do trabalhador.

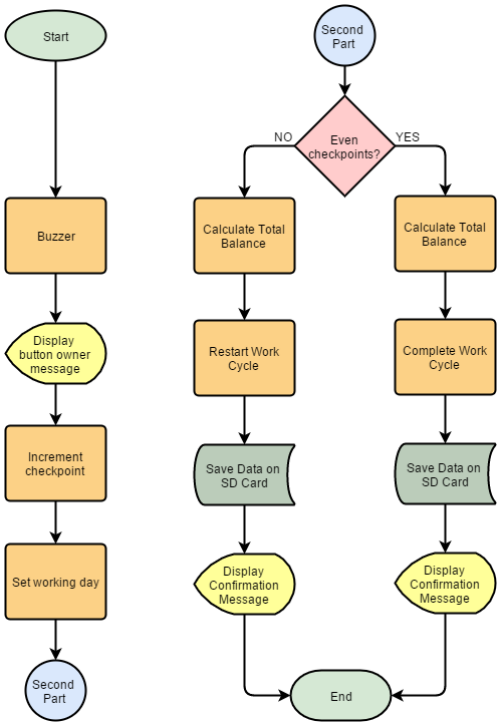

Figura 3.16: Função *RFID_ACTION*.

3.8 Acesso *Web* aos dados do cartão SD

Os dados do cartão SD são mostrados na tela do navegador através de uma página HTML gerada pelo próprio microcontrolador, utilizando o *driver* do cartão SD juntamente com a operações disponibilizadas pelo fabricante do chip W5100 para o *Parallax Propeller*.

Para que isso aconteça, primeiramente é mostrada uma tela de login, onde o usuário e a senha são necessários. Esta é uma tela de *login* simples utilizando o método *"get"*. Assim, a informação é confrontada com o usuário e senha necessários para o acesso ao sistema e caso a informação seja correspondente, o usuário tem a opção de baixar os arquivos presentes no

cartão SD. Por uma questão de simplicidade, esta é a única função presente neste trabalho, permitindo apenas a recuperação dos arquivos do cartão SD.

O *driver* do controlador W5100 possui uma maneira especial de tratar um método *get*, mostrado na figura 3.17 diferentemente de um sistema *desktop* comum. Assim, na linguagem SPIN e fazendo o uso deste *driver*, é necessário que este método seja chamado e tratado dentro da linguagem SPIN, e não diretamente em um arquivo HTML ou PHP, como seria o mais usual.

```
PRI DoGetResponse(fileName) | fileSize

    PST.Str(string("GET Response with file: "))
    PST.Str(fileName)
    PST.Str(string(PST#NL))

    fileSize := DoHeadResponse(fileName, 0)

    ' fileSize will be -1 for not found files, and we don't need to do
    ' otherwise now we send the file
    if fileSize <> -1
      SDCard.openFile(fileName)
      if fileSize < maxPacketSendSize
        ' send the file in one packet
        PST.Str(string("Sending small file.", PST#NL))
        SDCard.readFromFile(@data[0], fileSize)
        W5100.txTCP(0, @data[0], fileSize)
      else
        ' send the file in a bunch of packets
        PST.Str(string("Sending large file"))
        repeat
          SDCard.readFromFile(@data[0], maxPacketSendSize)
          W5100.txTCP(0, @data[0], maxPacketSendSize)
          PST.Str(string("."))
          fileSize -= maxPacketSendSize
          ' once the remaining fileSize is less then the max packet si
          if fileSize < maxPacketSendSize and fileSize > 0
            SDCard.readFromFile(@data[0], fileSize)
            W5100.txTCP(0, @data[0], fileSize)
            PST.Str(string("."))
            quit
      PST.Str(string("Done.",PST#NL))

      SDCard.closeFile
```

Figura 3.17: Função *get* implementada na linguagem SPIN.

Por questão de simplicidade, neste caso, é possível baixar todos os arquivos contidos no cartão SD, inclusive os HTML's que dão origem à própria página, por estarem também presentes na raiz do cartão SD.

3.9 *SVN*

Durante a programação deste projeto, e até mesmo a realização desta monografia, a ferramente SVN esteve presente [11].

Sendo assim, todas as versões e todos os momentos onde estes projetos foram realizados estão guardados como *snapshots*, ou seja, estágios intermediários de desenvolvimento.

Esta ferramenta permite o versionamento de *softwares* ou qualquer produção de texto

relacionada a alguma linguagem de computação, como no caso do LaTeX na realização desta monografia, auxiliando de maneira excepcional o planejamento e a elaboração de um trabalho em computação.

Capítulo 4

Resultados e Discussão

Nesta seção são apresentados os resultados obtidos após a implementação do sistema.

Primeiramente, na figura 4.1 é mostrado o sistema de frente, funcionando com todos os recursos. É possível destacar na imagem o *display* LCD e o leitor RFID, que devem estar à frente para que possam realizar suas funções.

Figura 4.1: Visão frontal do controle de acesso.

Já na figura 4.2 é mostrado o sistema instalado, logo ao lado da porta de entrada da sala onde se quer controlar o acesso. Sendo assim, logo ao entrar na sala o dispositivo RFID deve ser aproximado do leitor para que um evento seja realizado, da mesma maneira que deve ser feito ao sair.

Figura 4.2: Visão do local onde o controle de acesso foi instalado.

4.1 Cartão SD - Sistema Local

Localizados diretamente no controle de acesso, os arquivos do controle de acesso estão mostrados na figura 4.3. Ficam claros nesta imagem os aspectos descritos na seção 3.2, referentes às inciais dos nomes cadastrados no sistema e ao período registrado.

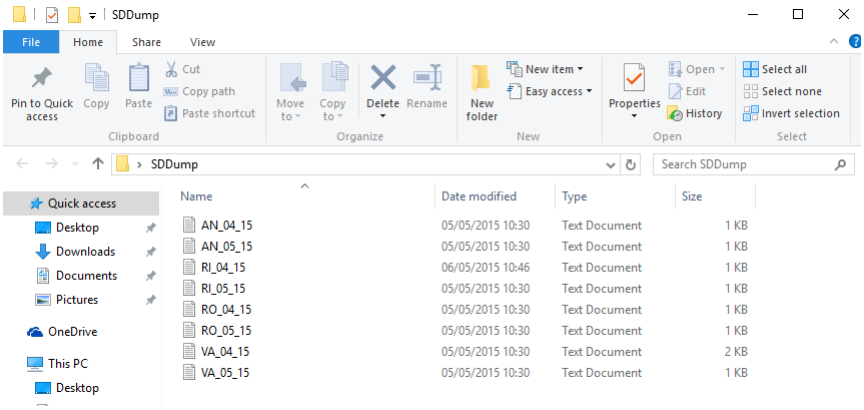

Figura 4.3: Arquivos armazenados localmente no sistema.

Além disso, o conteúdo dos arquivos está presente na figura 4.4. Aqui é possível notar o balanço de horas cadastrado, além de ilustrar quatro dias em que não foi dada a saída do sistema, tornando-os inválido. É importante notar, também, a presença do cabeçalho indicando o significado de cada posição registrada no cartão.

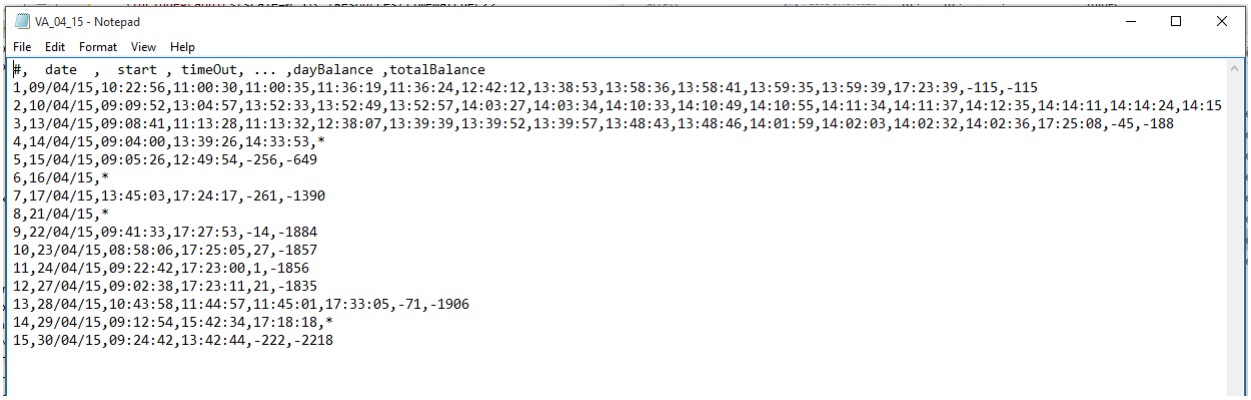

Figura 4.4: Conteúdo de um arquivo no cartão SD.

4.2 Plataforma *Web*

Para a plataforma *Web* existe primeiramente uma tela de login, para depois ter acesso aos arquivos gravados pelo sistema, juntamente com os arquivos HTML das páginas presentes. As imagens 4.5 e 4.6 mostram a tela de login e, posteriormente, os arquivos presentes no cartão SD.

Figura 4.5: Tela de *login* do sistema de controle de acesso.

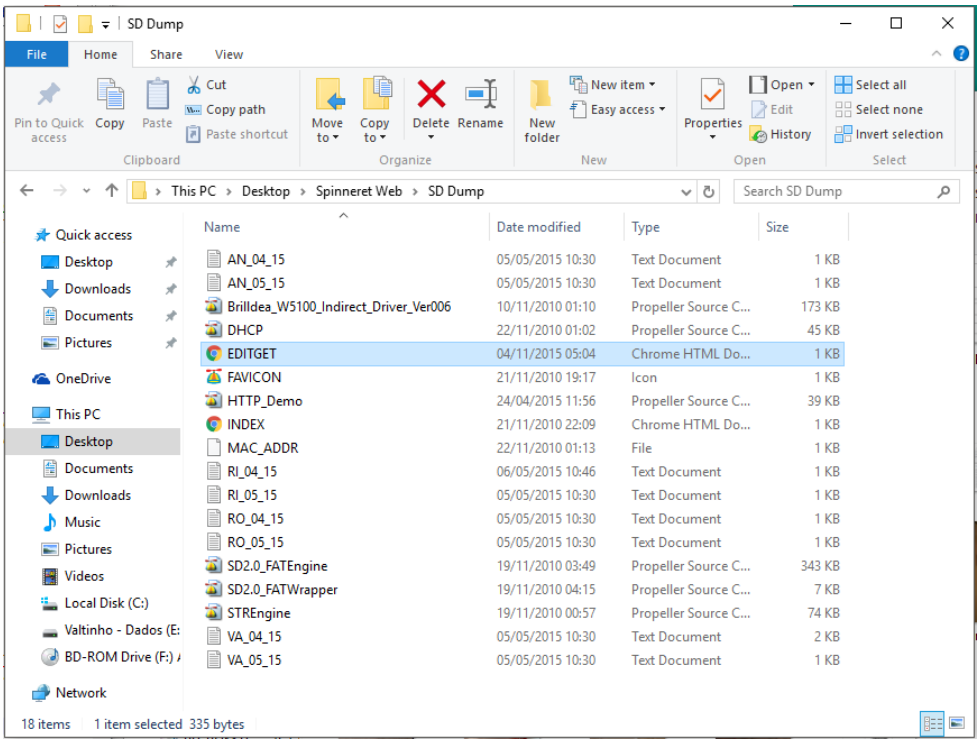

Figura 4.6: Conteúdo do cartão SD para o acesso *Web*.

4.3 Microcontrolador *Parallax Propeller* e linguagem SPIN

Primeiramente, dois aspectos interessantes a se destacar são a combinação que existir entre a linguagem SPIN e o microcontrolador, com inúmeras funcões baseadas na arquitetura do microcontrolador, e a facilidade de adaptação e desenvolvimento em linguagem SPIN.

A IDE *Propeller Tool* primeiramente compila o código escrito em linguagem SPIN e o transforma em um objeto, como descrito na seção 2.2, fazendo as medidas necessárias para que não seja excedida a memória utilizada, como exemplificado na figura 4.7. Também é possível gravar o programa tanto na RAM do microcontrolador como na EEPROM, como mostrado na figura 4.8, ou mesmo salvar localmente o arquivo executável que é carregado no

microcontrolador, além de se obter acesso ao código implementado em Hexadecimal.

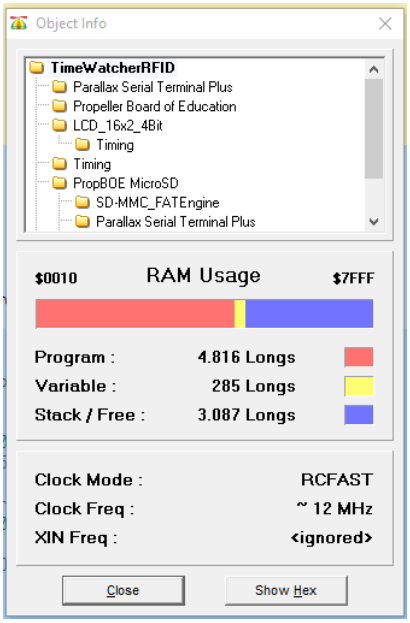

Figura 4.7: Informações gerais do objeto.

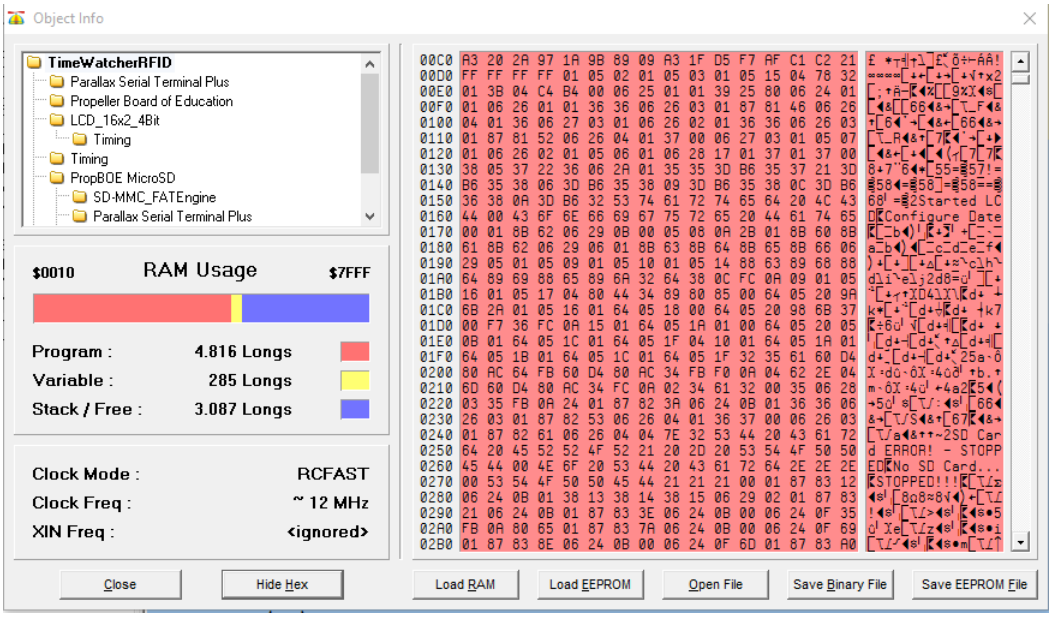

Figura 4.8: Informações expandidas do objeto.

A partir disso, é possível inclusive carregar vários arquivos ".eeprom"em um cartão SD, por exemplo, e fazer com que o microcontrolador os carregue na sua memória RAM em tempo de execução. Isso significa que, dentro da própria linguagem SPIN, é possível fazer com que vários programas sejam executados seguidamente sem que seja necessário ligar novamente o microcontrolador num computador para que seja reprogramado. Essa possi-

bilidade abre um campo enorme para exploração, sendo possível controlar vários sistemas independentemente e carregar inúmeros programas dentro de um cartão SD. Por uma questão de não necessidade, neste trabalho esse recurso não é utilizado, fazendo uso apenas das funções *multicore* que são descritas abaixo.

É importante atentar para o fato de que mesmo um cartão SD de 2 GB pode carregar inúmeros programas, pois cada programa salvo como ".eeprom"tem apenas 32 KB de extensão.

Permeando essas facilidades do sistema, a linguagem SPIN encaixa perfeitamente no que diz respeito ao uso do microcontrolador. Operações simples são necessárias para carregar um código. Em geral, usa-se o comando *COGINIT* ou *COGNEW* para executar uma função em um novo *cog*. A diferença desses métodos está nos parâmetros que eles exigem. Enquanto o primero necessita como parâmetro o *id* do *cog* onde uma função deve ser realizada, o segundo simplesmente inicia a função em um *cog* ocioso. Sendo assim, como boa prática, recomenda-se o uso da função *COGINIT* como um fator de organização e administração do sistema.

Além disso, o trabalho com um microcontrolador que não se baseia em interrupção é feito de maneira muito simples, apenas fazendo o monitoramento de um botão, ou de algo que deve realizar algum evento dentro do sistema. No caso deste trabalho, o que seria análogo a uma interrupção é o evento em que uma pessoa dá entrada ou saída do controle de acesso. Isso acontece dentro de um *loop* que verifica o tempo todo o resultado de uma função do *driver* do RFID. Quando este resultado é positivo para um cartão, um evento é desencadeado, conforme a figura 4.9

```
result := Rfid.RequestFromMfrc522(Rfid#PICC_REQIDL, @serNum)

if result == Rfid#MI_OK
    'debug.Str(string(13, "Card detected, type = $"))
    'debug.Hex(serNum[0], 2)
    'debug.Hex(serNum[1], 2)

    ' Anti-collision, return card serial number 4 bytes
    result := Rfid.Mfrc522Anticollision(@serNum)

    if result == Rfid#MI_OK
        result := Rfid.GetChecksum(@serNum)
        cardNumber := 0
        Rfid.SwapEndians(@cardNumber, @serNum)

        if result & $FF == serNum[4]
            'debug.Str(string(13, "Card Data Successfully Read"))
        else
            'debug.Str(string(13, "Warning CRC Error!!!"))
            'next

        'debug.Str(string(13, "Checking to see if card can be identified."))
        cardIndex := FindCardIndexNumber(cardNumber)
```

Figura 4.9: Evento análogo a uma interrupção em linguagem SPIN.

4.4 Comparação entre a linguagem SPIN, linguagem C *Propeller* e *Assembly* PASM

Como uma simples comparação, o método de ordenação *ShellSort* foi carregado nas três linguagens e rodado no microcontrolador. Inicialmente um vetor de 500 posições foi gerado aleatoriamente, sendo ordenado posteriormente por uma rotina *ShellSort*. A figura 4.10 mostra a ordenação implementada na linguagem C utilizando a Simple IDE. A figura 4.11, por sua vez, mostra a implesmentação em SPIN, na IDE *Propeller Tool* e, finalmente, a figura 4.12 representa o algoritmo *ShellSort* implementado em PASM.

ShellSort.c *

```
13
14  for(i=num/2; i>0; i=i/2)
15  {
16     for(j=i; j<num; j++)
17     {
18       for(k=j-i; k>=0; k=k-i)
19       {
20          if(arr[k+i]>=arr[k])
21              break;
22          else
23          {
24              tmp=arr[k];
25              arr[k]=arr[k+i];
26              arr[k+i]=tmp;
27          }
28       }
29     }
30  }
```

Figura 4.10: *ShellSort* na linguagem C *Propeller*.

```
PUB shellsort (arrayAddr, arraylength, asc_desc) | inc, val, i, j
'' consistantly the fastest with less than 70 elements and more than 20

  inc := arraylength-- >> 1                                     ' get middle point (reduce arraylength so it's not re-evaluated each loop)
  REPEAT WHILE (inc > 0)                                        ' while still things to sort
    REPEAT i FROM inc TO arraylength
      val := long[arrayAddr][i]                                 ' store value for later
      j := i
      REPEAT WHILE (j => inc AND ((asc_desc == ASC AND long[arrayAddr][j - inc] > val) OR (asc_desc == DESC AND long[arrayAddr][j - inc] < val))) ' compare value
        long[arrayAddr][j] := long[arrayAddr][j - inc]          ' insert value
        j -= inc                                                ' increment
      long[arrayAddr][j] := val                                 ' place value (from earlier)
    inc >>= 1
```

Figura 4.11: *ShellSort* na linguagem SPIN.

```
DAT

                ORG
shellsrt
                MOV     parAddr2, parAddr

                MOV     pinc, parlen                            ' inc := arraylength
                SHR     pinc, #1                                ' arraylength >> 1

bigloop         TJZ     pinc, #end                              ' REPEAT WHILE (inc > 0)
                MOV     idx, pinc                               ' REPEAT i FROM inc
frompinc        CMP     idx, parlen     WZ, WC                  ' TO arraylength
        IF_AE   JMP     #lfrompinc

                MOV     parAddr, parAddr2                       ' arrayAddr
                MOV     addrAdd, idx                            ' i
                SHL     addrAdd, #2                             ' [i]
                ADD     parAddr, addrAdd                        ' long[arrayAddr][i]
                RDLONG  pval, parAddr                           ' val := long[arrayAddr][i]

                MOV     jdx, idx                                ' j := i

innerloop       CMP     jdx, pinc       WZ, WC                  ' REPEAT WHILE (j => inc
        IF_B    JMP     #linnerloop

                MOV     addr, parAddr2                          ' arrayAddr
                MOV     addrAdd, jdx                            ' j
                SUB     addrAdd, pinc                           ' j - inc
                SHL     addrAdd, #2                             ' [j - inc]
                ADD     addr, addrAdd                           ' long[arrayAddr][j - inc]
                RDLONG  p1, addr                                ' long[arrayAddr][j - inc]

                CMPS    p1, pval        WZ, WC
        IF_Z    JMP     #linnerloop
                CMP     ascdesc, #0     WZ
IF_Z_AND_C      JMP     #linnerloop                             ' long[arrayAddr][j - inc] > val
 IF_A           JMP     #linnerloop                             ' long[arrayAddr][j - inc] < val   (IF_A == IF

                MOV     parAddr, parAddr2                       ' arrayAddr
                MOV     addrAdd, jdx                            ' j
                SHL     addrAdd, #2                             ' [j]
                ADD     parAddr, addrAdd                        ' long[arrayAddr][j]
                WRLONG  p1, parAddr                             ' long[arrayAddr][j] := long[arrayAddr][j - in
                SUBS    jdx, pinc                               ' j -= inc
                JMP     #innerloop

linnerloop      MOV     parAddr, parAddr2                       ' arrayAddr
                MOV     addrAdd, jdx                            ' j
                SHL     addrAdd, #2                             ' [j]
                ADD     parAddr, addrAdd                        ' long[arrayAddr][j]
                WRLONG  pval, parAddr                           ' long[arrayAddr][j] := val
                ADD     idx, #1                                 ' STEP 1
                JMP     #frompinc

lfrompinc       SHR     pinc, #1                                ' inc >>= 1
                JMP     #bigloop

end             WRLONG  negone, PAR
                COGID   p1                                      ' get cog id
                COGSTOP p1                                      ' kill this cog
```

Figura 4.12: *ShellSort* na linguagem PASM.

É notável que o desempenho tanto na linguagem C quanto na linguagem SPIN são bem próximos, enquanto a ordenação feita na linguagem PASM foi muito superior em relação ao desempenho, utilizando muito menos ciclos de *clock*. Duas coisas devem ser levadas em consideração neste caso. Primeiramente, caso seja necessário um desempenho diferenciado, a linguagem *Assembly* é altamente recomendada, sendo muito mais rápida em relação ao processamento. Caso processamento não seja o mais crítico, as outras duas linguagens são mais recomendadas, principalmente pela simplicidade, visto que mesmo uma rotina simples em SPIN ou em C *Propeller* pode ter uma implementação bastante complexa em *Assembly*. Felizmente, para este trabalho, não foi necessária a implementação de nenhuma rotina em *Assembly*, sendo o SPIN a linguagem escolhida para a implementação do projeto.

O resultado da ordenação tanto em C como em SPIN está mostrado na figura 4.13. Esta figura mostra a quantidade média de ciclos utilizados para ordenar o vetor em questão.

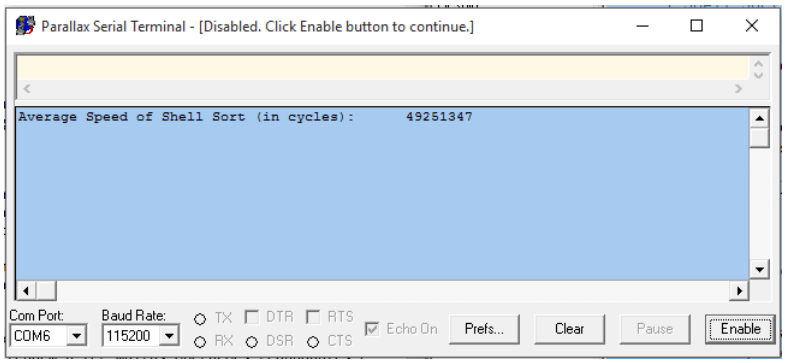

Figura 4.13: Resultado da ordenação nas linguagens C e SPIN.

Este resultado é esperado, visto que rotinas em *Assembly* são normalmente otimizadas, enquanto níveis mais altos tendem a ter muitas operações redundantes e com ciclos muito mais demorados. O resultado da ordenação em PASM se apresenta na figura 4.14

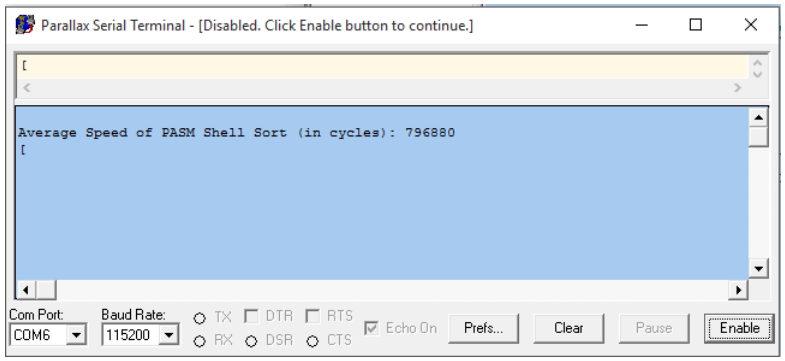

Figura 4.14: Resultado da ornação na linguagem PASM.

4.5 Limites do sistema de controle de acesso.

Este sistema é alimentado tanto por bateria de três células comum quanto ligado diretamente na tomada. Testes mostraram que o controle de acesso pode ser feito por mais de uma semana mesmo que sem alimentação externa, trabalhando apenas consumindo energia diretamente da bateria. Sendo assim, em relação à alimentação, os limites são mais do que aceitos, visto que, em geral, não se passa mais do que algumas horas sem abastecimento de energia num ambiente de trabalho.

Em relação à quantidade de informação que o sistema é capaz de carregar, é importante notar que todos os arquivos são gerados no formato texto comum ".txt". Sendo assim, mesmo com um cartão SD de 2 GB de memória, uma quantidade enorme de dados pode ser armazenada, referente até mesmo a anos de registro de eventos de entrada e saída.

Quanto à segurança, aspectos básicos devem ser considerados, visto que os eventos registrados são feitos através de dispositivos físicos como cartões ou qualquer outro dispositivo com uma *tag* RFID identificada pelo sistema. Portanto, como boa prática, não se deve portar um cartão que não seja pessoal. Além disso, qualquer pessoa com acesso ao ambiente tem condição de remover o cartão SD do sistema e alterar os dados. Esse tipo de problema não foi contornado neste trabalho.Já na plataforma *Web*, é necessário que o usuário faça um *login* para que tenha acesso aos arquivos do cartão SD, não sendo possível alterá-los por essa via.

Capítulo 5

Conclusão

Primeiramente, é importante dizer que o projeto funcionou como o projetado, ou seja, ele realiza todas as funções originalmente propostas.

É importante salientar que este projeto tem um potencial de desenvolvimento bem grande a partir do ponto onde parou, sendo expansível para inúmeras características além do que já está implementado.

Este *software* foi desenvolvido no decorrer do ano, tendo sido modificado inúmeras vezes desde sua ideia original. Com isso, várias características foram mudadas com o tempo, e novas foram adicionadas. Originalmente, este projeto não possuía RFID, sendo controlado por botões, que, porém, não tornavam o sistema reutilizável para uma quantidade maior de pessoas. Sendo assim, o RFID foi incluído após uma pesquisa sobre qual método seria mais viável a partir do preço de mercado de cada um.

A maior dificuldade encontrada certamente foi a de integrar todos os dispositivos presentes no sistema e fazê-los funcionar como um produto fluido e sem falhas. Muitas horas foram gastas para identificar os problemas de fluxo do software e, com isso, torná-los mais efetivos.

De um modo geral, o aprendizado durante este processo de desenvolvimento foi muito grande e contribuiu muito para a formação do aluno. Desde o planejamento do *software*, quando ele tinha proporções menores, até a etapa final em que todo o sistema funciona sem problemas de fluxo, a capacidade de trabalho e a condição técnica do aluno evoluíram rapidamente, preparando ainda mais o aluno para o mercado de trabalho.

Atualmente, este sistema encontra-se instalado num ambiente de trabalho e numa escola, onde os professores e colaboradores registram suas entradas e saídas. Isto não tem como objetivo ser usado para controle legal de horas, mas serve como base para o controle do fluxo de pessoas nestes ambientes.

Em relação ao trabalho com as ferramentas utilizadas neste projeto, algumas considerações são importantes. Primeiramente, foi grande o crescimento técnico, visto que ferramentas que não são utilizadas na graduação tiveram um papel principal aqui. Utilizar um microcontrolador sem interrupções e fazer com que ele funcione num ambiente diferente do ambiente da graduação foram tarefas difíceis porém muito motivantes. Infelizmente, a linguagem SPIN não possui um *debugger*, o que em alguns casos foi bastante frustrante, sendo esta a maior limitação de toda a *suite* do microcontrolador *Parallax Propeller*. Porém, mesmo com esta limitação, o microcontrolador se saiu muito bem em todos os testes realizados, todos os *benchmarks* e se tornou um ponto a ser considerado em trabalhos futuros em que o aluno esteja envolvido.

5.1 Trabalhos Futuros

Inúmeras são as possibilidades de expansão para este sistema. Neste momento, o principal caminho para isso seria a incorporação de reconhecimento facial ou leitura biométrica, para que uma pessoa não precise carregar sempre um cartão ou qualquer outro dispositivo que possa interagir com o leitor RFID. Dessa maneira, o sistema se tornaria ainda mais abrangente, podendo alcançar uma gama maior de lugares para ser instalado.

Além disso, seria muito interessante o desenvolvimento de uma plataforma centralizada, que armazenasse dados coletados em vários ambientes em vez de se obter um acesso direto a cada um dos ambientes. Isso tornaria possível que uma empresa toda, com todas as suas repartições, tivesse seu acesso controlado de maneira ágil e fácil. Também, com a adição de um módulo *wifi*, este sistema se alinharia necessariamente ao conceito de Internet das coisas, ou IoT, em inglês, tornando-se um mecanismo de controle de acesso interligado à Internet sem fio local, aumentando sua abrangência.

Outras funções também poderiam ser implementadas, como a possibilidade de interação com o sistema *Web*, para que fosse possível cadastrar uma pessoa à distância, ou remover um cadastro do sistema. Também seria possível deixar pequenos avisos direcionados individualmente, como um lembrete ou uma chamada para que uma pessoa compareça a outro lugar.

Referências Bibliográficas

[1] Parallax propeller manual. https://www.parallax.com/sites/default/files/downloads/P8X32A-Web-PropellerManual-v1.2.pdf, Acesso em: 30 de outubro de 2015.

[2] Bob Violino. Rfid applications. http://www.rfidjournal.com/articles/view?1339/, Acesso em: 30 de outubro de 2015.

[3] Datasheet spinneret web. https://www.parallax.com/sites/default/files/downloads/32203-Spinneret-Web-Server-Documentation-v1.1.pdf, Acesso em: 30 de outubro de 2015.

[4] Datasheet wiznet w5100. http://www.wiznet.co.kr/wp-content/uploads/wiznethome/Chip/W5100/Document/W5100_Datasheet_v1.2.6.pdf, Acesso em: 30 de outubro de 2015.

[5] Ronald L. Rivest Stephen A. Weis, Sanjay E. Sarma. Security and privacy aspects of low cost radio frequency identification systems. http://saweis.net/pdfs/spc-rfid.pdf, Acesso em: 10 de novembro de 2015.

[6] Parallax propeller. http://www.parallax.com/, Acesso em: 30 de outubro de 2015.

[7] Mark Roberti. History of rfid. http://www.rfidjournal.com/articles/view?1338, Acesso em: 30 de outubro de 2015.

[8] Propeller tool ide. https://www.parallax.com/downloads/propeller-tool-software, Acesso em: 30 de outubro de 2015.

[9] Parallax propeller 2 - development. https://www.parallax.com/news/2014-09-19/propeller-2-schedule-update-longer-we-work-simpler-our-new-multicore-design-will, Acesso em: 30 de outubro de 2015.

[10] Mfrc522 mifare rfid reader. http://www.nxp.com/documents/data_sheet/MFRC522.pdf, Acesso em: 30 de outubro de 2015.

[11] Svn reference. http://svnbook.red-bean.com/en/1.7/index.html, Acesso em: 30 de outubro de 2015.

Apêndice A

Código a ser executado no microcontrolador *Parallax Propeller*.

```
OBJ

    debug : "Parallax Serial Terminal Plus"
    system : "Propeller Board of Education"
    lcd : "LCD_16X2_4BIT"
    timing : "Timing"
    sd : "PropBOE MicroSD"
    rtc : "DS1302_full"
    Rfid : "Mfrc522_140321e"

VAR

    byte hour, minute, second, day, month, year, dow
    byte old_second, old_day, old_month, old_year
    byte checkpoints[4] 'number of daily checkpoints
    byte cycleFinished[4] 'gets value TRUE when the 4th checkpoint is achieved
    byte sdFileIndex[4] 'index created by CheckNewDay on SD Card
    byte pressedBy 'keeps who did pressed the button
    long randomNumber 'random number for the visitors function

    byte fileNameOnSDCard[13] 'array of file names to be created on SD Card

    byte workingDay 'flag that tells if a day is a working day

    long totalBalance[4] 'total time balance
    long lastTotalBalance[4] 'total balance of the day before
    long dailyWork[4] 'daily time worked
    long dayBalance [4] 'remaining work time on a day
```

```
    long actualTime 'hour*60 + minute
    long lastCheckpointTime[4] 'last time an event was registered (hour*60 + minute)

    long cardNumber, cardIndex

    byte serNum[Rfid#MAX_LEN]
    byte tempBuffer[Rfid#MAX_LEN]

CON
    workTime = 480 'Daily work time, 8hours * 60minutes

    END_OF_SERIAL_NUMBER_LIST = -1

    ' cardIndex enumeration
    #0, RICARDO, RONALDO, VALTERKEYRING, VALTERCARD, VALTERBUSCARD

    ' Buttons Pin Assignment
    btn1 = 0
    btn2 = 3
    btn3 = 6
    btn4 = 9
    btn5 = 12

    {{
    btn -> ricardo = 0 -> dailyWork[0] -> dayBalance[0] -> totalBalance[0]
    btn -> ronaldo = 3 -> dailyWork[1] -> dayBalance[1] -> totalBalance[1]
    btn -> valter = 6 -> dailyWork[2] -> dayBalance[2] -> totalBalance[2]
    btn -> ana = 9 -> dailyWork[3] -> dayBalance[3] -> totalBalance[3]
    btn -> visitante = 12
    }}

    SPI_CLK = 5
    SPI_MOSI = 4
    SPI_MISO = 2
    SPI_CS = 7
    RESET = 1

    buzzer = 10
```

```
PUB MAIN

    INIT_ALL

    repeat
        STAND_BY
        'Listen if any button was pressed
        checkAction

PUB INIT_ALL
{{
Initiates the components used by the TimeWatcher
}}

    system.clock(80_000_000)
    'system.clock(5_000_000)

    'Initiating Serial Terminal
    debug.start(9600)

    'Initiating LCD Display
    lcd.start

    lcd.move(1,1)
    lcd.str(string("Started LCD"))

    timing.pause1s(1)
    lcd.clear

    lcd.move(1,1)
    lcd.str(string("Configure Date"))
    timing.pause1s(2)

    'Initiating and Configuring RTC Module
    configureDate
    lcd.clear
```

```
    'Mounting SD Card
    mountSD
    sd.Display
    'it should be sd.NoDisplay, but in the PropBOE MicroSD, if I use NoDisplay,
    'it sets the variable displayMode to FALSE (0) and then verifies 0 to display
    'any message on the serial terminal.

    'Initiatint RFID Module in SPI configuration
    Rfid.Init(SPI_MOSI, SPI_MISO, SPI_CLK, SPI_CS, RESET)

    'Setting direction registers (OUTPUTS / INPUTS)
    dira[btn1] := 0
    dira[btn2] := 0
    dira[btn3] := 0
    dira[btn4] := 0
    dira[btn5] := 0

    dira[buzzer] := 1

PRI mountSD

    if sd.Mount(0) <> 0
        debug.Str(string("SD Card ERROR! - STOPPED"))

        lcd.move(1,1)
        lcd.str(string("No SD Card..."))
        lcd.move(1,2)
        lcd.str(string("STOPPED!!!"))

        repeat

PRI configureDateli,_month,_day,_year,_dow,_hour,_min,_sec
    Debug.Str(String("Initiating...",13))

    '================================================================================
    'call this function each time the propeller starts
    rtc.init(19, 20, 21) 'ports Clock, io, chip enable
```

```
    '================================================================================
    'set time
    debug.str(string("Configure TimeWatcher Clock",13))
    debug.str(string("Enter 1 to change clock, 0 to proceed without setting time",13))
    if(debug.DecIn<>0)
        debug.str(string("Enter month (1-12)",13))
        _month:=debug.DecIn
        debug.str(string("Enter day (1-31)",13))
        _day:=debug.DecIn
        debug.str(string("Enter year (00-99)",13))
        _year:=debug.DecIn
        debug.str(string("Enter day of week (1-7)",13))
        _dow:=debug.DecIn
        debug.str(string("Enter hour (0-23)",13))
        _hour:=debug.DecIn
        debug.str(string("Enter minute (0-60)",13))
        _min:=debug.DecIn
        debug.str(string("Enter second (0-60)",13))
        _sec:=debug.DecIn

      'call this function only after DS1302 power on
      rtc.config 'Set configuration register

      'call this function to set DS1302 time
      rtc.setDatetime( _month, _day, _year, _dow, _hour, _min, _sec ) 'month, day, year, day of week, hour, minute, second

    '================================================================================
    'change the tricle charger setup from that currently defined in the config function
    'Trickle charger setup tc_enable diodeSel resistSel
    ' | | |
    rtc.write(rtc.command(rtc#clock,rtc#tc,rtc#w),(%1010 << 4) + (1 << 2)+ ( 1 ))

PUB STAND_BY

    rtc.readSecond(@second)

    'Checking if Second has changed
    if checkNewSecond
```

```
        'Reading data from RTC configured
        rtc.readTime( @hour, @minute, @second ) 'read time from DS1302
        rtc.readDate( @day, @month, @year, @dow ) 'read date from DS1302

        'Checking if Day has changed
        checkNewDay

        'Checking if Month has changed .
        checkNewMonth

        'Update LCD with new Date and Time
        updateClockLCD

        'Updating old date values
        old_day := day
        old_month := month
        old_year := year

PRI checkNewSecond

    if second == old_second
        result := FALSE
    else
        result := TRUE
        old_second := second

PRI checkNewDay | loop

    if newDay
        if workingDay
            repeat loop from 0 to 3
                if cycleFinished[loop]
                    'Saving tha last data on SD_Card
                    saveBalancesOnSDCard(loop)

                else
                    invalidateDayOnSDCard(loop*3) 'loop*3 = btn
```

```
                    if checkpoints[loop] == 0 'increment hours of worktime
                        totalBalance[loop] := totalBalance[loop] - workTime

                restartCycleFinished(loop)
                sdFileIndex[loop]++ 'if it's a working day, update file index

                lastTotalBalance[loop] := totalBalance[loop]

                checkpoints[loop] := 0

            'Reset workingDay flag
            workingDay := FALSE

        'lcd.clear 'displaying all the last time balances for debug purposes only
        'displayLastTotalBalances

PRI newDay

    if day <> old_day
        return TRUE
    else
        return FALSE

PRI restartCycleFinished (index)

    cycleFinished[index] := FALSE

PRI invalidateDayOnSDCard (btn)

    'Preparing the first two bytes to open the respective file
    'The other bytes are already prepared from the newMonthFileOnSDCard function
    if btn == btn1
        'Ricardo
        fileNameOnSDCard[0] := "R"
        fileNameOnSDCard[1] := "I"

    elseif btn == btn2
        'Ronaldo
```

```
            fileNameOnSDCard[0] := "R"
            fileNameOnSDCard[1] := "O"

        elseif btn == btn3
            'Valter
            fileNameOnSDCard[0] := "V"
            fileNameOnSDCard[1] := "A"

        elseif btn == btn4
            'Ana
            fileNameOnSDCard[0] := "A"
            fileNameOnSDCard[1] := "N"



        'Openning the file to invalidate the day
        sd.FileOpen(@fileNameOnSDCard[0],"A")

        'Checking if the worker came to work on that day, if he came, checkpoints > 0
        'If not, saves the data before invalidating the day.
        if checkpoints[btnIndex(btn)] == 0
            sd.WriteByte(13)
            sd.WriteByte(10)
            sd.WriteDec(sdFileIndex[btnIndex(btn)]) 'Writing INDEX on SDCard
            sd.WriteByte(44)
            writeLastDataOnSDCard

        'Invalidating the day line
        sd.WriteByte("*")
        sd.FileClose

PRI writeLastDataOnSDCard
        'Writing DD/MM/YY
        if old_day < 10
            sd.WriteByte("0")
        sd.WriteDec(old_day)
        sd.WriteByte("/")
        if old_month < 10
            sd.WriteByte("0")
```

```
    sd.WriteDec(old_month)
    sd.WriteByte("/")
    if old_year < 10
        sd.WriteByte("0")
    sd.WriteDec(old_year)

    sd.WriteByte(44)

PRI displayLastTotalBalances 'NOT AT WORKFLOW

    lcd.move(1,1)
    lcd.dec(lastTotalBalance[0])

    lcd.move(1,2)
    lcd.dec(lastTotalBalance[1])

    lcd.move(8,1)
    lcd.dec(lastTotalBalance[2])

    lcd.move(8,2)
    lcd.dec(lastTotalBalance[3])

    timing.pause1s(1)

    lcd.clear

PRI saveBalancesOnSDCard (index)

    'Preparing the first two bytes to open the respective file
    'The other bytes are already prepared from the newMonthFileOnSDCard function
    if index == 0
        'Ricardo
        fileNameOnSDCard[0] := "R"
        fileNameOnSDCard[1] := "I"

    elseif index == 1
        'Ronaldo
        fileNameOnSDCard[0] := "R"
```

```
            fileNameOnSDCard[1] := "O"

        elseif index == 2
            'Valter
            fileNameOnSDCard[0] := "V"
            fileNameOnSDCard[1] := "A"

        elseif index == 3
            'Ana
            fileNameOnSDCard[0] := "A"
            fileNameOnSDCard[1] := "N"

        'Open the file to save balances
        sd.FileOpen(@fileNameOnSDCard[0],"A")

        sd.WriteDec(dayBalance[index]) 'Saving day balance
        sd.WriteByte(44)
        sd.WriteDec(totalBalance[index]) 'Saving total time balance

        'Close File
        sd.FileClose

PRI checkNewMonth | loop

        if month <> old_month
            'creating new month files
            newMonthFilesOnSDCard
            'reseting line index of SD Card files.
            resetIndexArray

            'restarting Total Balance
            repeat loop from 0 to 3
                totalBalance[loop] := 0
                lastTotalBalance[loop] := 0


PRI newMonthFilesOnSDCard | monthUpper, monthLower, yearUpper, yearLower
{{
```

On SDCard, each file will be correspondent to a name followed by the working year and month.

The file scope will be:

"NN_MM_YY.txt"

Where:

NN −> worker correspondent initials;

MM −> number of the month;

YY −> number of the year.

}}

```
'Setting monthUpper and monthLower bytes, ASCII number format (added by 48)
monthUpper := 48 + month / 10
monthLower := 48 + month // 10

'Setting yearUpper and yearLower bytes, ASCII number format (added by 48)
yearUpper := 48 + year / 10
yearLower := 48 + year // 10

'Setting the 3..12 bytes for the general file name
fileNameOnSDCard[2] := "_"
fileNameOnSDCard[3] := monthUpper
fileNameOnSDCard[4] := monthLower
fileNameOnSDCard[5] := "_"
fileNameOnSDCard[6] := yearUpper
fileNameOnSDCard[7] := yearLower
fileNameOnSDCard[8] := "."
fileNameOnSDCard[9] := "t"
fileNameOnSDCard[10] := "x"
fileNameOnSDCard[11] := "t"
fileNameOnSDCard[12] := 0

'Setting first two bytes for each user in the user pool and creating a file on SDCard.

'Ricardo
fileNameOnSDCard[0] := "R"
fileNameOnSDCard[1] := "I"

'Creating a month file with a header
sd.FileNew(@fileNameOnSDCard[0])
```

```
    sd.FileOpen(@fileNameOnSDCard[0], "W")
    printHeaderOnSDCard
    sd.FileClose

    'Ronaldo
    fileNameOnSDCard[0] := "R"
    fileNameOnSDCard[1] := "O"

    'Creating a month file with a header
    sd.FileNew(@fileNameOnSDCard[0])
    sd.FileOpen(@fileNameOnSDCard[0], "W")
    printHeaderOnSDCard
    sd.FileClose

    'Valter
    fileNameOnSDCard[0] := "V"
    fileNameOnSDCard[1] := "A"

    'Creating a month file with a header
    sd.FileNew(@fileNameOnSDCard[0])
    sd.FileOpen(@fileNameOnSDCard[0], "W")
    printHeaderOnSDCard
    sd.FileClose

    'Ana
    fileNameOnSDCard[0] := "A"
    fileNameOnSDCard[1] := "N"

    'Creating a month file with a header
    sd.FileNew(@fileNameOnSDCard[0])
    sd.FileOpen(@fileNameOnSDCard[0], "W")
    printHeaderOnSDCard
    sd.FileClose

PRI resetIndexArray | loop

    repeat loop from 0 to 3
        sdFileIndex[loop] := 1
```

```
PRI printHeaderOnSDCard 'NOT AT WORKFLOW

    sd.WriteStr(string("#, date , start , timeOut, ... ,dayBalance ,totalBalance"))

PRI updateClockLCD

    lcd.move(5,1)
    lcd.str(string("BIOSPACE"))
    lcd.move(1,2)
    if day < 10
        lcd.str(string("0"))
    lcd.dec(day)
    lcd.str(string("/"))
    if month < 10
        lcd.str(string("0"))
    lcd.dec(month)
    lcd.str(string(" - "))
    if hour < 10
        lcd.str(string("0"))
    lcd.dec(hour)
    lcd.str(string(":"))
    if minute < 10
        lcd.str(string("0"))
    lcd.dec(minute)
    lcd.str(string(":"))
    if second < 10
        lcd.str(string("0"))
    lcd.dec(second)

PRI checkAction
{{
    if ina[btn1] == 0
            BTN_ACTION(btn1)
            repeat while ina[btn1] == 0
    elseif ina[btn2] == 0
            BTN_ACTION(btn2)
            repeat while ina[btn2] == 0
```

```
    elseif ina[btn3] == 0
            BTN_ACTION(btn3)
            repeat while ina[btn3] == 0
    elseif ina[btn4] == 0
            BTN_ACTION(btn4)
            repeat while ina[btn4] == 0
    elseif ina[btn5] == 0
            BTN_ACTION(btn5)
            repeat while ina[btn5] == 0
}}

    if ina[btn5] == 0
        BTN_ACTION(btn5)
        repeat while ina[btn5] == 0

    result := Rfid.RequestFromMfrc522(Rfid#PICC_REQIDL, @serNum)

    if result == Rfid#MI_OK
        'debug.Str(string(13, "Card detected, type = $"))
        'debug.Hex(serNum[0], 2)
        'debug.Hex(serNum[1], 2)

        ' Anti-collision, return card serial number 4 bytes
        result := Rfid.Mfrc522Anticollision(@serNum)

        if result == Rfid#MI_OK
            result := Rfid.GetChecksum(@serNum)
            cardNumber := 0
            Rfid.SwapEndians(@cardNumber, @serNum)

            if result & $FF == serNum[4]
                'debug.Str(string(13, "Card Data Successfully Read"))
            else
                'debug.Str(string(13, "Warning CRC Error!!!"))
                'next

            'debug.Str(string(13, "Checking to see if card can be identified."))
            cardIndex := FindCardIndexNumber(cardNumber)
```

```
            if cardIndex == -1
                'debug.Str(string(13, "****** Unknown card ******"))
                'debug.Str(string(13, "You might want to write this number down in order"))
                'debug.Str(string(13, "to add it to the program's choices.", 13, "$"))
                'debug.Hex(cardNumber, 8)
            else
                case cardIndex
                    RICARDO:
                        BTN_ACTION(btn1)
                    RONALDO:
                        BTN_ACTION(btn2)
                    VALTERKEYRING:
                        BTN_ACTION(btn3)
                    VALTERCARD:
                        BTN_ACTION(btn3)
                    VALTERBUSCARD:
                        BTN_ACTION(btn3)

PUB BTN_ACTION (btn)

    'Verifying the button pressed
    if btn == btn5
        buzzerConfirmation
        funnyMessageLCD
        'newDayOnRTC
        'timeIncrementOnTRC

    else
        'Setting the workingDay flag, indicating that someone's working on this day
        'The first worker to press sets that the day will be computed as a working day.
        workingDay := TRUE

        'button checkpoint counter increment
        checkpoints[btnIndex(btn)]++

        buzzerConfirmation
        displayBtnOwnerMessageOnLCD(btn)
```

```
        'checking start of a cycle
        if (checkpoints[btnIndex(btn)] // 2) == 1
            calculateDayTime(btn)
            restartCycleFinished(btnIndex(btn))
            saveDataTimeOnSDCard(btn)
            displayConfirmationOnLCD(btn)

        'end of a cycle
        else
            calculateDayTime(btn)
            finishCycle(btn)
            saveDataTimeOnSDCard(btn)
            displayConfirmationOnLCD(btn)

PRI buzzerConfirmation

    'Configuring the Counter A
    ctra[30..26] := %00100 'Setting to NCO
    ctra[5..0] := buzzer 'Setting APIN to 1
    frqa := 126_130 '(80MHz)Setting buzz frequency to 2349,3Hz. By all means, it's a D7.
    'frqa := 2018080 '(5MHz)

    'Broadcast the signal for 50ms, system clock divided by 20 (clkfreq/20)
    dira[buzzer] := 1
    waitcnt((clkfreq/20)+cnt)
    dira[buzzer] := 0

PRI funnyMessageLCD

    randomNumber := random

    case randomNumber
        0:
            lcd.clear
            lcd.move(1,1)
            lcd.str(string("Funciona..."))
            lcd.move(2,2)
```

```
        lcd.str(string("Curioooso ;)"))
        timing.pause1s(2)
        lcd.clear
1:
        lcd.clear
        lcd.move(1,1)
        lcd.str(string("1, 2, 3,"))
        lcd.move(3,2)
        lcd.str(string("Testando..."))
        timing.pause1s(2)
        lcd.clear
2:
        lcd.clear
        lcd.move(1,1)
        lcd.str(string("Pode ficar"))
        lcd.move(3,2)
        lcd.str(string("e trabalhar..."))
        timing.pause1s(2)
        lcd.clear
3:
        lcd.clear
        lcd.move(1,1)
        lcd.str(string("Nao tem o"))
        lcd.move(3,2)
        lcd.str(string("que fazer???"))
        timing.pause1s(2)
        lcd.clear
4:
        lcd.clear
        lcd.move(1,1)
        lcd.str(string("Pare de me"))
        lcd.move(3,2)
        lcd.str(string("apertar!!!"))
        timing.pause1s(2)
        lcd.clear
5:
        lcd.clear
        lcd.move(1,1)
```

```
        lcd.str(string("Aventuras de"))
        lcd.move(1,2)
        lcd.str(string("3,14159265358979"))
        timing.pause1s(2)
        lcd.clear
    6:
        lcd.clear
        lcd.move(1,1)
        lcd.str(string("Fale com o chefe"))
        lcd.move(1,2)
        lcd.str(string("Muito obrigado!"))
        timing.pause1s(2)
        lcd.clear
    7:
        lcd.clear
        lcd.move(1,1)
        lcd.str(string("Preciso de"))
        lcd.move(4,2)
        lcd.str(string("FERIAS!!!"))
        timing.pause1s(2)
        lcd.clear
    8:
        lcd.clear
        lcd.move(5,1)
        lcd.str(string("BioSpace"))
        lcd.move(2,2)
        lcd.str(string("a todo vapor!!"))
        timing.pause1s(2)
        lcd.clear
    9:
        lcd.clear
        lcd.move(3,1)
        lcd.str(string("Olha ai..."))
        lcd.move(3,2)
        lcd.str(string("Olha ai..."))
        timing.pause1s(2)
        lcd.clear
```

```
PRI displayBtnOwnerMessageOnLCD (btn)

    if btn == btn1
        lcd.clear
        displayCheckpointMessage (btn)
        lcd.move(4,2)
        lcd.str(string("Ricardo!!"))
        timing.pause1s(1)
        lcd.clear

    elseif btn == btn2
        lcd.clear
        displayCheckpointMessage (btn)
        lcd.move(4,2)
        lcd.str(string("Ronaldo!!"))
        timing.pause1s(1)
        lcd.clear

    elseif btn == btn3
        lcd.clear
        displayCheckpointMessage (btn)
        lcd.move(4,2)
        lcd.str(string("Valter!!"))
        timing.pause1s(1)
        lcd.clear

    elseif btn == btn4
        lcd.clear
        displayCheckpointMessage (btn)
        lcd.move(6,2)
        lcd.str(string("Ana!!"))
        timing.pause1s(1)
        lcd.clear

PRI displayCheckpointMessage(btn)

    'First event of the day
    if checkpoints[btnIndex(btn)] == 1
```

```
            lcd.move(1,1)
            lcd.str(string("Hello,"))

        'when the worker comes to work
        elseif (checkpoints[btnIndex(btn)] // 2) == 1
            lcd.move(1,1)
            lcd.str(string("Welcome back,"))

        'when the worker goes out
        else
            lcd.move(1,1)
            lcd.str(string("Good job,"))

PRI calculateDayTime (btn)

    actualTime := (hour*60) + minute

    'odd checkpoints. It means the end of a cycle on a working day
    if (checkpoints[btnIndex(btn)] // 2) == 0
        dailyWork[btnIndex(btn)] := actualTime - lastCheckpointTime[btnIndex(btn)]

        dayBalance[btnIndex(btn)] := dayBalance[btnIndex(btn)] + dailyWork[btnIndex(btn)]

        totalBalance[btnIndex(btn)] := totalBalance[btnIndex(btn)] + dailyWork[btnIndex(btn)]
        lastTotalBalance[btnIndex(btn)] := totalBalance[btnIndex(btn)]

    'even checkpoints. It means the start of a cycle on a working day
    else
        'restarting dayBalance at the first event of the day
        'iterating totalBalance variable to comprehend new day working hours
        if checkpoints[btnIndex(btn)] == 1
            dayBalance[btnIndex(btn)] := -workTime
            totalBalance[btnIndex(btn)] := totalBalance[btnIndex(btn)] - workTime
        'restarting dailyWork on every start
        dailyWork[btnIndex(btn)] := 0
        'saving the last checkpoint of an event to calculate hours of work
        lastCheckpointTime[btnIndex(btn)] := actualTime
```

```
PRI finishCycle (btn) ’NOT AT WORKFLOW

    cycleFinished[btnIndex(btn)] := TRUE

PRI saveDataTimeOnSDCard (btn)

    ’Preparing the first two bytes to open the respective file
    ’The other bytes are already prepared from the newMonthFileOnSDCard function
    if btn == btn1
        ’Ricardo
        fileNameOnSDCard[0] := "R"
        fileNameOnSDCard[1] := "I"

    elseif btn == btn2
        ’Ronaldo
        fileNameOnSDCard[0] := "R"
        fileNameOnSDCard[1] := "O"

    elseif btn == btn3
        ’Valter
        fileNameOnSDCard[0] := "V"
        fileNameOnSDCard[1] := "A"

    elseif btn == btn4
        ’Ana
        fileNameOnSDCard[0] := "A"
        fileNameOnSDCard[1] := "N"

    ’Openning respective file
    sd.FileOpen(@fileNameOnSDCard[0],"A")

    ’Saving timing checkpoints of the day
    if checkpoints[btnIndex(btn)] == 1 ’FIRST SAVE OF THE DAY
        sd.WriteByte(13)
        sd.WriteByte(10)

        sd.WriteDec(sdFileIndex[btnIndex(btn)]) ’Writing INDEX on SDCard
        sd.WriteByte(44) ’Separating by comma ","
```

```
            writeDataOnSDCard
            writeTimeOnSDCard

        else 'NORMAL SAVING, ONLY DATA AND TIME
            writeTimeOnSDCard

        'Closing file
        sd.FileClose

PRI writeDataOnSDCard 'NOT AT WORKFLOW

        if day < 10 'Writing DD/MM/YY
            sd.WriteByte("0")
        sd.WriteDec(day)
        sd.WriteByte("/")
        if month < 10
            sd.WriteByte("0")
        sd.WriteDec(month)
        sd.WriteByte("/")
        if year < 10
            sd.WriteByte("0")
        sd.WriteDec(year)

        sd.WriteByte(44) 'Separating by comma ","

PRI writeTimeOnSDCard 'NOT AT WORKFLOW

        if hour < 10 'Writing HH:MM:SS
            sd.WriteByte("0")
        sd.WriteDec(hour)
        sd.WriteByte(":")
        if minute < 10
            sd.WriteByte("0")
        sd.WriteDec(minute)
        sd.WriteByte(":")
        if second < 10
            sd.WriteByte("0")
```

```
    sd.WriteDec(second)

    sd.WriteByte(44) 'Separating by comma ","

PRI displayConfirmationOnLCD (btn) | hourLCD, minuteLCD

    if (checkpoints[btnIndex(btn)] // 2) == 1
        hourLCD := lastTotalBalance[btnIndex(btn)] / 60
        minuteLCD := lastTotalBalance[btnIndex(btn)]// 60

        minuteLCD := ||minuteLCD

        lcd.move(2,1)
        lcd.str(string("Last Balance:"))

        lcd.move(3,2)
        lcd.dec(hourLCD)
        lcd.str(string("h and "))
        lcd.dec(minuteLCD)
        lcd.str(string("m"))

        timing.pause1s(2)
        lcd.clear

    else
        hourLCD := totalBalance[btnIndex(btn)] / 60
        minuteLCD := totalBalance[btnIndex(btn)]// 60

        minuteLCD := ||minuteLCD

        lcd.move(1,1)
        lcd.str(string("Total Balance:"))

        lcd.move(3,2)
        lcd.dec(hourLCD)
        lcd.str(string("h and "))
        lcd.dec(minuteLCD)
        lcd.str(string("m"))
```

```
        timing.pause1s(2)
        lcd.clear


PRI btnIndex (btn)
{{
    This function simply returns the index on a 0..4 range from
    the pins related to btn1..btn5.
}}

    return btn/3

PRI newDayOnRTC | _hour,_min,_sec 'NOT AT WORKFLOW
{{
    This function is only for debug purposes.
    It moves the time on the RTC to a point that is very close to the end of the day,
    so we can cause the program to pass day by day very quickly by pressing a button.
}}

    _hour := 23
    _min := 59
    _sec := 57

    rtc.setDatetime( month, day, year, dow, _hour, _min, _sec ) 'month, day, year, day of week, hour, minute, second

PRI timeIncrementOnTRC | _hour, _minute, _second 'NOT AT WORKFLOW
{{
    This function is only for debug purposes.
    It moves the time on the RTC to a point that is very close to the end of the day,
    so we can cause the program to pass day by day very quickly by pressing a button.
}}

    _hour := hour + 4
    _minute := minute
    _second := second

    if _hour > 23
```

```
        _hour := 23
        _minute := 59
        _second := 55

    rtc.setDatetime( month, day, year, dow, _hour, _minute, _second ) 'month, day, year, day of week, hour, minute, second


PRI random | x 'NOT AT WORKFLOW

    x := second + minute + hour + dow
    x++
    ?x
    x := ||x

    return x // 10

PUB FindCardIndexNumber(serialNumberToMatch)
'' Finds the index number of card's serial number.
'' Used to find information associated with the
'' card.

  result := 0

  repeat while serialNumber[result] <> serialNumberToMatch and {
    } serialNumber[result] <> END_OF_SERIAL_NUMBER_LIST
    result++
  if serialNumber[result] == END_OF_SERIAL_NUMBER_LIST
    result := -1

DAT
'' Add serial numbers from previouly read cards.

serialNumber long $7E208623 '' Ricardo
                            long $972A20A3 '' Ronaldo
                            long $09899B1A '' ValterKeyRing
                            long $F7D51FA3 '' ValterCard
                            long $21C2C1AF '' ValterBusCard
                            long END_OF_SERIAL_NUMBER_LIST
```