Centro Universitário de Brasília – UniCEUB

FACULDADE DE TECNOLOGIA E CIÊNCIAS SOCIAIS APLICADAS – FATECS

Curso de Engenharia da Computação

Bruno Eduardo G. Flecha

MECANISMO DE ABERTURA ELETRÔNICO CRIPTOGRAFADO

Brasília
2010

Bruno Eduardo G. Flecha

MECANISMO DE ABERTURA ELETRÔNICO CRIPTOGRAFADO

Trabalho apresentado ao Cento Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção do Certificado de Conclusão de Curso de Engenharia da Computação.

Orientador: Thiago Toribio

Brasília
2010

Autoria: Bruno Eduardo Gonçalves Flecha
Título: Mecanismo de abertura Criptografado

Monografia apresentada à
Banca Examinadora da
Faculdade de Tecnologia e
Ciências Sociais Aplicadas do
UniCEUB
Brasília, 05 de Julho de 2010

Banca Examinadora
Os componentes da banca de avaliação, abaixo listados, consideram este trabalho aprovado.

Nome	**Titulação**	**Assinatura**	**Instituição**
Flávio Antônio Klein	Mestre em Estatística e Métodos Quantitativos		
Gil Renato Ribeiro Gonçalves	Doutor em Física		
Maria Marony Souza Farias Nascimento	Mestre em Engenharia Elétrica		

Agradecimentos

Acima de tudo ao Senhor Deus, pois sem ele nada seria possível.

Agradeço a minha mãe Carmem que é o amor da minha vida e com seu amor, fé, carinho me ensinou o verdadeiro motivo da vida e Humberto Flecha meu herói que é um exemplo de Homem, trabalhador e amigo onde nas suas palavras sempre encontro sabedoria e por tudo o que fizeram para que eu chegasse até aqui.

Agradeço à minha linda Renata Maia Pinheiro que nessa jornada foi uma verdadeira companheira, sempre pronta a ajudar e nos momentos mais difíceis uma palavra de tranqüilidade e de incentivo sempre estiveram presentes. Nas várias noites sem dormir a sua compreensão e apoio.

Aos meus irmãos Tiago e Fernanda sem quais a vida seria muito chata.

A minha madrinha Lourdes que com tanto amor torce por mim.

A minha tia Ilza uma incentivadora de toda a família.

Ao professor Thiago Toribio.

Resumo

A partir da observação das crescentes necessidades humanas, com objetivo de melhorar a segurança e comodidade das pessoas, utilizando os procedimentos de controle e automação, nasceu à motivação para a elaboração de um sistema eletrônico criptografado para abrir uma fechadura. Desse modo, este projeto destina-se a construção de um sistema de abertura eletrônico, que utiliza um código criptografado para abrir uma fechadura elétrica, alimentada por um transformador, por meio de um sinal enviado pelo microcontrolador e amplificado por um transistor. Assim, a partir do desenvolvimento de um código de criptografia utilizando a linguagem de programação C, inserido em um microcontrolador Atmega8, por meio de uma gravadora BSD, foi possível o desenvolvimento de uma solução de segurança integrada, que pode ser empregada em diferentes sistemas, como nas residências, comércios ou em automóveis, usando um único dispositivo de entrada criptografado.

Palavras-chave: segurança, fechadura, microcontrolador,criptografia.

Abstract

This project arose From the observation of increasing human needs, aiming to improve safety and convenience of people using the procedures of control and automation, was born to the motivation for the development of an electronic encrypted to open a lock. Thus, this project aims to build an electronic opening system, which uses an encrypted code to open an electric door locks, powered by a processor, via a signal sent by the microcontroller and amplified by a transistor. Thus, from the development of an encryption code using the C programming language, inserted in an ATmega8 microcontroller through a BSD label, it was possible to develop an integrated security solution that can be used in different systems, as in homes, businesses or automobiles, using a single input device encrypted.

Keywords: security, lock, microcontroller, encryption.

Sumário

Lista de Figuras

Lista de Quadros

Lista de Siglas

LED	*Diodo Emissor de Luz*
LCD	*Liquid Cristal Display*
V	Volt
A	Ampére
USB	Universal serial Bus
EEPROM	Electrically Erasable Programmable Read only Memory
SPP	Standart Parallel Port
EPP	Enhanced Parallel Port
ECP	Extended Port
RAM	*Random Access Memory*
ROM	*Read Only Memory*
EEPROM	*Electrically- Erasable Programmable Read- Only Memory*
SRAM	*Static Random Access Memory*

CAPÍTULO 1 – INTRODUÇÃO

1.1 Motivação

A evolução tecnológica e as rápidas mudanças ocorridas principalmente na última metade do século XX revolucionaram a sociedade. Desse modo, vários aspectos da vida cotidiana estão sendo revistos e repensados, pois a evolução da tecnologia agregada às necessidades humanas impulsionou o crescimento e interesse na área de controle e automação.

Dentro de um contexto global, a engenharia de controle e automação encerra diversos prismas, tendo em vista seu caráter multidisciplinar e as diferentes formas de ser aplicada, seja no contexto organizacional ou de cada indivíduo, como, por exemplo, no processamento de produtos de consumo [ROSÁRIO, 2005].

Assim, a automação mostra-se como uma ferramenta multifuncional capaz de atender desde as exigências do mundo industrial e corporativo até proporcionar maior conforto dentro das residências, tanto que atualmente fala-se em automação industrial, automação comercial, automação predial, automação residencial, dentre outras.

Ademais, essa flexibilidade mostrou-se um ponto fundamental na motivação para utilizar a automação em sistemas que busquem não apenas facilitar as atividades cotidianas, mas também visem atender as necessidades de segurança pelas quais a sociedade vem passando nos tempos atuais, tendo em vista a exigência de se alcançar o denominado bem estar humano, hoje presente nos mercados e políticas de todo o mundo

Atualmente, existem no mercado fechaduras que utilizam códigos numéricos ou reconhecimento biométrico, contudo o sistema mais utilizado para abrir uma fechadura ainda são as conhecidas chaves.

Este projeto se propõe a criar uma solução diferenciada, mais prática do que os dispositivos já existentes e sem perder a segurança, pois não utiliza as

comuns chaves e nem necessita de um processo de cadastramento prévio, ainda, possui outra vantagem: não é suscetível à falibilidade do sistema biométrico, cuja eficácia é naturalmente condicionada à constância das características físicas do usuário (ex. digital).

Assim, basta o fornecimento do dispositivo e da chave em si para que os objetivos de comodidade e segurança sejam alcançados, pois o código de criptografia cria um obstáculo a ser ultrapassado por um invasor com intuito de entrar, ligar ou acessar o dispositivo.

Desse modo, a partir da análise do mundo atual, no qual a tecnologia está cada vez mais presente, da necessidade de trazer modernização a um nicho ainda pouco explorado, reside a motivação para a construção do dispositivo de fechadura eletrônica criptografada.

1.2 Objetivo

O principal objetivo deste projeto é construir um mecanismo de abertura criptografado para interagir com um microcontrolador programado pelo método criptográfico de múltiplas letras denominado algoritmo de *Playfair,* para verificação de uma chave com a finalidade de abrir uma fechadura.

Para atingir o objetivo principal, é possível identificar os seguintes objetivos específicos:

1. Desenvolver um software em linguagem C, a ser inserido em um microcontrolador, com o objetivo de realizar a verificação do código criptografado inserido no dispositivo contendo uma memória EEPROM;

2. Construir uma gravadora BSD efetuar a conexão entre o computador e o microcontrolador;

3. Desenvolver um dispositivo capaz de abrir eletronicamente utilizando como técnica de criptografia o algoritmo *Playfair*;

4. Certificar, por meio da implementação de mostrador de cristal líquido, *display*, a correta operação do microcontrolador e a sua interface com o dispositivo.

1.3 Estrutura da Monografia

Além deste capítulo, esta monografia está organizada em outros quatro:

- Capítulo 2: Referencial Teórico – São apresentados os conceitos essenciais ao desenvolvimento do dispositivo proposto;
- Capítulo 3: Desenvolvimento do Protótipo – É abordado o desenvolvimento de hardware e software, utilizando os métodos e conceitos descritos no referencial teórico.
- Capítulo 4: Testes – Apresenta a descrição dos principais testes realizados com o protótipo.
- Capítulo 5: Conclusão – Apresenta as principais conclusões obtidas dos resultados desta monografia e aponta as perspectivas para trabalhos futuros.

CAPÍTULO 2 – REFERENCIAL TEÓRICO

Este capítulo destina-se à apresentação das bases teóricas e conceitos envolvidos no desenvolvimento do projeto.

2.1 Microcontrolador Atmega8

Pode-se definir microcontrolador como um componente eletrônico que já tem incorporado em seu invólucro vários blocos componentes, permitindo a construção de sistemas compactos e poderosos de processamento. [SILVA, 1998].

Os microcontroladores (MCU) podem também denominados de controladores embutidos, pois representam um computador-num-chip, ou seja, o microcontrolador é um circuito integrado de alta densidade, pois estão inseridos dentro do chip a maioria dos componentes necessários para o controlador, como elucidado na figura 2.1, onde estão demonstrados os recursos tipicamente inseridos em um único sistema integrado. [ZELENOVSKY E MENDONÇA, 2010]

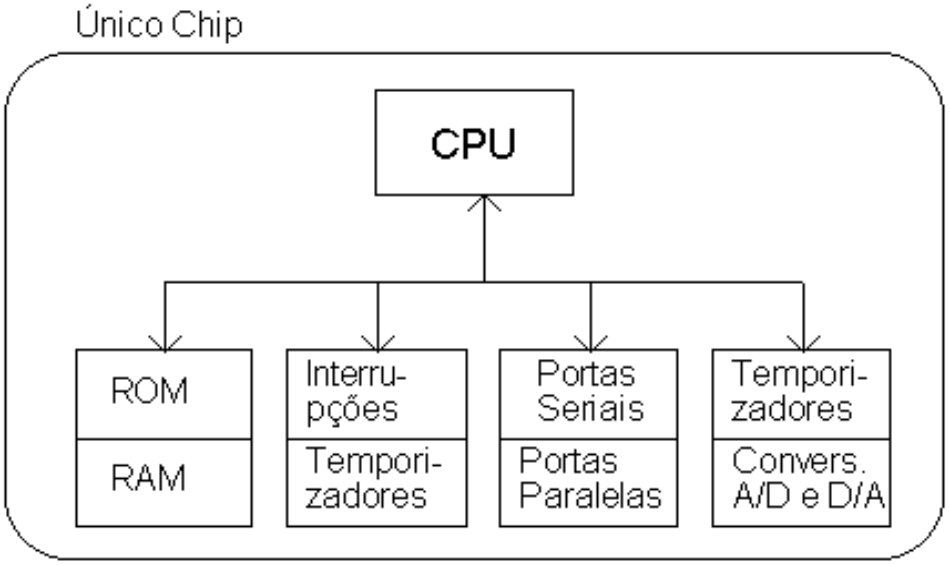


Figura 2.1 - Diagramas de blocos de um típico microcontrolador

[ZELENOVSKY E MENDONÇA, 2010]

A figura 2.1 representa as características básicas de um microcontrolador típico, porém existem diversos tipos de microcontroladores no mercado, cada qual com características especificas e utilidades variadas.

O Atmega8 é um microcontrolador de 8 bits que utiliza a tecnologia CMOS (*complementary metal-oxide-semiconductor*) e arquitetura RISC (*Reduced InstructionSet Computer*), capaz de executar uma instrução por ciclo de relógio, tendo em vista a conexão direta de seus 32 registradores de propósito geral com a unidade lógica aritmética [ATMEL, 2008].

Ademais, O microcontrolador Atmel AVR Atmega8 segue arquitetura Harvard[1], em que os barramentos associados às memórias de dados e do programa são distintos. Esse computador-num-chip utiliza *pipeline*, que consiste em uma técnica na qual o hardware processa mais de uma instrução por vez, de modo que enquanto uma instrução começa a ser executada, uma outra já é buscada na memória para ser executada no próximo ciclo de relógio.

Outra característica, é que esse microcontrolador possui um grande número de instruções (130), o que permite a otimização de código de alto nível em linguagem C. Isso significa que deve haver pouca diferença de tamanho entre um código C e o seu equivalente direto escrito em assembly. [ATMEL, 2008]. A figura 2.2 ilustra o microcontrolador Atmega8 e a figura 2.3 seu diagrama de blocos.

Figura 2.2 – Microcontrolador Atmega8. [Atmel, 2008]

[1] A arquitetura Harvard surgiu a partir da necessidade de melhorar o desempenho dos micro-controladores. Para isso, ela define o uso de duas memórias separadas: uma memória de dados e outra memória de programa (instruções). Essas duas memórias são conectadas por barramentos distintos e isso permite o uso de diferentes tamanhos de palavras para dados e instruções. [POZZO E VITTI, 2007]

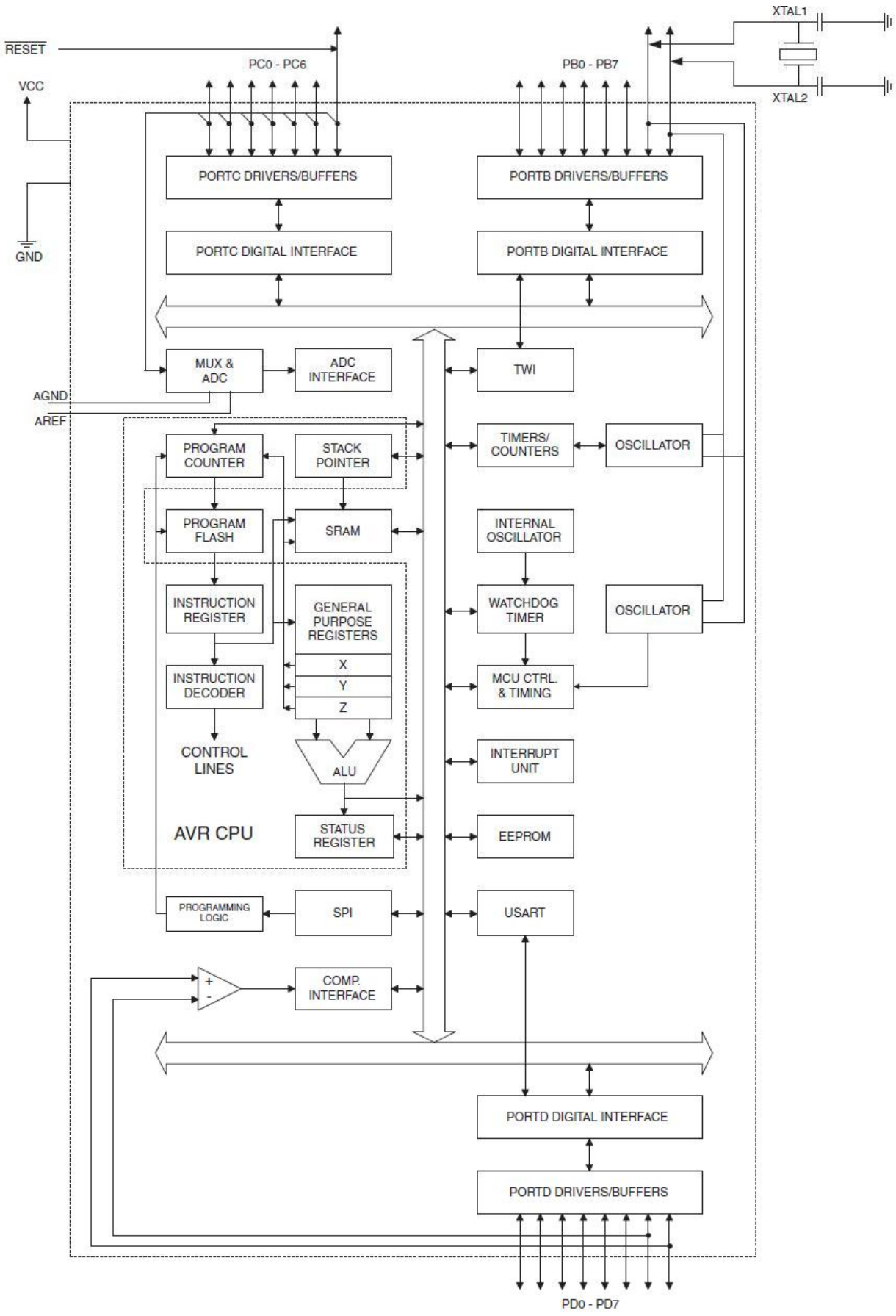


Figura 2.3– Diagrama de blocos do microcontrolador [Atmel, 2008]

São recursos inseridos no microcontrolador Atmega8 [SOUZA, 2010]:

• 130 Instruções de Programação;

• Processamento de até 16 MIPS (Milhões de Instruções por Segundo).

• 8Kbyte de memória de programa Flash que pode ser escrita até 10000 vezes;

• 1Kbyte de memória de dados SRAM;

• 512 Bytes de memória EEPROM;

• 6 Entradas analógicas no modelo DIP;

• 23 Linhas de I/O Programáveis

O diagrama pinos do Atmega8 está representado na figura 2.4 e descrito no quadro 2.1.

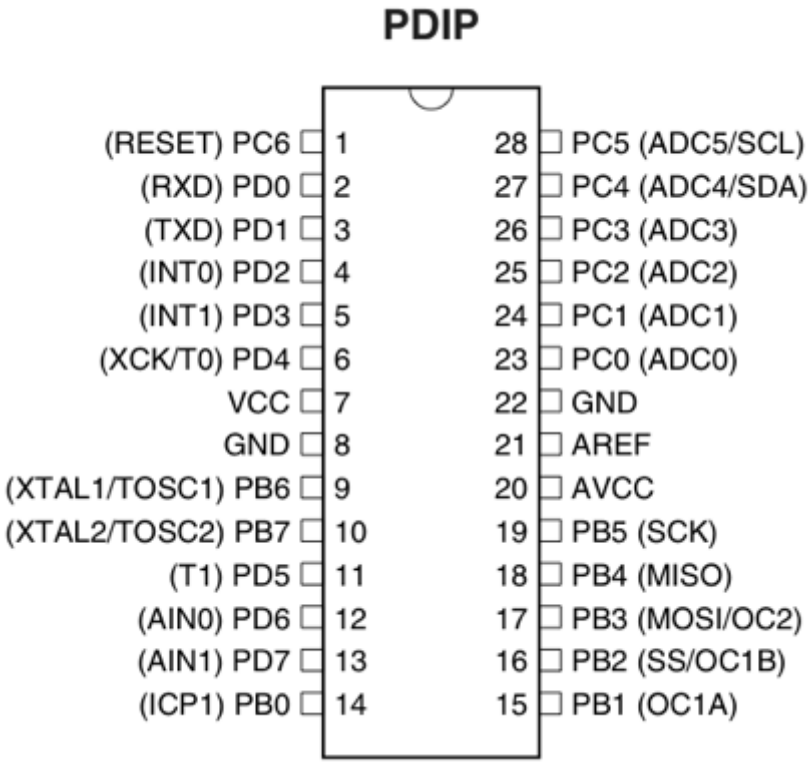


Figura 2.4 – Pinagem do Atmega8 [Atmel, 2008]

Quadro 2.1 – Referencias do ATmega8 [ATMEL, 2008]

VCC	Tensão Digital
GND	Terra (Ground)
Port B (PB7.. PB0) XTAL1 / XTAL2/TOSC1/TOSC2	As portas PB0 a PB7 são portas de 8 bits bidirecional de entrada e saída com resistores pull-up (selecionado para cada bit). A porta de saída buffers B possuem características de unidade simétrica. . Os pinos da porta B são de *tristate* quando a condição de reset se torna ativo, mesmo se o relógio não está funcionando. Dependendo das configurações do relógio fusível seleção, PB6 pode ser usado como entrada para a inversão Oscilador e amplificador de entrada para o circuito de relógio interno de funcionamento. Dependendo das configurações do relógio fusível seleção, PB7 pode ser usado como saída do amplificador inversor do oscilador.
Port C (PC5.. PC0)	Port C é uma entrada de 7 bits bidirecional . Os buffers de saída das portas dos pinos C possuem características de unidade de simetria. Os pinos da porta C são *tristate* indicado quando uma condição de redefinição torna-se ativa, mesmo se o relógio não está funcionando.
PC6/RESET	Se o fusível RSTDISBL está programado, PC6 é usado como um pino de entrada e saída. A característica elétrica do PC6 difere daquelas dos outros pinos das Port C. Se o fusível RSTDISBL não é programado, PC6 é usada como uma entrada de Reset. Um baixo nível deste pino para mais do que o comprimento do pulso mínimo vai gerar um Reset, mesmo se o relógio não está funcionando. Os pulsos não são garantidos para gerar um Reset.
Port D (PD7.. PD0)	Port D é uma porta de 8 bits bidirecional Port D serve também as funções de vários recursos especiais do ATmega8.
RESET	Entrada de Reset. Um baixo nível desta pino por mais tempo do que o comprimento de pulso mínimo vai gerar um reset, nem mesmo se o relógio está correndo.
AVCC	É o pino de tensão para o conversor A/ D, Porto C (3 .. 0) e ADC (7 .. 6). Ele deve ser conectado externamente ao VCC, mesmo que o ADC não esteja sendo usado. Se o ADC estiver sendo usado, deve ser ligado ao VCC através de um filtro passa-baixa. Note-se que
AREF	É o pino de referência analógica para o conversor A / D.

.

2.2 Gravadora

A gravadora é o dispositivo responsável por inserir os comandos de programação em um microcontrolador, pois os microcontroladores não possuem interface direta com o computador.

Assim, a gravadora pode ser definida como “um circuito que faz a interface entre o microcomputador e o microcontrolador, realizando as tarefas de programação e configuração” [ATMEL, 2008]. Esse processo de inserção pode se dar por meio de portas seriais ou paralelas, pois por intermédio dessa interface transfere-se o código programado para a memória flash do microcontrolador.

As gravadoras são estruturadas a partir do microcontrolador escolhido e podem ser compradas ou construídas. Nesse projeto optou-se pela construção da gravadora compatível com o microcontrolador escolhido, conforme verifica-se no Capitulo 3, item 3.3 dessa monografia.

2.3 Porta Paralela

Porta paralela pode ser definida como uma interface de comunicação entre computadores e periféricos. As portas paralelas foram criadas inicialmente para proporcionar uma conexão unidirecional e evoluíram para a transmissão bidirecional, proporcionando que diversos periféricos de entrada e saída de dados passassem a utilizá-la para transmitir suas informações. [VENTURI in AXELSON, 1997].

A transmissão em paralelo se dá por meio de várias vias condutoras de sinais que transferem grupos de *bits* simultâneos e possui 3 modos de operação:

Standard Parallel Port (SPP): Foi a primeira versão de porta paralela e possuía apenas comunicação unilateral. O padrão SPP pode transferir 8 *bits* de uma só vez a uma taxa de 150 KB/s. [VENTURI in AXELSON, 1997].

Enhanced Parallel Port (EPP): Possui transmissão bilateral. Tem capacidade de transmissão de 32 bits, quebrado em grupos de 8 bits durante a transmissão, a uma taxa de 2MB/s. [VENTURI in AXELSON, 1997].

Extended Capabilities Port (ECP): Também possui capacidade de transmissão bidirectional. A capacidade de transmissão das portas ECP é maior que a das portas EPP, pois essa utiliza um algoritmo de compressão e tem Acesso Direto à Memória (DMA). [VENTURI in AXELSON, 1997].

2.4 USB (*Universal Serial Bus*)

A USB (Universal Serial Bus) é uma arquitetura de barramento desenvolvida em colaboração por diversas empresas com o objetivo de criar uma porta padrão para os dispositivos externos. [TORRES, 2001]

O barramento USB possui a possibilidade de transferência de dados em diversas taxas. A versão 1.0 do USB permite a transmissão de dados a taxa de 1,5 Mbps, e a versão 1.1 atinge a taxa de até 12Mbps.

Com o desenvolvimento de periféricos de alto desempenho, foi necessário criar uma nova versão do USB. Assim, sem perder a compatibilidade, foi lançado o USB 2.0, com taxa máxima de transferência de 480 Mbps. [TORRES, 2001]

2.5 Memória EEPROM

A memória EEPROM (*Electrically Erasable Programmable Read Only Memory* - Memória Somente de Leitura Programável Apagável Eletricamente) é um tipo de circuito de memória ROM (*Read-Only Memory* - Memória Somente de Leitura). Nesse sentido, necessário estabelecer o conceito de memória ROM. [TORRES, 2001]

A memória ROM é um tipo de circuito integrado contendo um programa em seu interior. A principal característica desse tipo de memória é

que os dados uma vez gravados não podem ser alterados, ou seja, os dados podem ser gravados nela apenas uma vez, depois de efetuada a gravação não podem ser alterados ou apagados, apenas podem ser lidos pelo computador, por isso o nome de memória somente de leitura. Outra característica marcante das memórias ROM é sua não volatilidade, ou seja, os dados gravados não são perdidos na ausência de energia elétrica, ao contrário do que ocorre com as memórias RAM (*Random Access Memory)*, nas quais os dados inseridos são apagados na ausência de energia eletrica. [TORRES, 2001]

Assim, várias outras memórias com o tipo de circuito ROM foram criadas, dentre as principais possível identificar as seguintes: [TORRES, 2006]

- Mask-ROM: Tipo de memória na qual o programa é gravado durante o processo de fabricação do chip.
- PROM (*Programmable Read-only* Memory - Memória Programável só de Leitura*)*: A memória PROM, por meio de um gravador apropriado, pode sofrer apenas uma gravação após sua fabricação, ou seja, não é possível apagá-la após gravada ou reprogramar o chip.
- EPROM (*Erasable Programmable Read-only Memory* - Memória Programável só de Leitura Apagável*)*: A memória EPROM, assim como a PROM, é gravada por meio de um gravador especial, contudo difere dessa porque pode ser apagada e regravada. O apagamento é feito expondo o chip a uma luz ultra-violeta por um determinado período de tempo, de acordo com a idade da memória.
- EEPROM ou E2PROM: Assim como a memória EPROM, a memória EEPROM pode ser reprogramada, porém o que diferencia é o modo de apagamento, que é feio eletricamente e não mais através de luz ultra-violeta. A vantagem de apagar o conteúdo eletricamente reside na possibilidade de gravar e apagar o conteúdo da memória sem a necessidade de retirá-la do circuito, o que torna o processo mais rápido. Outra vantagem da memória EEPROM é a possibilidade de apagar e reprogramar palavras individuais, sendo assim desnecessário programar toda a memória.
- Flash-ROM: O método de apagamento também é realizado eletricamente e não por meio de luz ultra-violeta. A diferença básica entre a Flash-ROM e a

EEPROM é que na Flash-ROM não é possível apagar somente um determinado endereço dentro da memória e reprogramar apenas um dado, ou seja, é necessário reprogramar toda a memória.

2.6 Mostradores de Cristal Líquido ou Display de Cristal Líquido

Os mostradores de cristal líquido, diferente dos *leds*, não geram luz, simplesmente controlam a luz existente como forma de alterar sua aparência, assim esses mostradores alteram suas propriedades de luz para selecionar quais áreas deverão ficar claras e quais escuras [BOGART, 2001].

Existem dois métodos fundamentais de funcionamento dos cristais líquidos: espalhamento dinâmico e absorção.

No método de espalhamento dinâmico, em virtude do potencial elétrico aplicado externamente, as moléculas de cristal líquido adquirem orientação randômica assim, a luz que passa através do material é refletida em várias direções o que acarreta um brilho de aparecia fosca quando a luz emerge [BOGART, 2001].

No método de absorção as moléculas são orientadas de modo que alteram a polarização da luz que atravessa o material. Nesse método são usados filtros polarizadores, que dependendo da polarização que foi dada, eles absorvem ou deixam passar a luz, portanto a luz é visível apenas nas regiões onde ela pode emergir do filtro. [BOGART, 2001].

Um mostrador de cristal líquido é construído basicamente com uma camada fina de cristal líquido encapsulada em duas folhas de vidro, que possuem um material condutor transparente para possibilitar a aplicação de um campo elétrico no cristal. Os materiais condutores são os eletrodos com os quais uma tensão externa é conectada quando se deseja mudar a estrutura molecular do cristal líquido. Esses eletrodos podem ser gravados em segmentos-padrão ou individualmente, de modo que podem ser energizados seletivamente para dar origem ao padrão desejado.

2.6.1 Display de Cristal Líquido 20x2 com controladora HD44780

O modelo LCD HD44780 permite utilizar os 8x80 pixels de display e contêm um conjunto de caracteres ASCII standard, caracteres japoneses, gregos e símbolos matemáticos esse modelo usa o procedimento de absorção. [TORRES, 2001]

Cada um dos 640 pixels do display pode ser aceso individualmente, pois esse display possui chips de controle montados na superfície, da parte detrás do display, responsáveis por essa tarefa, o que permite poupar uma enorme quantidade de fios e linhas de controle, pois utilizando poucas linhas é possível fazer a ligação do display ao microcontrolador.

Esse mostrador de cristal líquido permite a comunicação com o microcontrolador por meio de um bus de 8 bits ou 4 bits.

Considerando a utilização de um bus de dados de 8 bits, o display precisa de uma alimentação de +5V mais 11 linhas de entrada e saída. Caso utilize-se o bus de dados de 4 bits, precisa de uma alimentação de +5V mais 07 linhas de entrada e saída. [TORRES, 2001]

Quando o display LCD não está habilitado, as linhas de dados *tristate* assumem o estado de alta impedância, assim não interferem com o funcionamento do microcontrolador.

Esse modelo de LCD requer que o microcontrolador gerencie as seguintes 3 linhas de “controle”: [TORRES, 2001]

O **Enable (E)** permite a ativação do display e a utilização das linhas R/W e RS. Quando a linha de Enable está no nível baixo, o LCD ignora os sinais R/W e RS. Quando Enable está no nível alto, o LCD verifica os estados das duas linhas de controle e reage de acordo com estes.

O **Read.Write (R/W)** determina o sentido dos dados entre o microcontrolador e o LCD. Quando está no nível baixo, significa que os dados será ser escritos no LCD. Quando está no nível alto, representa que os dados serão lidos do LCD.

A **Seleção de registro (RS)** permite que o LCD interprete o tipo de dados que estão presentes nas linhas de dados. No nível baixo, é escrito uma instrução no LCD. No nível alto um caracter é escrito no LCD.

O quadro 2.3 representa o estado lógico nas linhas de controle:

Quadro 2.3 – Do estado lógico do display

Enable	0 Acesso ao LCD inibido 1 Acesso ao LCD habilitado
Read.Write	0 Escrever dados no LCD 1 Ler dados do LCD
Seleção de registro	0 Instrução 1 Caracter

A leitura de dados do LCD é realizada utilizando o mesmo sistema descrito acima, porém a linha de controle tem que estar no nível alto. Desse modo, o LCD precisa ser iniciado e os comandos ou dados enviados para o módulo LCD. Os comandos típicos enviados depois de um reset podem ser: ativar um display, visualizar um cursor e escrever os caracteres da esquerda para a direita.

Depois de iniciado o LCD, ele está apto para continuar a receber dados ou comandos, assim ao receber um caracter, ele irá escrever no display e mover o cursor um espaço para a direita, pois o cursor marca a posição onde o próximo caracter será impresso.

Caso seja preciso escrever uma cadeia caracteres, primeiramente é necessário estabelecer um endereço de início e depois enviar os caracteres, um de cada vez. Os caracteres que o display possui estão guardados na RAM do display de dados (DD) e o espaço interno da DDRAM é de 80 bytes. [TORRES, 2001]

Cumpre acrescentar, que o display também possui 64 bytes de RAM Geradora de Caracteres (CG), sendo que os dados na RAM CG representam caracteres em um mapa de 8 bits. [TORRES, 2001]

Cada caracter consome até 8 bytes de RAM geradora de caracteres, assim o número total de caracteres que podem ser definidos pelo microcontrolador pode ir até oito. A figura 2.5 representa a conexão entre o micrcontrolador e o *display*. [TORRES, 2001]

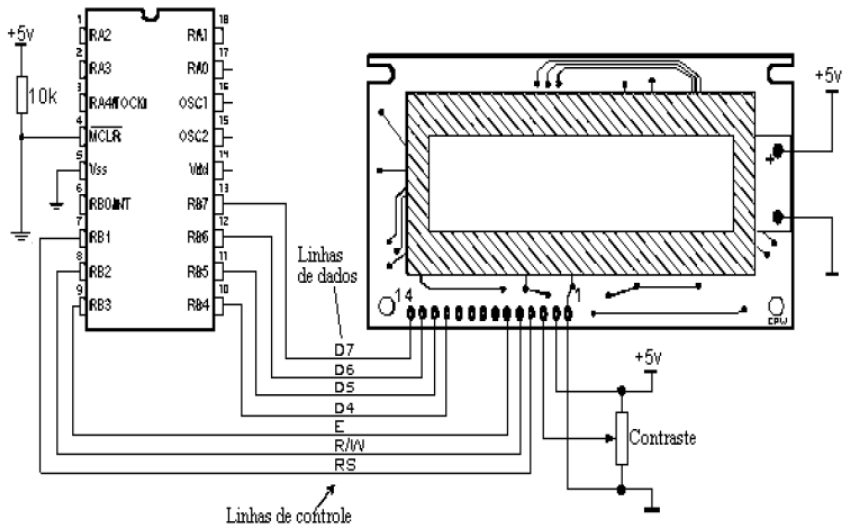


Figura 2.5 – Conexão entre o *Display* e o microcontrolador

2.7 Criptografia

Desde o surgimento das sociedades desenvolveu-se a necessidade de transmissão de informações de maneira secreta, de modo que o texto aparente se tornasse irreconhecível para estranhos, com o objetivo de proteger o conteúdo da mensagem. A partir dessa necessidade surgiu a técnica da criptografia, que continua sendo uma ferramenta utilizada até os dias de hoje.

Um dos primeiros relatos do uso de criptografia remonta à antiguidade clássica. Narrativas indicam que o imperador Júlio César utilizava

um método de substituição de letras em suas comunicações de guerra [2], com o intuito de esconder de seus inimigos as estratégias que deveriam ser adotadas [HINZ, 2000].

Etimologicamente, o termo criptografia surgiu da fusão das palavras gregas *kryptós* e *gráphein*, que significam respectivamente oculto e escrever. Desse modo, a criptografia pode ser definida como um conjunto de conceitos e técnicas que visa codificar uma informação, de forma que somente o emissor e o receptor possam acessá-la, evitando que um intruso consiga interpretá-la. [ALECRIM, 2009].

Segundo Oliveira [OLIVEIRA, 2008], a criptografia pode ser definida como a ciência de desenvolver cifras, com o objetivo de enviar uma mensagem em forma reconhecível e interpretável apenas para o destinatário e incompreensível para qualquer possível interceptador.

Com isso, a criptografia é uma técnica, na qual o transmissor transforma o conteúdo de uma mensagem clara em um texto irreconhecível, por meio do processo de cifragem. De outro lado, existe a decifragem, processo pelo qual o texto cifrado é transformado em texto claro.

A criptografia desempenha basicamente quatro diferentes funções nos modernos sistemas de informação: confidencialidade, autenticação, integridade e não-repúdio. [MENDES *in* GARFINKEL e SPAFFORD, 1999].

A confidencialidade ou privacidade representa um dos pilares da segurança da informação e significa que apenas o destinatário autorizado deve ser capaz de extrair o conteúdo da mensagem, ou seja, somente ele deve ser capaz de proceder a decifragem, de modo a preservar a privacidade, garantindo que apenas o transmissor e o receptor tenham conhecimento do conteúdo da mensagem.

[2] Algoritmo de chave simétrica por substituição criada por Júlio César, que recebeu o nome de Chave de César. Seu funcionamento pode ser conferido em [STLLINGS, 2008].

A autenticação do remetente denota a garantia de que o destinatário seja capaz de identificar se foi realmente o remetente quem lhe enviou a mensagem.

A integridade da mensagem cuida da capacidade do destinatário em determinar se a mensagem foi alterada durante a transmissão.

O não-repúdio tem o papel de impossibilitar o emissor de negar a autoria da mensagem, ou seja, se o receptor enviou uma mensagem, ele não pode negar falsamente que a tenha enviado.

Assim, é possível verificar que a criptografia possui em seu escopo promover a segurança da informação, tendo como função prevenir fraudes, promover certeza e privacidade, imputar responsabilidades, dentre outras. Essas características são responsáveis pela proliferação do uso da criptografia na era da informação e do desenvolvimento de novas tecnologias.

As técnicas mais conhecidas de criptografia envolvem o conceito de chaves criptográficas, que podem ser sintetizadas como um conjunto de bits baseados em um determinado algoritmo capaz de codificar e decodificar informações, de forma que o receptor da mensagem precisa usar uma chave compatível com a do emissor para extrair a informação contida na mensagem cifrada [ALECRIM, 2009].

Considerando o modelo de chaves criptográficas, em um primeiro momento, surgiu a criptografia simétrica, também chamada de criptografia de chave única e, em um segundo momento, surgiu a criptografia assimétrica ou de chave pública.

Dentre esses dois métodos, a criptografia de chave única ou simétrica continua sendo a mais usada [STALLINGS, 2008], principalmente porque, na prática, computacionalmente seu processamento é mais simples e rápido do que os algoritmos que utilizam chaves assimétricas.

De acordo com Stalligs [STALLINGS, 2008], a criptografia simétrica envolve a contextualização dos seguintes termos:

Texto claro: Mensagem ou dados inteligíveis, que serão usados como entrada, para alimentar o algoritmo de criptografia escolhido.

Algoritmo de criptografia: realiza as substituições e transformações no texto claro. Ele realiza o processo de cifragem utilizando a chave secreta.

Chave secreta: A chave secreta é um valor independente do texto claro e do algoritmo, também é entrada para o algoritmo de criptografia, pois as transformações e substituições realizadas pelo algoritmo dependem da chave utilizada. O algoritmo produz uma saída diferente, dependendo da chave especifica usada no momento.

Texto cifrado: É a mensagem embaralhada, produzida como saída. O texto cifrado pode ser definido como um fluxo de dados aparentemente aleatório e ininteligível.

Algoritmo de decriptografia: É basicamente o algoritmo de criptografia executado de modo inverso, ou seja, a partir do texto cifrado e da chave secreta, é produzido o texto claro.

A figura 2.6 demonstra um modelo simplificado da criptografia de chave simétrica.

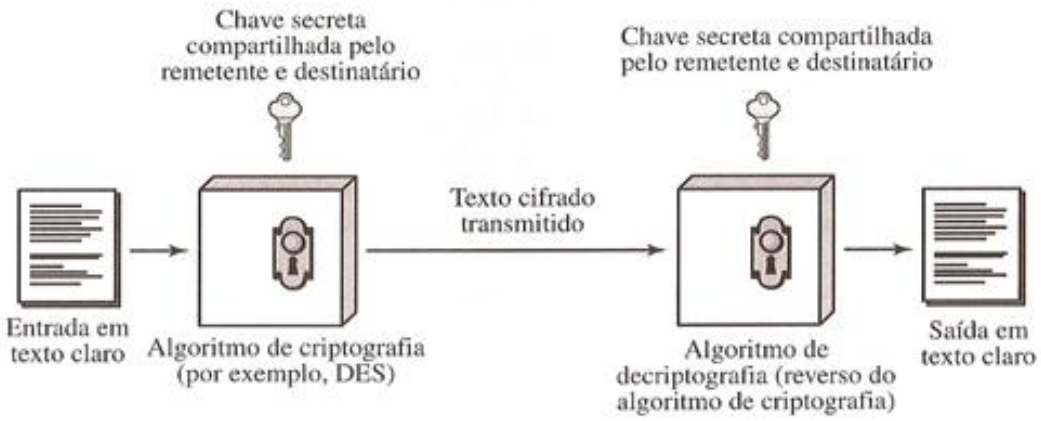


Figura 2.6 - Modelo simplificado do processo de criptografia convencional [STALLINGS, 2008]

Ainda, de acordo Stalligs em [STALLINGS, 2008], a segurança dos dados codificados dependem sinteticamente de dois requisitos: força do algoritmo empregado e habilidade de manter a chave secreta protegida.

Nesse sistema de chave única, tanto o emissor como o receptor compartilham a mesma chave, por isso a habilidade de manter a chave protegida de forma segura é essencial para manter o sigilo da informação.

Outro requisito para manter a segurança dos dados codificados utilizando a criptografia convencional é a força do algoritmo empregado, que representa a incapacidade de alguém não desejado decriptografar ou descobrir a chave secreta a partir da coleção de textos cifrados combinados com textos decifrados.

A criptografia de chave assimétrica, também denominada criptografia de chave pública, surgiu em 1970. Esse tipo de criptografia utiliza duas chaves, uma para realizar o processo de cifragem e outra para realizar o processo de decifragem. Cabe acrescentar que é possível combinar a criptografia simétrica com a assimétrica para obter melhores níveis de segurança. [STALLINGS, 2008]

A figura 2.7 elucida um modelo simplificado do processo de criptografia de chave pública.

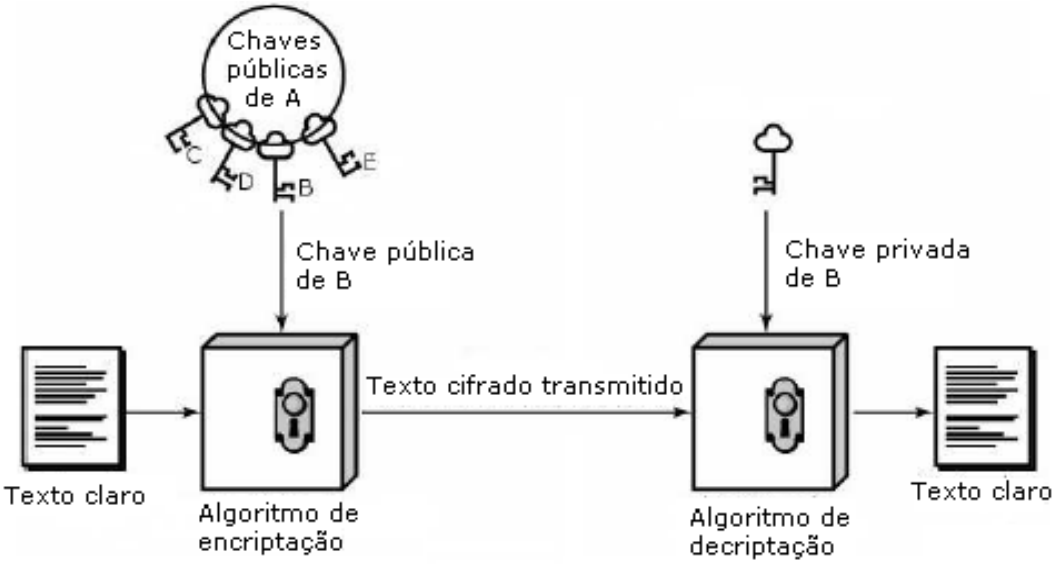


Figura 2.7 - Modelo simplificado do processo de criptografia de chave pública [STALLINGS, 2008].

A criptografia de chave pública funciona da seguinte forma: primeiramente, existe um par de chaves, uma chave é utilizada para cifrar a mensagem e outra chave é utilizada decifrar a mensagem, de modo que as mensagens são cifradas valendo-se da chave pública e somente podem ser decifradas fazendo uso da chave privada. Assim, a chave privada pode ser mantida em segredo e a chave pública ficada disponível para todos.

A figura 2.8 demonstra o processo de criptografia utilizando-se chave assimétrica.

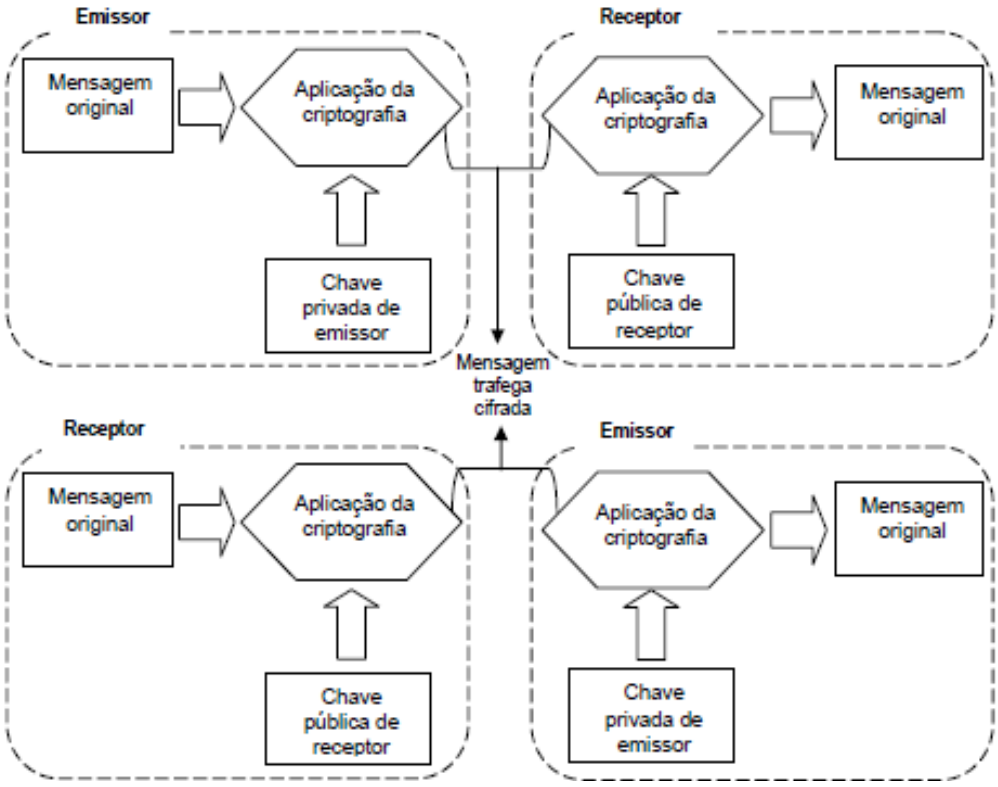


Figura 2.8 - Processo de criptografia por chave pública [MENDES *in* VOLPI, 2001]

A criptografia de chave pública representa um ganho em segurança quando comparada à criptografia convencional, porém seu processamento é mais complexo e demorado.

Assim, na prática, os algoritmos de chave pública ou assimétricos são entre 10 e 100 vezes mais lentos que os equivalentes de chave única ou simétricos. [MENDES *in* GARFINKEL e SPAFFORD, 1999].

2.8 *Playfair*

Em um breve contexto histórico, a cifra de criptografia *Playfair* foi criada por Charles Wheatstone em 1854, porém foi o Barão de *Playfair* que propôs sua utilização perante o ministério de assuntos externos do governo britânico e por isso foi batizada com seu nome. [STALLINGS, 2008]

O algoritmo de *Playfair* é a cifra de múltiplas letras mais conhecida dentre as técnicas clássicas de criptografia, que trata os diagramas no texto claro como unidades isoladas traduzindo tais unidades em diagramas de um texto cifrado, como anteriormente explicado não se trata de simétrico ou assimétrica e sim de múltiplas letras. [STALLIGS, 2008].

Desse modo, a criptografia de *Playfair* representa um cifrador de múltiplas letras baseado em uma matriz 5X5 de letras constituídas usando uma palavra-chave. Seu funcionamento pode ser sintetizado da seguinte forma: Uma matriz é construída a partir da palavra-chave escolhida, de modo que as letras da palavra-chave são colocadas da esquerda para a direita e de cima para baixo. Depois, a matriz é preenchida com as demais letras do alfabeto, conforme exemplificado na figura 2.9.

P	L	A	Y	F
I	R	B	C	D
E	G	H	J	K
M	N	O	Q	S
T	U	V	X	Z

Figura 2.9 – Matriz preenchida com a palavra-chave *playfair* e preenchida com as demais letras do alfabeto.

Por conseguinte, o texto claro é criptografado com duas letras de cada vez, conforme as seguintes regras especificadas por [STALLIGS,2008]:

1. As letras do texto claro repetidas no mesmo par são separadas por uma letra de preenchimento, como por exemplo, a letra X. Assim, a palavra balloon seria tratada como ba lx lo on;
2. As letras do texto claro que estejam na mesma linha da matriz são substituídas pela letra à direita e pela primeira letra da linha vinda após a última letra, de forma circular, conforme demonstrado na figura 2.10, na qual as letras IC deverão ser substituídas por RD respectivamente.

P	L	A	Y	F
I	R	B	C	D
E	G	H	J	K
M	N	O	Q	S
T	U	V	X	Z

Figura 2.10 - Exemplo de funcionamento da regra letras do texto claro na mesma linha.

3. As letras do texto claro que estejam na mesma coluna da matriz são pela letra que está abaixo com a letra de cima da coluna vinda após a última, de forma circular. O funcionamento dessa regra mostra-se exemplificado na figura 2.11, onde as letras YQ deverão ser substituídas pelas letras CX.

P	L	A	Y	F
I	R	B	C	D
E	G	H	J	K
M	N	O	Q	S
T	U	V	X	Z

Figura 2.11 – Exemplo de funcionamento da regra letras do texto claro na mesma coluna.

4. As letras do texto claro que estão em linhas e colunas diferentes são substituídas pelas letras da mesma fileira, mas no outro par dos cantos do quadrilátero definido pelo par original. De acordo com essa regra, ilustrada na figura 2.12, as letras do digrama serão substituídas pelas letras da mesma linha, mas no canto oposto. Sendo assim, M é substituído por Q e C é substituído por I.

P	L	A	Y	F
I	R	B	C	D
E	G	H	J	K
M	N	O	Q	S
T	U	V	X	Z

Figura 2.12 – Exemplo de funcionamento de linhas e colunas diferentes.

Com efeito, segue exemplo do processo de cifragem utilizando o método *playfair*:

O alfabeto cifrante proposto por Wheatstone trabalha com um alfabeto de 25 letras (matriz 5X5). Como o alfabeto latino possui 26 letras faz-se preciso eliminar uma das letras. Vários critérios podem ser utilizados, a variante inglesa mais utilizada considera I/J como apenas I. Nesse exemplo optou-se por excluir a letra W.

Dessa forma, o preenchimento da grade se inicia com uma palavra-chave escolhida, que no exemplo em tela optou-se pela palavra “chave” O restante das células é preenchido com as letras faltantes em ordem alfabética, conforme elucidado na figura 2.13. Esse exemplo foi alterado para trazer à realidade de como foi implementado no projeto.

C	H	A	V	E
B	D	F	G	I
J	K	L	M	N
O	P	Q	R	S
T	U	V	X	Z

Figura 2.13 – Matriz preenchida.

O texto claro escolhido é a palavra “decodificada”. Após o processamento das regras de cifragem do *Palyfair*, obtêm-se a seguinte mensagem cifrada: BTFBG BHVFH. De forma que os digramas apresentam-se conforme descriminado abaixo, que é a correlação entre a mensagem codificada e como ela fica embaralhada:

CO DI FI CA DA

BT FB GB HV FH

2.9 Linguagem C

A linguagem C foi inventada e implementada pela primeira vez por Dennis Ritchie e representa o resultado de um processo de desenvolvimento que começou com a linguagem BCPL, ainda em uso em algumas partes da Europa. [SCHILDT, 1997]

O padrão de fato, considerado por muitos anos, foi o fornecido com o sistema operacional UNIX versão 5, descrito na obra *The C Programming Language* de Brian Kernighan e Dennis Ritchie. Porém, com a popularização da linguagem C surgiram diversas discrepâncias e a necessidade de se estabelecer um padrão oficial para a linguagem. Assim, no verão de 1983, a ANSI (American Nacional Standards Institute), criou um comitê com o objetivo de definir um padrão para a linguagem C, criou-se assim o padrão C ANSI, reconhecido pelos principais compiladores C. [SCHILDT, 1997]

Dentre as principais características da linguagem C estão: linguagem de médio nível, estruturada e portável entre hardwares e sistemas operacionais.

A linguagem C é dita de médio nível porque combina elementos de linguagem de alto nível com a funcionalidade das linguagens de baixo nível, como o assembly, desse modo a linguagem C permite a manipulação de elementos básicos com os quais o computador funciona, tais como bits, bytes e endereços. [SCHILDT, 1997]

A característica de ser uma linguagem estruturada deriva da compartimentalização de códigos e dados, ou seja, possui a habilidade de dividir e esconder do resto do programa todas as informações para se realizar uma tarefa específica. Ademais, outra característica da estruturação deriva do uso de funções, construções de laço e blocos de códigos, dentre outras. [SCHILDT, 1997]

A portabilidade diz respeito à capacidade conferida a linguagem de se adaptar ao tipo de máquina ou sistema operacional, de forma que um código escrito em linguagem C pode ser executado em diferentes máquinas, independentemente de configuração física (hardware) e do sistema operacional residente. [SCHILDT, 1997]

CAPÍTULO 3 - DESENVOLVIMENTO DO PROTÓTIPO

Este capítulo apresenta o desenvolvimento do software inserido no microcontrolador e a montagem do protótipo proposto. Nesta topologia é possível perceber como o dispositivo funciona. Em substituição a uma chave é usado um dispositivo contendo um código para abrir uma fechadura.

Figura 3.1 – Topologia do sistema

3.1 Desenvolvimento do Hardware

Esse tópico destina-se a descrever o desenvolvimento físico do dispositivo, que pode ser observado no diagrama elétrico elucidado na figura 3.2.

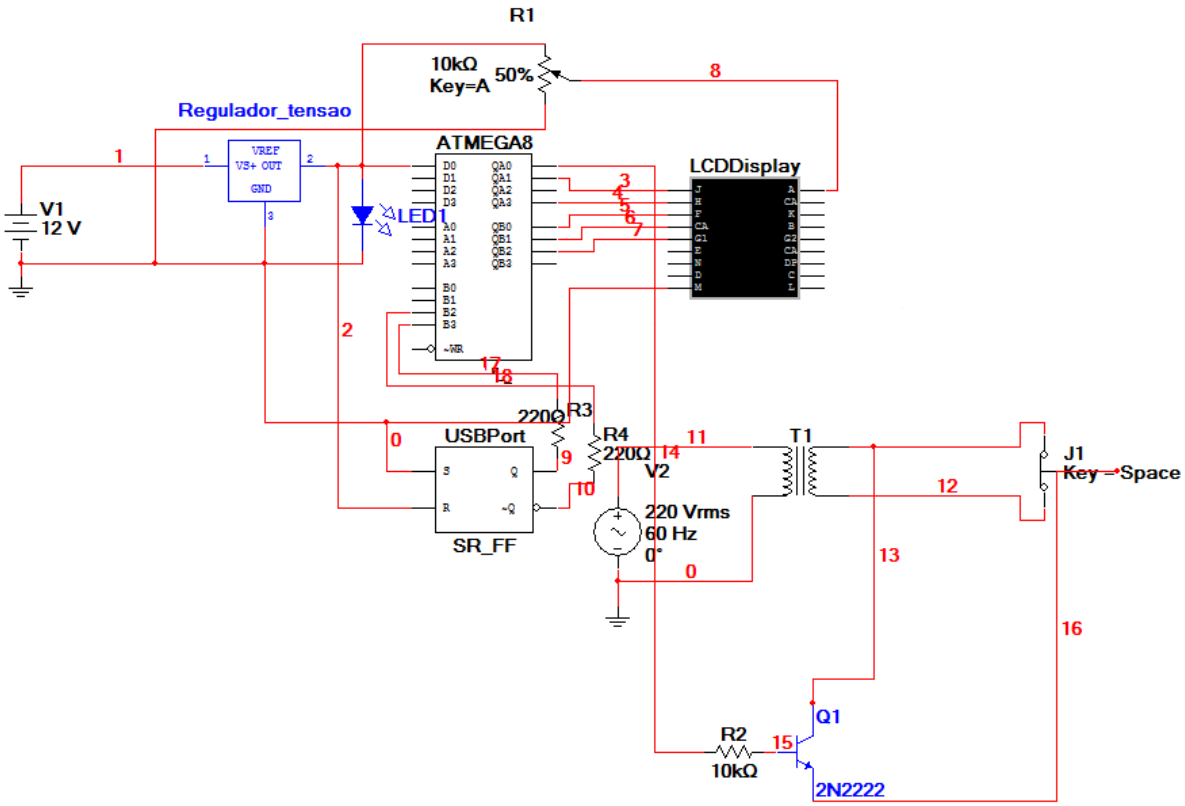


Figura 3.2 – Diagrama de elétrico do dispositivo.

3.1.1 Gravadora BSD

A gravadora BSD representa o elemento de conexão entre o microcontrolador e o computador, assim após a escolha do microcontrolador procedeu-se então a elaboração da gravadora. Cabe ressaltar, que pensando no desenvolvimento acadêmico foi decidido pela construção da própria gravadora, conforme se verifica nas figuras 3.3 a 3.11.

A montagem da gravadora BSD seguiu o diagrama esquemático apresentado na figura 3.3.

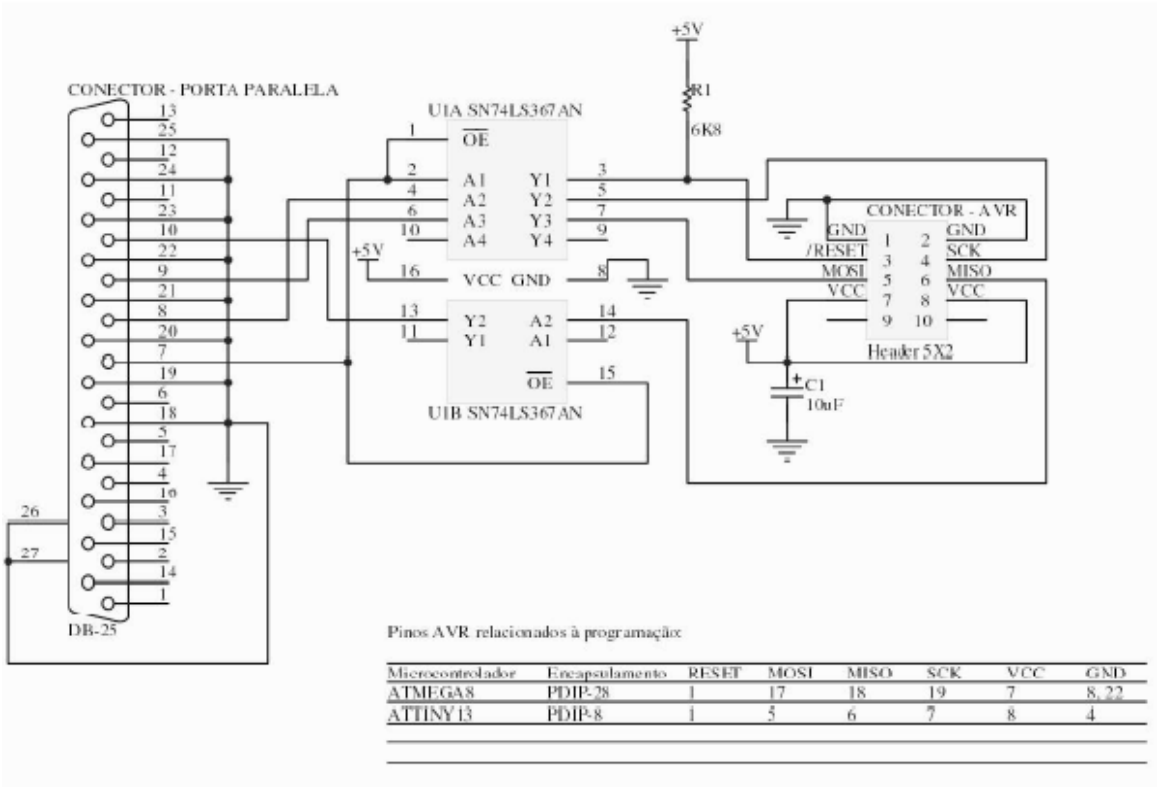


Figura 3.3 - Diagrama esquemático da gravadora BSD [ATMEL, 2008]

Para tanto, foram utilizados os seguintes componentes:

• 1 placa de circuito impresso confeccionada para a gravadora BSD;

• 1 circuito integrado 74LS367;

• 1 resistor de 4700 a 6800, 1/8W;

• 1 capacitor eletrolítico 10µF 16V, encapsulamento radial de tamanho mini;

• 1 conector DB-25 macho com capa plástica;

• 1 conector header 5x2;

• 70 cm de flat cable c/ 10 fios;

• fios para jumpers;

A gravadora, por intermédio de uma porta paralela, conecta-se ao microcomputador e, por meio de um cabo *flat,* com um conector de 10 pinos na extremidade, a gravadora é conectada ao microcontrolador. Para seu devido funcionamento, a alimentação da gravadora provém do próprio circuito em que está inserida.

De acordo com o diagrama ilustrado na figura 3.3, é possível identificar os principais elementos da confecção da gravadora BSD, quais sejam, o circuito integrado 74LS367, um resistor de 6800 ohms e um capacitor eletrolítico 10uF 16V. Como elemento de integração entre o microcomputador e o microcontrolador, do lado esquerdo da figura 3.4 é possível verificar os pinos que foram ligados ao conector DB-25, que por sua vez, foi ligado ao computador. Já do lado direito da figura 3.4 estão demonstrados os 8 pinos que foram conectados do circuito integrado para o microcontrolador.

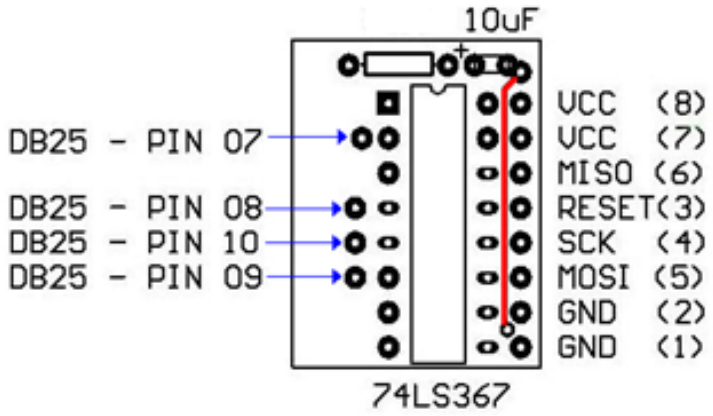


Figura 3.4 - Diagrama esquemático da gravadora BSD. [ATMEL, 2008]

Primeiramente, foi usada uma placa de circuito impresso, que foi cortada para o melhor dimensionamento. Após esse procedimento foi conectado na placa o circuito integrado 74LS367, bem como um resistor de 6800 Ω e um 1 capacitor eletrolítico 10µF 16V. Após essas conexões, foram realizadas as ligações de pinagem, da seguinte forma: os pinos 9 e 10 do circuito integrado foram ligados no terra, o pino 11 foi ligado no pino MOSI do microcontrolador, que significa *master out* e *slave in*, o pino 12 foi ligado no *clock* do

microcontrolador, o pino 13 foi ligado no RESET do microcontrolador, o pino 14 foi ligado no MISO que *master in* e *slave out* e os pinos 15 e 16 foram ligados no VCC.

O VCC representa a fonte de tensão usada e nesse projeto usou-se 5V, pois é a tensão que o microcontrolador Atmega8 trabalha e também o *display* 20x2 que usa a controladora HD44780.

A ligação do circuito integrado 74LS367 não foi feita diretamente no microcontrolador foi usado um conector header 5x2 para facilitar o processo de encaixe.

A figura 3.5 demonstra o diagrama de blocos referente ao circuito integrado 74LS367

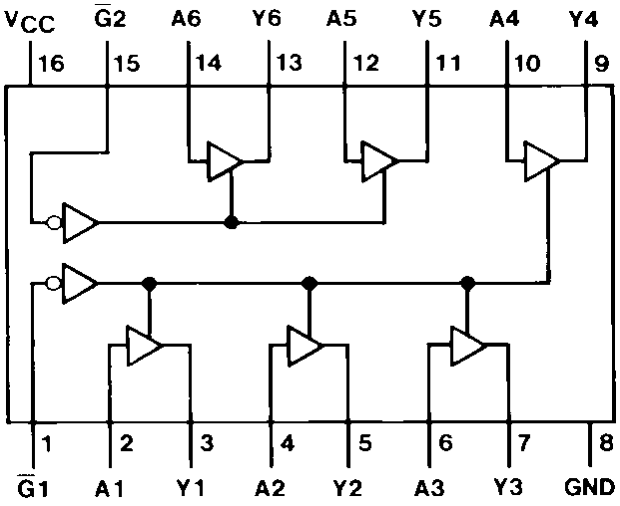


Figura 3.5 – Diagrama de pinos do circuito integrado 74LS367

3.2.1 Desenvolvimento do Dispositivo

Após a montagem da gravadora, deu-se inicio à montagem do dispositivo em que Microcontrolador Atmega8 se encontra.

O dispositivo foi montado em um *protoboard* usando 2 capacitores de 100 *n*F e 2 capacitores de 22 *p*F. Também foi incluído um conversor de tensão para que a entrada através de uma fonte de 12V fosse reduzida para 5V, que é a tensão de trabalho do microcontrolador. O diagrama de blocos do regulador de tensão é demonstrado na figura 3.6 .

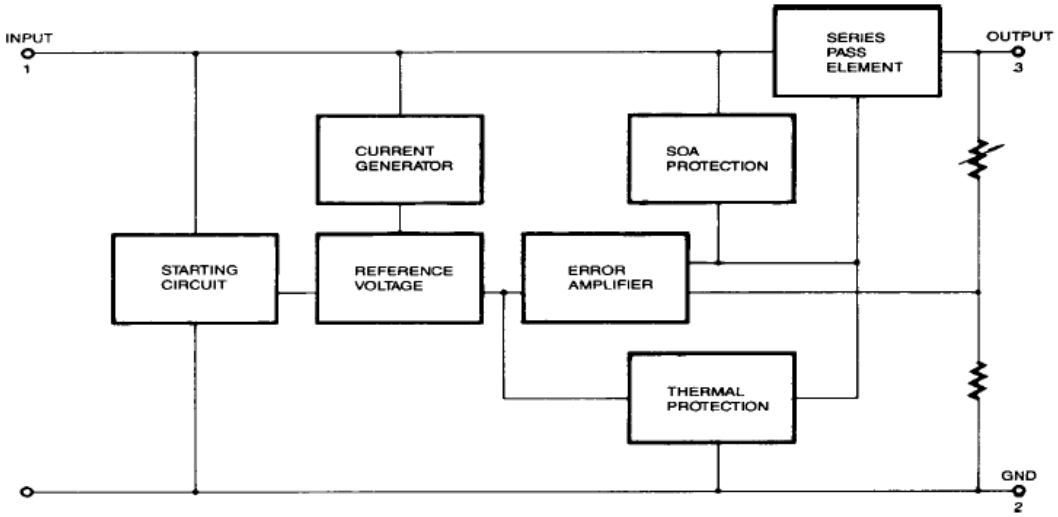


Figura 3.6 - Diagrama de blocos do regulador de tensão
LM7805CT[DATASHEETCATALOG,2010]

A fim de confirmar que existe corrente no circuito, foi inserido um *led* com um resistor de 400 Ω para preservar o *led* de um desgaste muito alto.

O *clock* representa a velocidade de operação do microcontrolador. O Atmega8 vem com um *clock* padrão de 1MHz, um *clock* muito baixo para a implementação do dispositivo projetado.

O Fuso é uma região de memória dentro do ATmega 8 onde se configura as funções do AVR, desse modo foi realizada a programação do fuso para 4MHz usando o clock interno.

O procedimento de programação do fuso foi realizado utilizado o programa AVRdude, que é um programa de gravação no ATmega8. No modo

terminal do computador, foi usada seguinte linha de comando para conectar o computador ao AVRdude: “-avr –p –atmega8 –c bsd”. Realizada a interface entre computador e AVR, foi inserida a linha de comando “w lsfuse 0 0xé3” para alterar o *clock* para os 4MHz.

Depois foi montado o dispositivo que representa a chave da fechadura. Primeiramente, seria utilizado um pen-drive, que é *um memory flash*, porém, tendo em vista o custo do projeto, foi substituído por uma Memória EEPROM. Após verificar a veracidade da chave será enviado um sinal para a fechadura, devido a baixa corrente enviada pelo microcontrolador foi utilizado um relé para ampliar o sinal para 1,5A e abrir a fechadura.

A figura 3.7 representa o bloco onde se encontra o transformador 220V para 12V juntamente com a fechadura elétrica e o dispositivo amplificador de tensão.

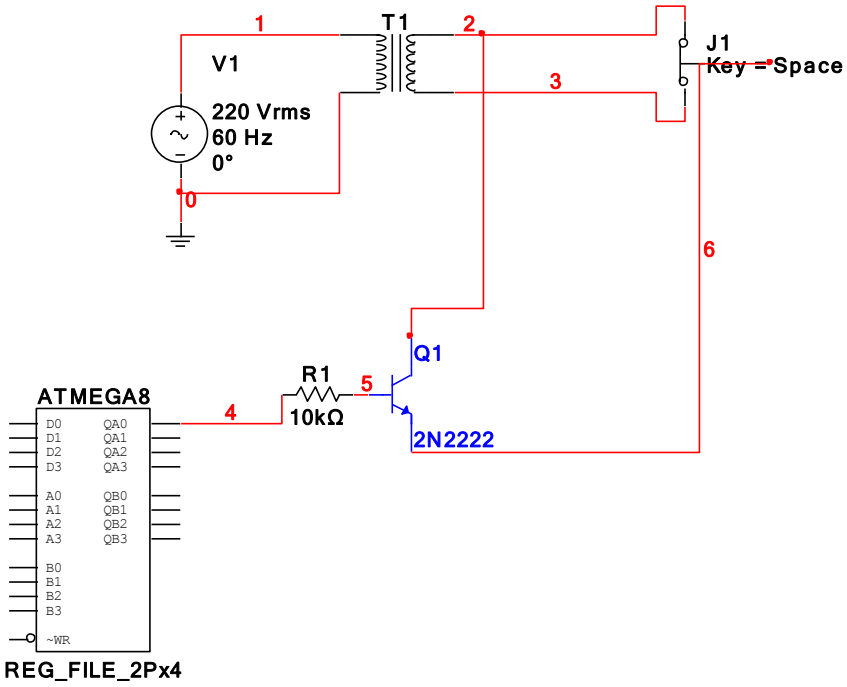


Figura 3.7 - Figura do diagrama elétrico da fechadura com o transformador

Para o acionamento da fechadura foi usado um transformador de 220V para 12V, pois essa fechadura elétrica necessita de 12V a 1,5A de corrente para seu perfeito funcionamento. Esse transformador possui uma fase e um neutro para, identificar na tomada a fase, foi usada uma chave de fenda com identificador de tensão digital.

No que diz respeito a sinal, como a resistência RL está conectada em série com o emissor, o modelo T do TBJ é o mais conveniente a ser utilizado. A figura 3.8 mostra o circuito equivalente para pequenos sinais do seguidor de emissor com o TBJ substituído pelo modelo T expandido com a inclusão de Ro.

A inspeção do circuito ilustrado na figura 3.8 revela que Ro aparece em paralelo com RL, portanto o circuito foi redesenhado para enfatizar esse ponto e para simplificar a análise.

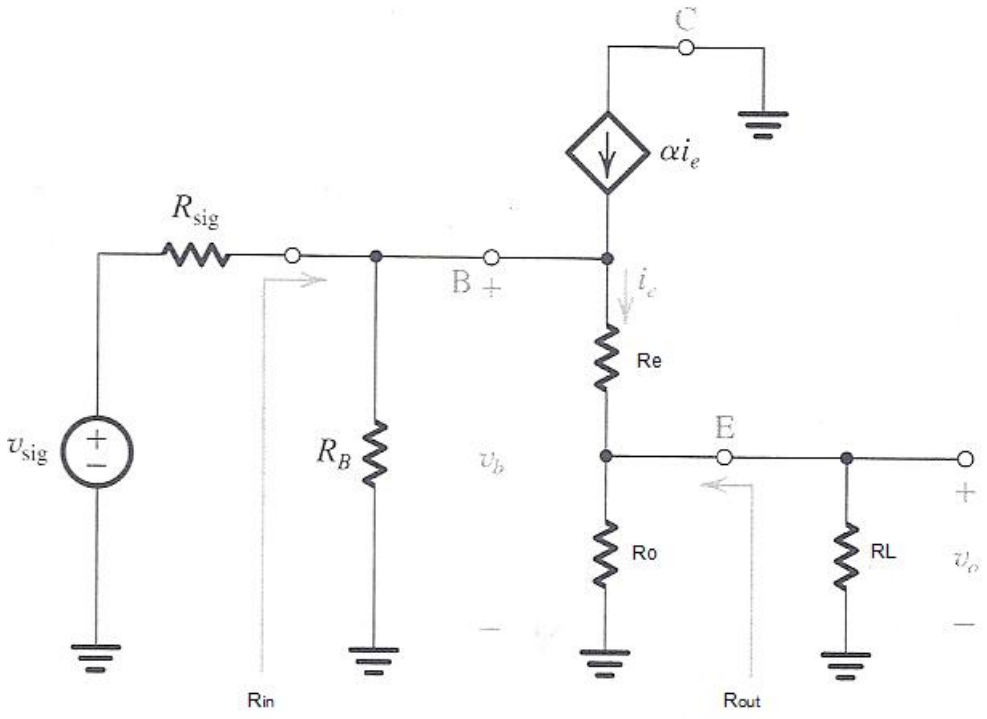


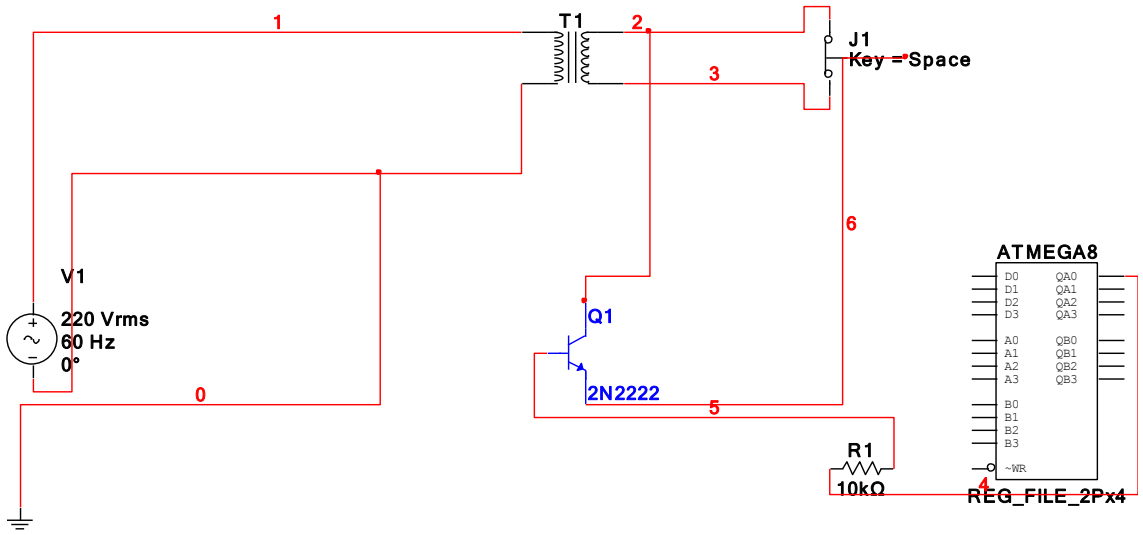
Figura 3.8 - Circuito equivalente para pequenos sinais [SEDRA e SMITH]

Em resumo, o seguidor de emissor exibe elevada resistência de entrada, baixa resistência de saída, um ganho de tensão que é menor e muito próximo da unidade e um ganho de corrente relativamente elevado. Ele é, portanto, mais adequado para aplicações nas quais elevadas resistências de fonte devam ser conectadas a uma carga de valor baixo – Isto é, assim como o último estágio ou estágio de saída em um amplificador multiestágio, em que o objetivo não é aumentar a tensão, mas, antes de mais nada, proporcionar baixa resistência de saída ao amplificador em cascata.

Ligado em conjunto com o relé está um diodo denominado comumente como um “diodo de roda livre”, e a sua utilização pode ser explicada de forma bem sucinta. O relé é um dispositivo bobinado e apresenta um comportamento basicamente indutivo. Logo, ao desligar o dispositivo ainda existirá uma corrente residual presente nessas bobinas, de forma inteiramente análoga as condições iniciais de um indutor ou capacitor de um circuito, ou seja, haverá uma corrente de um circuito de potência percorrendo a malha após cada chaveamento do dispositivo, e essa corrente pode causar a queima do micro controlador, dessa forma adicionamos um diodo que irá ocasionar um caminho de baixa impedância para a passagem da corrente, fala-se aqui de um desvio de corrente.

Na primeira versão foram utilizados dois leds um vermelho indicando que a chave não foi reconhecida e um verde indicando que a chave foi reconhecida. Tendo em vista uma modernização do projeto foi incluído um Display de LCD 20x2 de controladora HD44780.

As diagramas elétricos 3.9 a 3.12 demonstram a construção do dispositivo, em sua primeira versão, ainda com o implemento dos leds, que como mencionado anteriormente foram substituídos pelo display de LCD.



3.9 – Diagrama elétrico e componentes que fazem parte do dispositivo.

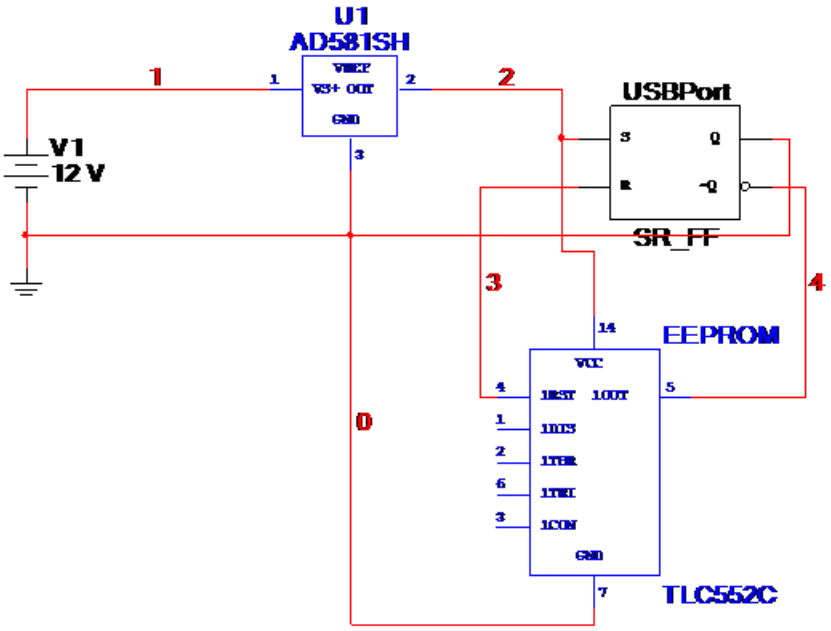


Figura 3.10 – Diagrama elétrico da alimentação de 12V ligada

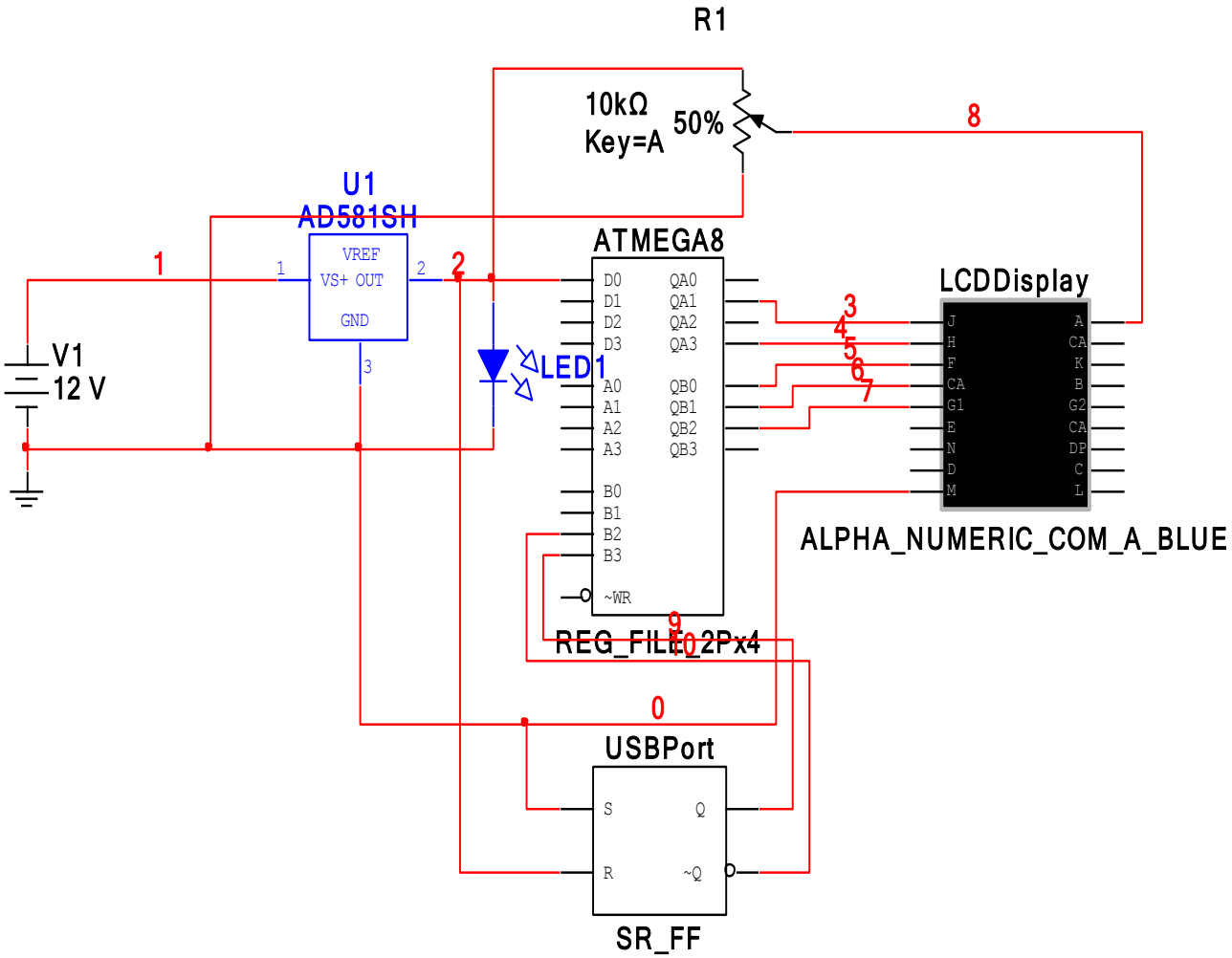
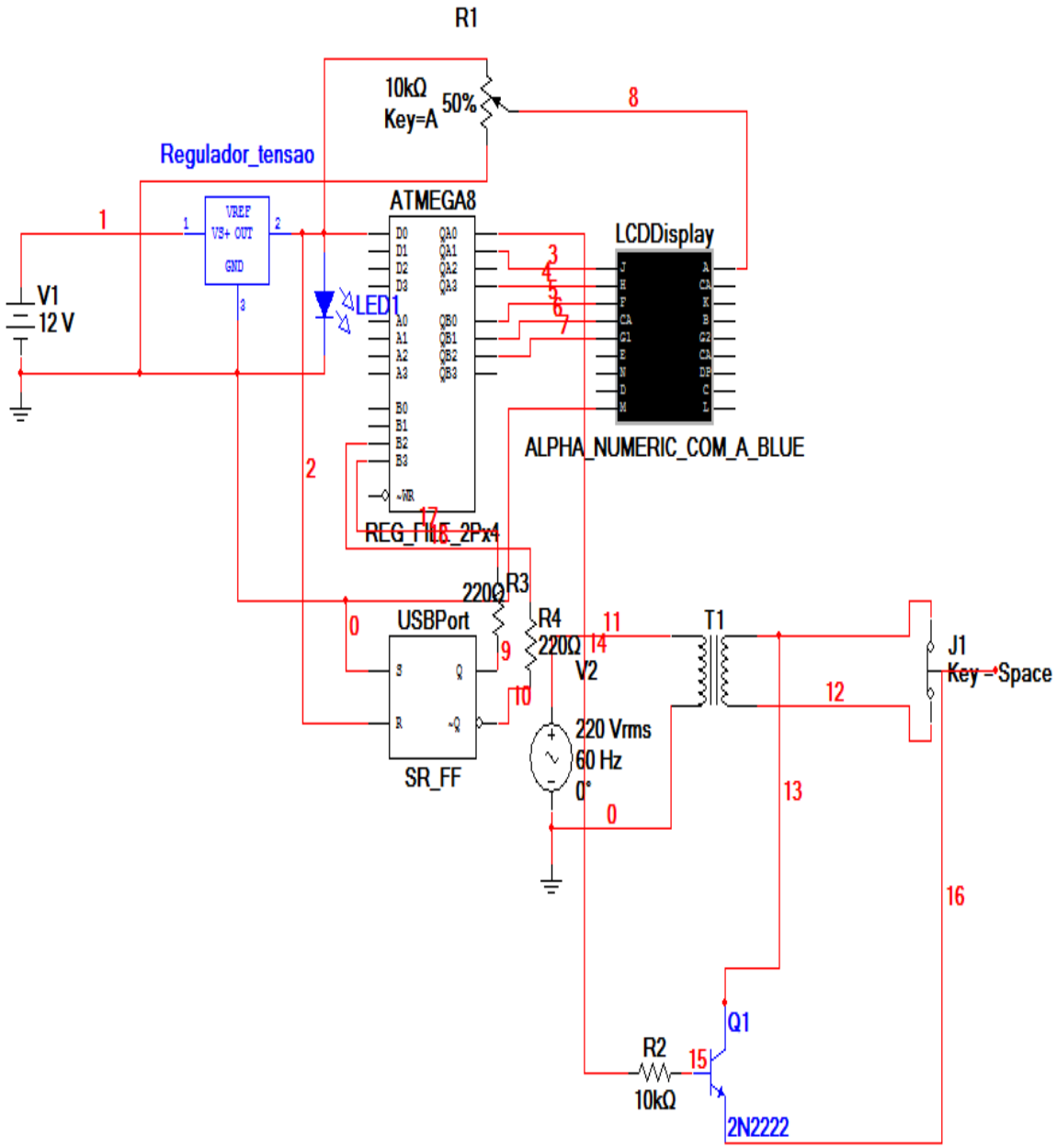


Figura 3.11 – Diagrama elétrico



3.12 Diagrama elétrico de todo o circuito

3.2.2 Desenvolvimento do Display

No projeto foi usado um display de LCD modelo 20x2 ,que possui uma controladora HD44780, com memória RAM interna para armazenar os dados e memória ROM para guardar as formas dos caracteres usadas na tela do display.

Esse modelo de mostrador possui 192 caracteres, interface paralela de 4 vias e 8 vias e opera tanto em modo serial quanto em modo de mapeamento de memória, possuindo também um circuito de controle. O circuito de controle permite controlar o LCD sem que para isso necessite comandar o LCD pixel por pixel.

Primeiramente, houve a solda dos fios no display, após soldar todos os fios foi feito a ligação no protoboard do pino 1 que é o terra, o 2 VCC que está ligado em 5v, o pino 3 é o contraste que serve para ajustar a luminosidade do display que foi ligado em série com o potenciômetro de 10k, o pino 4 seletor de registro ele determina se ele irá mandar um dado ou instrução, 5 read/write que determina se a próxima operação é de escrita ou gravação, 6 enable que é a chave que habilita o circuito, que funciona como se fosse um clock ativando assim o LCD para receber os dados. No display, do pino 11 ao 14, foram ligadas as entradas saídas digitais do ATmega8, as portas 2, 3 4, 5, de acordo com o quadro 3.1.

Tipo	Pino LCD	Pino AVR
D4	11	2
D5	12	3
D6	13	4
D7	14	5
RS	4	6
Rw	5	11
E	6	14

Quadro 3.1 –Relação de pinos do LCD para o AVR

Foi necessário ajustar o header que é um arquivo de cabeçalho .h onde se configura as funções do display.

A figura 3.13 ilustra o projeto em seu pleno funcionamento, onde é possível identificar as partes descritas e elaboradas.

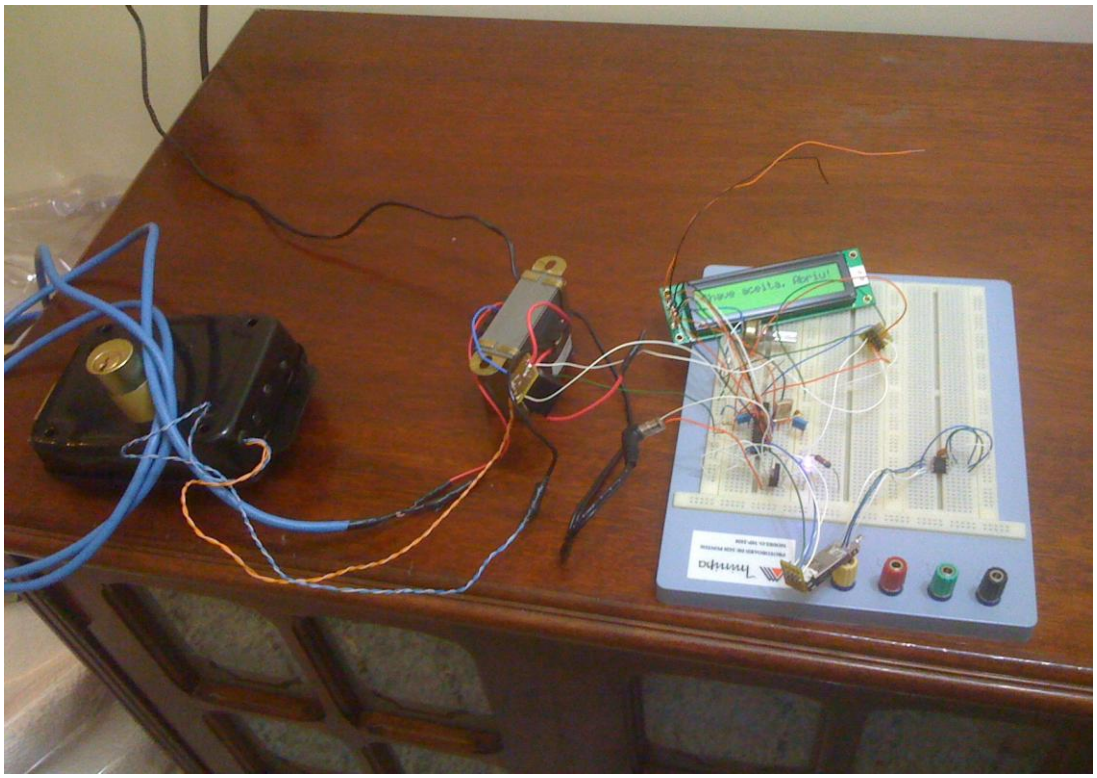


Figura 3.13 - Dispositivo concretizado e em pleno funcionamento.

3.2 Desenvolvimento do Software

Esse tópico é dedicado a demonstrar os passos adotados para o desenvolvimento do software que compõem o dispositivo de abertura eletrônico criptografado. Os arquivos abaixo compõem uma série de códigos que são responsáveis pelo funcionamento do dispositivo e estão representados no apêndice A.

O arquivo denominado de EEprom.h é um arquivo de endereçamento da memória EEPROM, que endereça as funções principais da memória: inicialização, leitura, escrita, transmissão e tratamento de erro, como o processo de *restart.* Esse arquivo é pequeno, mas de muita importância para o dispositivo, pois é nele que se encontra a chave criptografada, ou seja, a chave da fechadura. Uma observação importante é que esse arquivo fosse um componente ao mesmo tempo estável e confiável, mais do que isso, durante sua implementação se mostrou bastante robusto em relação a programação inserida e aos teste realizados.

O primeiro programa a ser inserido no Atmega8 foi chamado de EEpromLe. Esse programa consiste em reconhecer, ler a memória EEPROM e jogar o sinal nas portas portA e portB do ATmega8 para piscar *Leds*. Esse programa serviu como teste de identificação de erros durante todo o processo de construção do dispositivo, pois foi intensamente usado para testar os componentes e suas conexões bem como seu normal funcionamento. Ainda foi por diversas vezes capaz de indicar que determinado componente do dispositivo havia queimado ou simplesmente estava com mal funcionamento.

Foi também necessário um código responsável pelo controle do *Display* de LCD 20x2. Esse controle é realizado por intermédio de controladora HD44780, que realiza um processo de controle pixel a pixel, pois caso contrário seria necessário controlá-los individualmente. Para tanto, foi usada a biblioteca do autor Peter Fleury.[FLEURY,2010]

Por fim, foi feito o programa central do projeto, que controla todas as chamadas do dispositivo. A função que leva o nome de *playfair* é chamada em primeiro lugar para gerar a chave, que em seguida será gravada na memória EEPROM, para isso foram incluídas as funções *toupper* .

```
inline int toupper(int ch) {
  if ( (unsigned int)(ch - 'a') < 26u )
    ch += 'A' - 'a';
  return ch;
```

Essa função foi usada para verificar se a entrada da chave estava usando maiúscula ou minúscula.

```
int __isalpha_ascii ( int ch );
int __isalpha_ascii ( int ch ) {
    return (unsigned int)((ch | 0x20) - 'a') < 26u;

int isalpha ( int ch ) __attribute__((weak,alias("__isalpha_ascii")));
```

Esta função, chamada de isalpha é chamada para verificar se a entrada possui apenas letras em sua total abrangência, pois a criptografia de playfair só usa letras em sua formação sendo assim necessária a verificação.

```
char *alph = "ABCDEFGHIJKLMNOPQRSTUVXYZ";
        char keycopy[25], pt0, pt1;
        char *ptcopy = ct;
        int i, j, k, lk, lp, rpt0, rpt1, cpt0, cpt1;
        int alphabet [25], row[25], col[25];
```

Essa função além de definir todas as letras necessárias para a montagem da matriz 5x5, ainda pode definir quais as letras usadas. Este projeto que a letra “W” não é usada, a escolha é aleatória. E este fato influencia na questão da segurança ainda que alguém soubesse que tipo de criptografia foi usada, ainda teria que descobrir qual letra foi omitida, o que aumento ainda mais a segurança, pois aumenta o número de combinações. Essa função também monta a matriz 5x5.

```
for (i = 0; i < 25; i++)
        {
                row[i] = alphabet[i] / 5;
                col[i] = alphabet[i] % 5;
        }
```

Essa função faz a leitura de toda a matriz

A gravação na memória EEPROM se fará toda vez que o dispositivo for ligado na energia, depois disso o programa não mais será gravado a chave e apenas será feito a leitura para verificação se a chave é verdadeira, fazendo a chamada da função de decripta. Após isso, será enviado um sinal para o relé amplificar o sinal e assim, irá abrir a fechadura. O sinal enviado ao relé foi controlado para que esse fosse enviado uma vez.

Durante todo esse processo foi sendo chamado simultaneamente a função do display para ir incluindo na tela do LCD as mensagens: Favor inserir chave, Chave aceita, Abriu!, Iniciando gravação.

A figura 3.14 apresenta o fluxograma dos passos do programa.

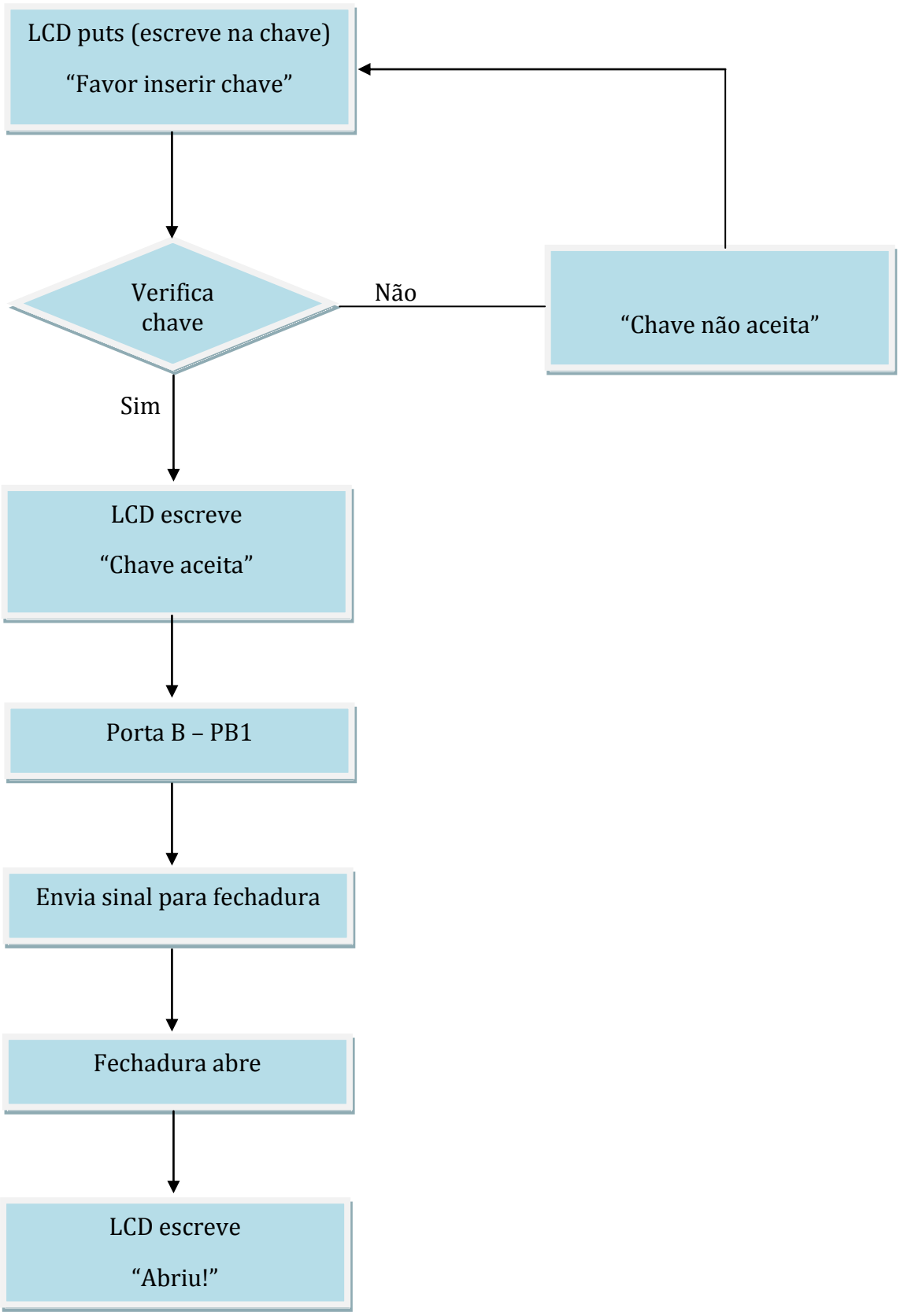


Figura 3.14-Fluxograma demonstrando os passos da programação.

A figura 3.15 apresenta o fluxograma dos passos da gravação.

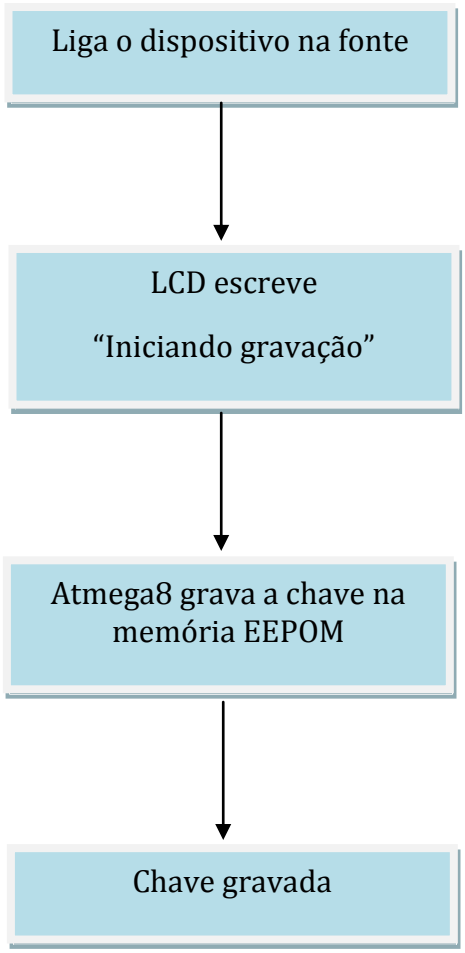


Figura 3.15-Fluxograma demonstrando os passos de gravação.

CAPÍTULO 4 – TESTES

Esse capítulo destina-se a relatar os testes executados durante a construção do dispositivo.

4.1 Testes dos Componentes

Com a inserção do regulador de tensão foi determinado que duas trilhas do *protoboard* fossem alimentadas com 5v e outras duas seriam ligadas ao terra.

Com a constatação de que o display e o microcontrolador ATmega8 usam a tensão de 5v, foi utilizado um voltímetro(multímetro) para verificar a tensão real nas trilhas do *protoboard* e desta maneira.

Foram realizados testes de gravação/escrita na EEPROM, utilizando o protocolo *Two Wire Interface* (TWI, ou i2c da Phillips), que permite comunicação utilizando apenas duas linhas - exatamente a quantidade disponível na porta USB. Após falha na escrita, percebeu-se que o clock fornecido pela linha TWI estava abaixo da especificação da memória 24c64 da Atmel.

4.2 Testes do AVR

Partiu-se então para reprogramação dos fusos de configuração do TWI e foi descoberto que o utilizado no AVR - 1MHz - seria insuficiente. Então, foi necessário reprogramar os fusos de *clock* do AVR para que este utilizasse o cristal externo de 16MHz, mas a operação não foi bem sucedida. O AVR começou a operar de modo imprevisível, apesar de ainda funcionar. Outro AVR foi utilizado para dar continuidade ao projeto. O novo AVR foi reprogramado para que os fusos *clock* interno de 4MHz.

Foi realizado teste de gravação/leitura, após várias tentativas de

configuração. O defeito era a troca das linhas de dados/*clock* entre o dispositivo de memória EEPROM e o AVR.

Foi realizado teste de acionamento do relé, onde se verificou a baixa capacidade de fornecimento de corrente do AVR, o que implicou na utilização de um transistor NPN BC548 como amplificador de corrente. Por fim, foi realizado teste do acionamento da fechadura, utilizando o relé e o AVR.

Após ter ligado o display foi feito um teste gravando no microcontrolador um programa que dispara um cronometro. A gravação no AVR é realizada, sempre, pelo programa avrdude.

4.3 Testes de Software

Na parte de programação foram realizados testes que põe a prova as condições lógicas contidas num módulo das funções que compõem a estrutura lógica. Esse teste concentrou-se cada condição do programa. O propósito foi verificar a condição e detectar não somente erros nas condições de uma determinada função, mas também outros erros nas outras funções.

Foi selecionado o método de teste de fluxo de dados no qual várias ramificações das funções foram selecionadas para verificar as localizações das definições e usos de variáveis. As estratégias de fluxo de dados são úteis para selecionar caminhos de teste de um programa que contenha instruções de laços

O teste nos laços foi muito importante para a verificação do correto funcionamento do dispositivo e se concentrou em grande parte na validade das construções de laços.

Após isto, foi verificado se havia funções incorretas ou ausentes e erros de interface, na medida em que as funções iriam sendo executadas foi verificando-se a existência da correta correlação entre a mensagem gravada e a que estava sendo mostrada no visor do display de LCD, assim foram feitos inúmeros testes e correções para que se chegasse ao resultado desejado pelo dispositivo, que foi de permitir mensagens concisas e coerentes com o processo executado.

Foram executados testes nas estruturas de dados para que o sistema não se perdesse em estruturas de laços infinitos, já que grande parte das funções possuem laços e o dispositivo deve estar sempre pronto para leitura da chave e sua verificação, bem como possuir a velocidade de resposta dada ao usuário já que se trata também de um requisito de segurança.

Foram executados testes para verificação dos erros de inicialização e término das várias funções, pois o sistema não tem um fim programado, a cada nova leitura e, por conseguinte verificação da chave, sendo essa positiva, foi enviado um único sinal para abertura da fechadura, bem como em caso negativo a mensagem retorna para a função de leitura da chave.

Foi certificado que a porta paralela estava operando no modo padrão (SPP, ou *Standard Parallel Port*), da seguinte forma: foi reiniciado o microcomputador para entrar no programa de configuração da BIOS, se a porta paralela estivesse configurada para qualquer outro modo, a configuração deveria ser alterada para SPP, para então sair do programa monitor da BIOS, salvando a nova configuração e então reiniciado o microcomputador para que as alterações entrassem em vigor.

Cabe acrescentar, que nunca se deve desconectar a gravadora do circuito (ou do PC) com o Atmega8 ligado, pois pode gerar gravação no registro LFUSE do valor 00h (hexadecimal), o que leva o microcontrolador a operar somente com fonte de relógio externa. Isto impossibilita a gravação do ATmega8 com qualquer outra fonte de relógio e para recuperar o microcontrolador é necessário construir um oscilador externo a 1MHz, conectá-lo ao ATmega8, e usar o avrdude.exe no modo unsafe para alterar o conteúdo de LFUSE (e possivelmente HFUSE).

Foi realizado testes com um *pendrive* que não contivesse a chave e foi verificado que a fechadura não abriu, após tal constatação foi inserida a chave correta e a fechadura abriu, aparecendo a mensagem no visor do display de LCD, assim foi verificado que as mensagens foram sendo mostradas de acordo com o evento ocorrido.

CAPÍTULO 5 – CONCLUSÃO

5.1 Conclusão

Neste projeto foi desenvolvido um protótipo de fechadura eletrônica criptografada, com intuito de criar uma solução alternativa aos dispositivos já existentes no mercado, como forma de incrementar um dos mecanismos mais antigos utilizados até hoje: as tradicionais chaves e fechaduras.

O projeto cumpriu com o objetivo de trazer uma inovação para o nicho de chaves e fechaduras, concretizando-se no mecanismo de abertura eletrônico criptografado, de acordo com a motivação e os objetivos propostos inicialmente.

Desse modo, com o apoio dos conhecimentos obtidos em diversas disciplinas do curso de Engenharia da Computação, das pesquisas bibliográficas e dos testes realizados foi possível construir um dispositivo estável, capaz de abrir uma fechadura utilizando a criptografia como meio de conferir maior segurança ao sistema.

Ademais, esse projeto traz uma inovação frente aos dispositivos presentes no mercado, que é a utilização da abertura via conexão USB como forma de agregar tecnologia e tornar a interface do dispositivo com o ser humano mais eficiente.

A implementação de um *display* veio a tornar a complexidade do sistema não se encontra aparente ao usuário, pois representa uma interface amigável, onde as informações são mostradas de acordo com as etapas do funcionamento do dispositivo de abertura eletrônico criptografado.

Assim, impulsionado pelas necessidades de segurança e comodidade, com o custo de hardware e software relativamente baixo mostrou-se possível a implantação do dispositivo em residências e automóveis pertencentes as diversas camadas sociais.

5.2 Projetos futuros

Visando a própria melhoria do dispositivo, seria interessante a melhoria do código criptográfico, com o objetivo de melhorar a segurança do dispositivo.

Implementar uma função de voz, bem como hardware para usuários cegos e incapacitados de ler a mensagem contida no *display* pudessem interagir com o dispositivo, bem como um sensor de presença para ligar o dispositivo somente quando houvesse alguém na porta, evitando assim gastos de energia.

Para uma melhoria desse projeto poderia ser inserida uma câmera conectada à internet para que pudesse filmar a pessoa que está tentando abrir a fechadura como forma de manter um arquivo de gravação, com vistas a aumentar a segurança.

CAPÍTULO 6 – REFERÊNCIAS BIBLIOGRÁFICAS

ALECRIM, Emerson. Artigo: Criptografia. Publicado em 12/08/2005 - Atualizado em 11/07/2009. Disponível em: http://www.infowester.com/criptografia.php. Acessado em: 07 de abril de 2010.

ATMEL, 2008. Nota técnica Atmega8. Disponível em: www.atmel.com. Acessado em: março de 2010.

BOUGART, Theodore F. Jr. Dispositivos de Circuitos Eletrônicos – Volume II. São Paulo: Makron Books, 2001.

DEITEL, HM e P.J., Como Programar em C. 2 edição. LTC Rio de Janeiro 1999.

ELETRONICADIDATICA, 2010. Disponível em: http://www.eletronicadidatica.com.br/

FLEURY, Peter. Biblioteca de Display. Disponível em :http://jump.to/Fleury acessado em 30 de abril de 2010.

HINZ, Marco Antônio. Um estudo descritivo de novos algoritimos de criptografia. Pelotas, 2000. Disponível em: http://www.ufpel.edu.br/prg/sisbi/bibct/acervo/info/2000/Mono-MarcoAntonio.pdf

INMETRO. Sistema Internacional de Unidades - SI.. 8. ed.(rev.) Rio de Janeiro, 2007. Disponível em: http://www.inmetro.gov.br/infotec/publicacoes/Si.pdf. Acessado em: 01 de junho de 2010.

MAURÍCIO, Douglas Morais. Monografia: INSTALAÇÃO DE CAPACITORES PARA REDUÇÃO DAS PERDAS EM UMA REDE DE DISTRIBUIÇÃO DE ENERGIA ELÉTRICA VIA ALGORITMOS GENÉTICOS. Ouro Preto, 2007.

MENDES, Aline Veloso. Monografia: Estudo de criptografia com chave pública baseadas em curvas elípticas, 2007 in GARFINKEL e SPAFFORD, Comércio e Segurança na Web. São Paulo: Market Press,1999. Disponível em: http://www.ccet.unimontes.br/arquivos/monografias/261.pdf

MENDES, Aline Veloso. Monografia: Estudo de criptografia com chave pública baseadas em curvas elípticas, 2007 in VOLPI, Marlon M. Assinatura Digital, aspectos Técnicos, Práticos e Legais. Rio de Janeiro: Axcel Books, 2001.

OLIVEIRA, Anderson Gomes. Monografia: CRIPTOGRAFIA USANDO PROTOCOLOS QUÂNTICOS. Minas Gerais, 2008. Disponível em: http://www.ginux.ufla.br/files/mono-AndersonOliveira.pdf

PINTO E SOUZA apud TOLEDO

POZZO, Douglas e VITTI, Diego. SEMINÁRIO ASSEMBLY: Arquitetura PIC. Florianópolis – Santa Catarina, Fevereiro – 2007. Disponível em: http://www.lisha.ufsc.br/teaching/sys/ine5309-2006-2/work/g2/monografia.pdf

ROSÁRIO, João Mauricio. Princípios da Mecatrônica. 1 ed. São Paulo: PEARSON, 2005.

SCHILDT, Herbert. C Completo e Total. 3 ed.Sao Paulo: Pearson,1997.

SEDRA E SMITH, A. S. SEDRA e K. C. SMITH. Microeletrônica 5 ed. São Paulo: Makron Books, 1999.

SILVA JUNIOR, Vidal Pereira. Microcontroladores. 1 ediçao. São Paulo: Érica 1998.

STALLINGS, William. Criptografia e segurança de redes – princípios e práticas. 4 ed. São Paulo: Pearson, 2008.

TORRES, Gabriel, 2006. ROM. Disponível em: http://www.clubedohardware.com.br/dicionario/termo/239. Acessado em março 2010.

TORRES, Gabriel. Hardware Curso Completo. 4 edição. Rio de Janeiro: ASCELBOOKS, 2001.

VENTURI, Paulo Arthur, 2009 in AXELSON, J. Parallel Port Complete: Programming, Interfacing, & Using the PC's Parallel Printer Port. Lakeview Research; Pap/Dis edition, 1997.

ZELENOVSKY, Ricardo E MENDONÇA, Alexandre. Artigo: INTRODUÇÃO AOS SISTEMAS EMBUTIDOS. Disponível em: http://www.mzeditora.com.br/artigos/embut.htm. Acessado em: maio de 2010.

SOUZA, Vitor Amadeu. Comunicação Serial com o AVR ATMEGA8. Disponível em: http://www.cerne-tec.com.br/serialavr.pdf. Acessado em: junho de 2010.

Datasheet do Regulador de Tensão LM785. Disponível em: http://www.datasheetcatalog.org/datasheet/fairchild/LM7805.pdf. Acessado em: abril de 2010.

APÊNDICE A – PROGRAMA DESENVOLVIDO

O código representado no arquivo EEprom.h está demonstrado abaixo:

```
// EEPROM functions
// NOTE: These are the functions only for a 24C64!
// If you use a smaller/bigger E2PROM, the I2C communication works different!
// Check the datasheet of the EEPROM you'd like to use and modify the
// functions. i.e. smaller I2C EEPROM's already have the high-byte address
// information in the inital byte (which is 0xa0 or 0xa1 in this case).

#include "e2prom.h"

uint8_t twst;

void eeprom_init(void) {
  /* initialize TWI clock: 100 kHz clock, TWPS = 0 => prescaler = 1 */
#if defined(TWPS0)
  /* has prescaler (mega128 & newer) */
  TWSR = 0;
#endif
  TWBR = 10;
  }
int eeprom_read_byte(uint16_t eeaddr, char *buf)
{
  uint8_t n = 0;

restart:
  if (n++ >= MAX_TRIES)
    return -1;

begin:
        // send start cond.
        TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
        while (!(TWCR & (1 << TWINT)));
```

```
        if (TW_STATUS == TW_MT_ARB_LOST) goto begin;
        if ( (TW_STATUS != TW_REP_START) && (TW_STATUS != TW_START)) {
                return -1;
        }

        // send 0xa0
        // 0xa0 = 1010 000 0
        // 4 bits:   <a..device-indentifier>
        // 3 bits:   <device-address set with chip pins>
        // last bit: <0..write>
        TWDR = 0xa0;
        TWCR = (1 << TWINT) | (1 << TWEN);
        while (!(TWCR & (1 << TWINT)));
        if (TW_STATUS == TW_MT_SLA_NACK) goto restart;
        if (TW_STATUS == TW_MT_ARB_LOST) goto begin;
        if (TW_STATUS != TW_MT_SLA_ACK) goto error;

        // send low 8 bits of eeaddr
        TWDR = eeaddr;
        TWCR = (1 << TWINT) | (1 << TWEN);
        while (!(TWCR & (1 << TWINT)));
        if (TW_STATUS == TW_MT_DATA_NACK) goto restart;
        if (TW_STATUS == TW_MT_ARB_LOST) goto begin;
        if (TW_STATUS != TW_MT_DATA_ACK) goto error;

        // send high 8 bits of eeaddr
        TWDR = eeaddr << 8;
        TWCR = (1 << TWINT) | (1 << TWEN);
        while (!(TWCR & (1 << TWINT)));
        if (TW_STATUS == TW_MT_DATA_NACK) goto restart;
        if (TW_STATUS == TW_MT_ARB_LOST) goto begin;
        if (TW_STATUS != TW_MT_DATA_ACK) goto error;

        // send start cond.
```

```
	TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
	while (!(TWCR & (1 << TWINT)));
	if (TW_STATUS == TW_MT_ARB_LOST) goto begin;
	if ( (TW_STATUS != TW_REP_START) && (TW_STATUS != TW_START)) {
		return -1;
	}

	// send 0xa1
	// 0xa0 = 1010 000 1
	// 4 bits:   <a..device-indentifier>
	// 3 bits:   <device-address set with chip pins>
	// last bit: <1..read>
	TWDR = 0xa1;
	TWCR = (1 << TWINT) | (1 << TWEN);
	while (!(TWCR & (1 << TWINT)));
	if (TW_STATUS == TW_MR_SLA_NACK) goto quit;
	if (TW_STATUS == TW_MR_ARB_LOST) goto begin;
	if (TW_STATUS != TW_MR_SLA_ACK) goto error;

	// start read transmission
	TWCR = (1 << TWINT) | (1 << TWEN);
	while (!(TWCR & (1 << TWINT)));

	switch ((twst = TW_STATUS)) {
		case TW_MR_DATA_NACK:
			// FALLTHROUGH
		case TW_MR_DATA_ACK:
			*buf = TWDR;
			break;
		default:
			goto error;
	}

quit:
```

```
        //stop condition
        TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
        return 1;

error:
        //stop condition
        TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
        return -1;
}

int eeprom_write_byte(uint16_t eeaddr, char buf) {
  uint8_t n = 0;

restart:
  if (n++ >= MAX_TRIES)
    return -1;
begin:

        // start cond.
        TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
        while (!(TWCR & (1 << TWINT)));
        if (TW_STATUS == TW_MT_ARB_LOST) goto begin;
        if ( (TW_STATUS != TW_REP_START) && (TW_STATUS != TW_START)) {
                return -1;
        }

        // send 0xa0
        // 0xa0 = 1010 000 0
        // 4 bits:   <a..device-indentifier>
        // 3 bits:   <device-address set with chip pins>
        // last bit: <0..write>
        TWDR = 0xa0;
        TWCR = (1 << TWINT) | (1 << TWEN);
        while (!(TWCR & (1 << TWINT)));
```

```
        if (TW_STATUS == TW_MT_SLA_NACK) goto restart;
        if (TW_STATUS == TW_MT_ARB_LOST) goto begin;
        if (TW_STATUS != TW_MT_SLA_ACK) goto error;

        // send low 8 bits of eeaddr
        TWDR = eeaddr;
        TWCR = (1 << TWINT) | (1 << TWEN);
        while (!(TWCR & (1 << TWINT)));
        if (TW_STATUS == TW_MT_DATA_NACK) goto quit;
        if (TW_STATUS == TW_MT_ARB_LOST) goto begin;
        if (TW_STATUS != TW_MT_DATA_ACK) goto error;

        // send high 8 bits of eeaddr
        TWDR = eeaddr << 8;
        TWCR = (1 << TWINT) | (1 << TWEN);
        while (!(TWCR & (1 << TWINT)));
        if (TW_STATUS == TW_MT_DATA_NACK) goto quit;
        if (TW_STATUS == TW_MT_ARB_LOST) goto begin;
        if (TW_STATUS != TW_MT_DATA_ACK) goto error;
        // put byte into data register and start transmission
        TWDR = buf;
        TWCR = (1 << TWINT) | (1 << TWEN);
        while (!(TWCR & (1 << TWINT)));
        if (TW_STATUS != TW_MT_DATA_ACK) goto error;
quit:
        // send stop condition
        TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
        return 1;
error:
        // send stop condition
        TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
        return -1;
}
```

Segue o código do EEpromLe:

```
/*****************************************
* Chip type           : ATmega8
* Clock frequency     : 2457600Hz
*****************************************/
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <inttypes.h>
#include <avr/iom8.h>

#include "e2prom.h"

//#define F_OSC 2457600                    /* oscillator-frequency in Hz */
#define UART_BAUD_RATE 9600
#define                          UART_BAUD_CALC(UART_BAUD_RATE,F_OSC)
((F_OSC)/((UART_BAUD_RATE)*16l)-1)

void delay_ms(unsigned short ms) {
        unsigned short outer1, outer2;
        outer1 = 200;
        while (outer1) {
```

```
                outer2 = 1000;
                while (outer2) {
                        while ( ms ) ms--;
                        outer2--;
                }
                outer1--;
        }
}

void usart_putc(unsigned char c) {
   // wait until UDR ready
        while(!(UCSRA & (1 << UDRE)));
        UDR = c;    // send character
}

void uart_puts (char *s) {
        //  loop until *s != NULL
        while (*s) {
                usart_putc(*s);
                s++;
        }
}
```

```
void init(void) {
	// set baud rate
	UBRRH = (uint8_t)(UART_BAUD_CALC(UART_BAUD_RATE,F_OSC)>>8);
	UBRRL = (uint8_t)UART_BAUD_CALC(UART_BAUD_RATE,F_OSC);
	// Enable receiver and transmitter; enable RX interrupt
	UCSRB = (1<<RXEN)|(1<<TXEN)|(1<<RXCIE);
	//asynchronous 8N1
	UCSRC = (1<<URSEL)|(3<<UCSZ0);
}

// INTERRUPT can be interrupted
// SIGNAL can't be interrupted
SIGNAL (SIG_UART_RECV) { // USART RX interrupt
	unsigned char c;
	c = UDR;
	usart_putc(c);
}

int main(void) {
	char buffer;
	char buffer2;
	init();
```

```
sei();


eeprom_init();
DDRB = _BV(DDB1)|_BV(DDB0);



// blink LED
// enable  PD5 as output
DDRD |= (1<<PD5);
while (1) {
        // PIN5 PORTD set -> LED on
        PORTD |= (1<<PD5);
        delay_ms(500);
        // PIN5 PORTD clear -> LED off
        PORTD &= ~(1<<PD5);
        delay_ms(500);


                // read first character from eeprom
        eeprom_read_byte(0, &buffer2);
        if(buffer2!=0)
                PORTB = _BV(PB0);
// read second character from e2prom
        eeprom_read_byte(1, &buffer2);
```

```
            if(buffer2!=0)
                    PORTB |= _BV(PB1);
            }
        return 0;
}
```

O próximo código é responsável pelo controle do *Display de LCD* 20x2.

```
        Title    :   HD44780U LCD library
        Author:    Peter Fleury <pfleury@gmx.ch>
        http://jump.to/fleury
        File:       $Id: lcd.c,v 1.13.2.2 2004/02/12 21:08:25
        peter Exp $
        Software:  AVR-GCC 3.3
        Target:    any AVR device, memory mapped mode only for
        AT90S4414/8515/Mega

        DESCRIPTION
            Basic routines for interfacing a HD44780U-based
        text lcd display

            Originally based on Volker Oth's lcd library,
            changed lcd_init(), added additional constants for
        lcd_command(),
```

```
      added 4-bit I/O mode, improved and optimized code.

      Library can be operated in memory mapped mode
     (LCD_IO_MODE=0) or in

      4-bit IO port mode (LCD_IO_MODE=1). 8-bit IO port

*********************************************************


*******************/

#include <inttypes.h>

#include <avr/io.h>

#include <avr/pgmspace.h>

#include "lcd.h"

/*

** constants/macros

*/

#define PIN(x) (*(&x - 2))  /* address of data direction
register of port x */

#define DDR(x) (*(&x - 1))  /* address of input register
of port x          */

#if LCD_IO_MODE

#define lcd_e_delay()   __asm__ __volatile__( "rjmp 1f\n


1:" );

#define lcd_e_high()    LCD_E_PORT  |=  _BV(LCD_E_PIN);

#define lcd_e_low()     LCD_E_PORT  &= ~_BV(LCD_E_PIN);

#define lcd_e_toggle()  toggle_e()

#define lcd_rw_high()   LCD_RW_PORT |=  _BV(LCD_RW_PIN)

#define lcd_rw_low()    LCD_RW_PORT &= ~_BV(LCD_RW_PIN)

#define lcd_rs_high()   LCD_RS_PORT |=  _BV(LCD_RS_PIN)

#define lcd_rs_low()    LCD_RS_PORT &= ~_BV(LCD_RS_PIN)

#endif
```

```
#if LCD_IO_MODE
#if LCD_LINES==1
#define LCD_FUNCTION_DEFAULT    LCD_FUNCTION_4BIT_1LINE
#else
#define LCD_FUNCTION_DEFAULT    LCD_FUNCTION_4BIT_2LINES
#endif
#else
#if LCD_LINES==1
#define LCD_FUNCTION_DEFAULT    LCD_FUNCTION_8BIT_1LINE
#else
#define LCD_FUNCTION_DEFAULT    LCD_FUNCTION_8BIT_2LINES
#endif
#endif


/*
** function prototypes
*/
#if LCD_IO_MODE
static void toggle_e(void);
#endif

/*
** local functions
*/



/************************************************************
```

```
*****************
 delay loop for small accurate delays: 16-bit counter, 4

cycles/loop
************************************************************

***************/
static inline void _delayFourCycles(unsigned int __count)
{
    if ( __count == 0 )
        __asm__ __volatile__( "rjmp 1f\n 1:" );    // 2

cycles
    else
        __asm__ __volatile__ (
             "1: sbiw %0,1" "\n\t"
             "brne 1b"                          // 4

cycles/loop
             : "=w" (__count)
             : "0" (__count)
            );
}



/*****************************************************

*****************
delay for a minimum of <us> microseconds
the number of loops is calculated at compile-time from

MCU clock frequency
```

```
**********************************************************

***************/
#define delay(us)  _delayFourCycles( ( ( 1*(XTAL/4000) )

*us)/1000 )


#if LCD_IO_MODE
/* toggle Enable Pin to initiate write */
static void toggle_e(void)
{
    lcd_e_high();
    lcd_e_delay();
    lcd_e_low();
}
#endif


/*********************************************************

*****************
Low-level function to write byte to LCD controller
Input:    data   byte to write to LCD
          rs     1: write data
                 0: write instruction
Returns:  none
*********************************************************

***************/
#if LCD_IO_MODE
```

```
static void lcd_write(uint8_t data,uint8_t rs)
{
   unsigned char dataBits ;



   if (rs) {   /* write data        (RS=1, RW=0) */
     lcd_rs_high();
   } else {    /* write instruction (RS=0, RW=0) */
     lcd_rs_low();
   }
   lcd_rw_low();


   if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && (

&LCD_DATA1_PORT      ==      &LCD_DATA2_PORT      )      &&      (
&LCD_DATA2_PORT

== &LCD_DATA3_PORT )
     && (LCD_DATA0_PIN == 0) && (LCD_DATA1_PIN == 1) &&

(LCD_DATA2_PIN == 2) && (LCD_DATA3_PIN == 3) )
   {
      /* configure data pins as output */
      DDR(LCD_DATA0_PORT) |= 0x0F;


      /* output high nibble first */
      dataBits = LCD_DATA0_PORT & 0xF0;
      LCD_DATA0_PORT = dataBits |((data>>4)&0x0F);
      lcd_e_toggle();


      /* output low nibble */
```

```
        LCD_DATA0_PORT = dataBits | (data&0x0F);
        lcd_e_toggle();

        /* all data pins high (inactive) */
        LCD_DATA0_PORT = dataBits | 0x0F;
    }
    else
    {
        /* configure data pins as output */
        DDR(LCD_DATA0_PORT) |= _BV(LCD_DATA0_PIN);
        DDR(LCD_DATA1_PORT) |= _BV(LCD_DATA1_PIN);
        DDR(LCD_DATA2_PORT) |= _BV(LCD_DATA2_PIN);
        DDR(LCD_DATA3_PORT) |= _BV(LCD_DATA3_PIN);

        /* output high nibble first */
        LCD_DATA3_PORT &= ~_BV(LCD_DATA3_PIN);
        LCD_DATA2_PORT &= ~_BV(LCD_DATA2_PIN);
        LCD_DATA1_PORT &= ~_BV(LCD_DATA1_PIN);
        LCD_DATA0_PORT &= ~_BV(LCD_DATA0_PIN);
          if(data & 0x80) LCD_DATA3_PORT |= _BV

(LCD_DATA3_PIN);
          if(data & 0x40) LCD_DATA2_PORT |= _BV

(LCD_DATA2_PIN);
          if(data & 0x20) LCD_DATA1_PORT |= _BV

(LCD_DATA1_PIN);
          if(data & 0x10) LCD_DATA0_PORT |= _BV

(LCD_DATA0_PIN);
```

```
        lcd_e_toggle();

        /* output low nibble */
        LCD_DATA3_PORT &= ~_BV(LCD_DATA3_PIN);
        LCD_DATA2_PORT &= ~_BV(LCD_DATA2_PIN);
        LCD_DATA1_PORT &= ~_BV(LCD_DATA1_PIN);
        LCD_DATA0_PORT &= ~_BV(LCD_DATA0_PIN);
          if(data & 0x08) LCD_DATA3_PORT |= _BV
(LCD_DATA3_PIN);
          if(data & 0x04) LCD_DATA2_PORT |= _BV
(LCD_DATA2_PIN);
          if(data & 0x02) LCD_DATA1_PORT |= _BV
(LCD_DATA1_PIN);
          if(data & 0x01) LCD_DATA0_PORT |= _BV
(LCD_DATA0_PIN);
        lcd_e_toggle();

        /* all data pins high (inactive) */
        LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);
        LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);
        LCD_DATA2_PORT |= _BV(LCD_DATA2_PIN);
        LCD_DATA3_PORT |= _BV(LCD_DATA3_PIN);
    }
}
#else
#define lcd_write(d,rs) if (rs) *(volatile uint8_t*)
```

```
(LCD_IO_DATA) = d; else *(volatile uint8_t*)

(LCD_IO_FUNCTION) = d;
/* rs==0 -> write instruction to LCD_IO_FUNCTION */
/* rs==1 -> write data to LCD_IO_DATA */
#endif



/*****************************************************

*****************
Low-level function to read byte from LCD controller
Input:    rs     1: read data
                 0: read busy flag / address counter
Returns:  byte read from LCD controller
*****************************************************

****************/
#if LCD_IO_MODE
static uint8_t lcd_read(uint8_t rs)
{
   uint8_t data;



   if (rs)
      lcd_rs_high();                  /* RS=1:

read data      */
   else
      lcd_rs_low();                   /* RS=0:
```

```
        read busy flag */
           lcd_rw_high();                    /* RW=1

        read mode      */

           if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && (

        &LCD_DATA1_PORT      ==      &LCD_DATA2_PORT      )      &&      (
&LCD_DATA2_PORT

        == &LCD_DATA3_PORT )
            && ( LCD_DATA0_PIN == 0 )&& (LCD_DATA1_PIN == 1) &&

        (LCD_DATA2_PIN == 2) && (LCD_DATA3_PIN == 3) )
           {
              DDR(LCD_DATA0_PORT) &= 0xF0;        /* configure

        data pins as input */


              lcd_e_high();
              lcd_e_delay();
              data = PIN(LCD_DATA0_PORT) << 4;     /* read high

        nibble first */
              lcd_e_low();


              lcd_e_delay();                   /* Enable

        500ns low      */


              lcd_e_high();
              lcd_e_delay();
              data |= PIN(LCD_DATA0_PORT)&0x0F;    /* read low
```

```
nibble       */
        lcd_e_low();
    }
    else
    {
        /* configure data pins as input */
        DDR(LCD_DATA0_PORT) &= ~_BV(LCD_DATA0_PIN);
        DDR(LCD_DATA1_PORT) &= ~_BV(LCD_DATA1_PIN);
        DDR(LCD_DATA2_PORT) &= ~_BV(LCD_DATA2_PIN);
        DDR(LCD_DATA3_PORT) &= ~_BV(LCD_DATA3_PIN);

        /* read high nibble first */
        lcd_e_high();
        lcd_e_delay();
        data = 0;
        if ( PIN(LCD_DATA0_PORT) & _BV(LCD_DATA0_PIN) )
data |= 0x10;
        if ( PIN(LCD_DATA1_PORT) & _BV(LCD_DATA1_PIN) )
data |= 0x20;
        if ( PIN(LCD_DATA2_PORT) & _BV(LCD_DATA2_PIN) )
data |= 0x40;
        if ( PIN(LCD_DATA3_PORT) & _BV(LCD_DATA3_PIN) )
data |= 0x80;
        lcd_e_low();

        lcd_e_delay();                  /* Enable
```

```
500ns low       */

        /* read low nibble */
        lcd_e_high();
        lcd_e_delay();
        if ( PIN(LCD_DATA0_PORT) & _BV(LCD_DATA0_PIN) )

data |= 0x01;
        if ( PIN(LCD_DATA1_PORT) & _BV(LCD_DATA1_PIN) )

data |= 0x02;
        if ( PIN(LCD_DATA2_PORT) & _BV(LCD_DATA2_PIN) )

data |= 0x04;
        if ( PIN(LCD_DATA3_PORT) & _BV(LCD_DATA3_PIN) )

data |= 0x08;
        lcd_e_low();
    }
    return data;
}
#else
#define lcd_read(rs) (rs) ? *(volatile uint8_t*)

(LCD_IO_DATA+LCD_IO_READ) : *(volatile uint8_t*)

(LCD_IO_FUNCTION+LCD_IO_READ)
/* rs==0 -> read instruction from LCD_IO_FUNCTION */
/* rs==1 -> read data from LCD_IO_DATA */
#endif
```

```
/*****************************************************
*****************
loops while lcd is busy, returns address counter
*****************************************************
***************/
static uint8_t lcd_waitbusy(void)

{
   register uint8_t c;

   /* wait until busy flag is cleared */
   while ( (c=lcd_read(0)) & (1<<LCD_BUSY)) {}

   /* the address counter is updated 4us after the busy

flag is cleared */
   delay(2);

   /* now read the address counter */
   return (lcd_read(0));  // return address counter

}/* lcd_waitbusy */


/*****************************************************
*****************
Move cursor to the start of next line or to the first
```

```
line if the cursor
is already on the last line.
*********************************************************

***************/
static inline void lcd_newline(uint8_t pos)
{
    register uint8_t addressCounter;



#if LCD_LINES==1
    addressCounter = 0;
#endif
#if LCD_LINES==2
    if ( pos < (LCD_START_LINE2) )
        addressCounter = LCD_START_LINE2;
    else
        addressCounter = LCD_START_LINE1;
#endif
#if LCD_LINES==4
    if ( pos < LCD_START_LINE3 )
        addressCounter = LCD_START_LINE2;
    else if ( (pos >= LCD_START_LINE2) && (pos <

LCD_START_LINE4) )
        addressCounter = LCD_START_LINE3;
    else if ( (pos >= LCD_START_LINE3) && (pos <

LCD_START_LINE2) )
        addressCounter = LCD_START_LINE4;
```

```
    else
        addressCounter = LCD_START_LINE1;
#endif
    lcd_command((1<<LCD_DDRAM)+addressCounter);

}/* lcd_newline */


/*
** PUBLIC FUNCTIONS
*/

/************************************************************

*****************
Send LCD controller instruction command
Input:   instruction to send to LCD controller, see

HD44780 data sheet
Returns: none
************************************************************

***************/
void lcd_command(uint8_t cmd)
{
    lcd_waitbusy();
    lcd_write(cmd,0);
}


/************************************************************
```

```
*****************
Set cursor to specified position
Input:    x  horizontal position  (0: left most position)
          y  vertical position    (0: first line)
Returns:  none
*********************************************************

****************/
void lcd_gotoxy(uint8_t x, uint8_t y)
{
#if LCD_LINES==1
   lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
#endif
#if LCD_LINES==2
   if ( y==0 )
      lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
   else
      lcd_command((1<<LCD_DDRAM)+LCD_START_LINE2+x);
#endif
#if LCD_LINES==4
   if ( y==0 )
      lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
   else if ( y==1)
      lcd_command((1<<LCD_DDRAM)+LCD_START_LINE2+x);
   else if ( y==2)
      lcd_command((1<<LCD_DDRAM)+LCD_START_LINE3+x);
   else /* y==3 */
      lcd_command((1<<LCD_DDRAM)+LCD_START_LINE4+x);
#endif
```

```
}/* lcd_gotoxy */


/*************************************************************
*****************
*************************************************************
***************/
int lcd_getxy(void)
{
    return lcd_waitbusy();
}


/*************************************************************
*****************
Clear display and set cursor to home position
*************************************************************
***************/
void lcd_clrscr(void)
{
    lcd_command(1<<LCD_CLR);
}


/*************************************************************
```

```
*****************
Set cursor to home position
***********************************************************

***************/
void lcd_home(void)
{
    lcd_command(1<<LCD_HOME);
}



/*********************************************************

*****************
Display character at current cursor position
Input:    character to be displayed


Returns:  none
***********************************************************

***************/
void lcd_putc(char c)
{
    uint8_t pos;



    pos = lcd_waitbusy();   // read busy-flag and address

counter
    if (c=='\n')
```

```
    {
        lcd_newline(pos);
    }
    else
    {
#if LCD_WRAP_LINES==1
#if LCD_LINES==1
        if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH )
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE1,0);
#elif LCD_LINES==2
        if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH )
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE2,0);

        else if ( pos == LCD_START_LINE2+LCD_DISP_LENGTH
)
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE1,0);
#elif LCD_LINES==4
        if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH )
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE2,0);

        else if ( pos == LCD_START_LINE2+LCD_DISP_LENGTH
)
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE3,0);
        else if ( pos == LCD_START_LINE3+LCD_DISP_LENGTH
)
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE4,0);
```

```
        else if ( pos == LCD_START_LINE4+LCD_DISP_LENGTH

)
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE1,0);
#endif
        lcd_waitbusy();
#endif
        lcd_write(c, 1);
    }

}/* lcd_putc */



/************************************************************

*****************
Display string without auto linefeed
Input:    string to be displayed
Returns:  none
************************************************************

****************/
void lcd_puts(const char *s)
/* print string on lcd (no auto linefeed) */
{
    register char c;

    while ( (c = *s++) ) {
        lcd_putc(c);
    }
```

```
}/* lcd_puts */


/*****************************************************

*****************
Display string from program memory without auto linefeed
Input:     string from program memory be be displayed


Returns:   none
*****************************************************

***************/
void lcd_puts_p(const char *progmem_s)
/* print string from program memory on lcd (no auto

linefeed) */
{
    register char c;

    while ( (c = pgm_read_byte(progmem_s++)) ) {
        lcd_putc(c);
    }

}/* lcd_puts_p */


/*****************************************************

*****************
```

```
        Initialize display and select type of cursor
        Input:    dispAttr LCD_DISP_OFF           display off
                        LCD_DISP_ON            display on,

        cursor off
                        LCD_DISP_ON_CURSOR      display on,

        cursor on
                        LCD_DISP_CURSOR_BLINK   display on,

        cursor on flashing
        Returns:  none
        *********************************************************

        ***************/
        void lcd_init(uint8_t dispAttr)
        {
        #if LCD_IO_MODE
           /*
            *  Initialize LCD to 4 bit I/O mode
            */
             if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && (

        &LCD_DATA1_PORT      ==      &LCD_DATA2_PORT      )      &&      (
&LCD_DATA2_PORT

        == &LCD_DATA3_PORT )
             && ( &LCD_RS_PORT == &LCD_DATA0_PORT) && (

        &LCD_RW_PORT == &LCD_DATA0_PORT) && (&LCD_E_PORT ==

        &LCD_DATA0_PORT)
             && (LCD_DATA0_PIN == 0 ) && (LCD_DATA1_PIN == 1) &&
```

```
(LCD_DATA2_PIN == 2) && (LCD_DATA3_PIN == 3)
    && (LCD_RS_PIN == 4 ) && (LCD_RW_PIN == 5) &&

(LCD_E_PIN == 6 ) )
  {
    /* configure all port bits as output (all LCD

lines on same port) */
    DDR(LCD_DATA0_PORT) |= 0x7F;
  }
  else if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && (

&LCD_DATA1_PORT == &LCD_DATA2_PORT ) && (
&LCD_DATA2_PORT

== &LCD_DATA3_PORT )
      && (LCD_DATA0_PIN == 0 ) && (LCD_DATA1_PIN ==

1) && (LCD_DATA2_PIN == 2) && (LCD_DATA3_PIN == 3) )
  {
    /* configure all port bits as output (all LCD

data lines on same port, but control lines on different

ports) */
    DDR(LCD_DATA0_PORT) |= 0x0F;
    DDR(LCD_RS_PORT)  |= _BV(LCD_RS_PIN);
    DDR(LCD_RW_PORT)  |= _BV(LCD_RW_PIN);
    DDR(LCD_E_PORT)   |= _BV(LCD_E_PIN);
  }
  else
  {
    /* configure all port bits as output (LCD data
```

```
and control lines on different ports */
        DDR(LCD_RS_PORT)    |= _BV(LCD_RS_PIN);
        DDR(LCD_RW_PORT)    |= _BV(LCD_RW_PIN);
        DDR(LCD_E_PORT)     |= _BV(LCD_E_PIN);
        DDR(LCD_DATA0_PORT) |= _BV(LCD_DATA0_PIN);
        DDR(LCD_DATA1_PORT) |= _BV(LCD_DATA1_PIN);
        DDR(LCD_DATA2_PORT) |= _BV(LCD_DATA2_PIN);
        DDR(LCD_DATA3_PORT) |= _BV(LCD_DATA3_PIN);
    }
    delay(16000);        /* wait 16ms or more after

power-on       */

    /* initial write to lcd is 8bit */
    LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);  // _BV

(LCD_FUNCTION)>>4;
    LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);  // _BV

(LCD_FUNCTION_8BIT)>>4;
    lcd_e_toggle();
    delay(4992);         /* delay, busy flag can't be

checked here */

    /* repeat last command */
    lcd_e_toggle();
    delay(64);           /* delay, busy flag can't be

checked here */
```

```
    /* repeat last command a third time */
    lcd_e_toggle();
    delay(64);          /* delay, busy flag can't be
checked here */

    /* now configure for 4bit mode */
    LCD_DATA0_PORT &= ~_BV(LCD_DATA0_PIN);   //
LCD_FUNCTION_4BIT_1LINE>>4
    lcd_e_toggle();
    delay(64);          /* some displays need this
additional delay */

    /* from now the LCD only accepts 4 bit I/O, we can
use lcd_command() */
#else
    /*
     * Initialize LCD to 8 bit memory mapped mode
     */

    /* enable external SRAM (memory mapped lcd) and one
wait state */
    MCUCR = _BV(SRE) | _BV(SRW);

    /* reset LCD */
    delay(16000);                       /* wait 16ms
```

```
after power-on     */
   lcd_write(LCD_FUNCTION_8BIT_1LINE,0);   /* function

set: 8bit interface */
   delay(4992);                          /* wait 5ms

               */
   lcd_write(LCD_FUNCTION_8BIT_1LINE,0);   /* function

set: 8bit interface */
   delay(64);                           /* wait 64us

               */
   lcd_write(LCD_FUNCTION_8BIT_1LINE,0);   /* function

set: 8bit interface */
   delay(64);                           /* wait 64us
      */
#endif
   lcd_command(LCD_FUNCTION_DEFAULT);      /* function

set: display lines  */
   lcd_command(LCD_DISP_OFF);             /* display

off                */
   lcd_clrscr();                        /* display

clear              */
   lcd_command(LCD_MODE_DEFAULT);         /* set entry

mode               */
   lcd_command(dispAttr);                 /*

display/cursor control       */
```

```
        }/* lcd_init */
```

Epromgrava código central do dispositivo com todas as chamadas

```
inline int toupper(int ch) {
  if ( (unsigned int)(ch - 'a') < 26u )
    ch += 'A' - 'a';
  return ch;
}
int __isalpha_ascii ( int ch );
int __isalpha_ascii ( int ch ) {
    return (unsigned int)((ch | 0x20) - 'a') < 26u;
}
int isalpha ( int ch ) __attribute__((weak,alias("__isalpha_ascii")));

void playfair (int dir, char *key, char *pt, char *ct)
{
        char *alph = "ABCDEFGHIJKLMNOPQRSTUVXYZ";
        char keycopy[25], pt0, pt1;
        char *ptcopy = ct;
        int i, j, k, lk, lp, rpt0, rpt1, cpt0, cpt1;
        int alphabet [25], row[25], col[25];

        for (i = 0; i < 25; i++) alphabet[i] = -1;

        lk = strlen (key);
        j = 0;
        for (i = 0; i < lk; i++)
        {
                k = toupper (key[i]);
                if (!isalpha (k)) continue;
                if (k == 'J') k = 'I';
                k = strchr (alph, k) - alph;
                if (alphabet[k]         != -1) continue;
                alphabet[k] = j;
                keycopy[j++] = alph[k];
        }
        for (i = 0; i < 25; i++)
        {
                if (alphabet[i] == -1)
                {
                        alphabet[i] = j;
```

```
                keycopy[j++] = alph[i];
        }
}
for (i = 0; i < 25; i++)
{
        row[i] = alphabet[i] / 5;
        col[i] = alphabet[i] % 5;
}

j = 0;
lp = strlen (pt);

for (i = 0; i < lp; i++)
        if (isalpha (pt[i]))
                ptcopy [j++] = toupper (pt[i]);
lp = j;

if (lp & 1)

ptcopy[lp] = '\0';

for (i = 0; i < lp; i += 2)
{
        pt0 = strchr (alph, ptcopy[i]) - alph;
        pt1 = strchr (alph, ptcopy[i+1]) - alph;
        cpt0 = col[pt0];
        cpt1 = col[pt1];
        rpt0 = row[pt0];
        rpt1 = row[pt1];

        if (pt0 == pt1)
        {
                if (dir > 0)
                {
                        ct[i] = ct[i+1] = keycopy[(cpt0 + 5 * rpt0 + 1) % 25];
                }
                else
                {
                        ct[i] = ct[i+1] = keycopy[(cpt0 + 5 * rpt0 + 24) % 25];
                }
        }
        else
        if (rpt0 == rpt1)
        {
                if (dir > 0)
                {
                        ct[i] = keycopy[(cpt0+1) % 5 + 5 * rpt0];
                        ct[i+1] = keycopy[(cpt1+1) % 5 + 5 * rpt1];
                }
                else
```

```
                {
                        ct[i] = keycopy[(cpt0+4) % 5 + 5 * rpt0];
                        ct[i+1] = keycopy[(cpt1+4) % 5 + 5 * rpt1];
                }
        }
        else
        if (cpt0 == cpt1)
        {
                if (dir > 0)
                {
                        ct[i] = keycopy[cpt0 + 5 * ((rpt0 + 1) % 5)];
                        ct[i+1] = keycopy[cpt1 + 5 * ((rpt1 + 1) % 5)];
                }
                else
                {
                        ct[i] = keycopy[cpt0 + 5 * ((rpt0 + 4) % 5)];
                        ct[i+1] = keycopy[cpt1 + 5 * ((rpt1 + 4) % 5)];
                }
        }
        else
        {
                ct[i] = keycopy[cpt1 + 5 * rpt0];
                ct[i+1] = keycopy [cpt0 + 5 * rpt1];
        }
    }
}


    /*****************************************

    * Chip type           : ATmega8

    * Clock frequency     : 2457600Hz

    *****************************************/

    #include <avr/io.h>

    #include <avr/interrupt.h>

    #include <inttypes.h>

    #include <util/delay.h>

    #include <string.h>

    #include <stdio.h>

    #include "lcd.h"

    #include "playfair.c"

    #include "e2prom.h"
```

```
#define F_OSC 4000000
#include <ctype.h>
#define CHAVE "gato"

void playfair (int dir, char *key, char *pt, char *ct);
void gravaMensagem(char mensagem[]);
void leMensagem(char mensagem[]);
void codifica(char *pt, char *ct);
void decodifica(char *pt, char *ct);

int main(void) {
	char mensagemCorreta[] = "Chave Correta";
	char buffer[100];
	char buffer2[100];
	int abriu = 0;

	sei();
	DDRB |= _BV(DDB1)|_BV(DDB2);
	eeprom_init();
	lcd_init(LCD_DISP_ON);
	gravaMensagem(mensagemCorreta);
	while (1) {
			if (abriu == 0){
			lcd_clrscr();
			lcd_home();
			lcd_puts("Favor inserir chave.");
			_delay_ms(500);
		}
		leMensagem(buffer2);
```

```
if (!strcmp(mensagemCorreta, buffer2)){
        if(abriu){
                PORTB=0;
                continue;
        }
        lcd_clrscr();
        lcd_home();
        lcd_puts("Chave aceita.");
        PORTB = _BV(PB1);
        _delay_ms(500);
        PORTB |= _BV(PB2);
        lcd_puts(" Abriu!");
        abriu = 1;
        _delay_ms(1000);
        PORTB = _BV(PB1);
} else {
        abriu = 0;
}
/*
eeprom_read_byte(0, &buffer2);
if(buffer2==0x23){
        _delay_ms(500);
        eeprom_read_byte(10, &buffer2);
        if(buffer2==0x44){
                if(abriu == 0) {
                        lcd_clrscr();
                        lcd_home();
                        lcd_puts("\nChave aceita.");
                        PORTB = _BV(PB1);
```

```
                                        _delay_ms(500);
                                        PORTB |= _BV(PB2);
                                        lcd_puts(" Abriu!");
                                        abriu = 1;
                                        _delay_ms(1000);
                                        PORTB = _BV(PB1);

                                }
                        } else {
                                abriu = 0;
                        }
                } else {
                        abriu = 0;
                }*/
        }
        return 0;
}
void codifica(char *pt, char *ct){
        playfair (1, CHAVE, pt, ct);
}
void decodifica(char *pt, char *ct){
        playfair (0, CHAVE, pt, ct);
}
void gravaMensagem(char mensagem[]){
        uint16_t n;
        lcd_clrscr();
        lcd_home();
        lcd_puts("Iniciando gravacao:\n");
              for(n=0; n<strlen(mensagem); n++){
                lcd_putc(mensagem[n]);
```

```
                eeprom_write_byte(n, mensagem[n]);
                _delay_ms(250);
        }
        eeprom_write_byte(strlen(mensagem), '\0');
        lcd_puts(" OK!");
        _delay_ms(1000);
}
void leMensagem(char mensagem[]){
        uint16_t n;
        char lido = 1;
        lcd_clrscr();
        lcd_home();
        lcd_puts("Mensagem Lida:\n");
        _delay_ms(500);
        for(n=0; n<50 ;n++){
                eeprom_read_byte(n, &lido);
                if (lido == '\0'){
                        mensagem[n] = lido;
                        break;
                }
                mensagem[n] = lido;
                //lcd_putc(lido);
                _delay_ms(250);
        }
}
```