Desenvolvimento com microcontroladores Atmel AVR

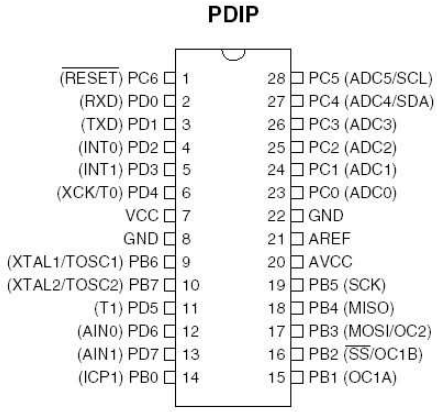
Geovany A. Borges, Antônio Padilha Lanari Bo, Alexandre Simões Martins,
Leandro César Cotta, Maurílio Fernandes, Gabriel Freitas,
Ener Diniz Beckmann, Augusto César Coelho Felix

Laboratório de Controle e Visão por Computador (LCVC)
Departamento de Engenharia Elétrica - ENE
Faculdade de Tecnologia - FT
Universidade de Brasília - UnB
Brasília - DF - Brasil

14 de junho de 2008

Resumo

Esta nota técnica pretende auxiliar aqueles que dão seus primeiros passos no desenvolvimento de projetos com microcontroladores. Mais especificamente, ela deve servir como um guia introdutório ao microcontrolador AVR ATmega8, da Atmel, descrevendo os procedimentos básicos para programação e algumas de suas funções básicas.



Revisões:

01/06/2008	Atualização do texto para a mais recente distribuição do WinAVR e correção no código exemplo para conversor A/D considerando a conexão usada no circuito de teste.
11/06/2006	Pequenas alterações (redução do tamanho da fonte usada para listagens de programas) e inclusão de uma figura que ilustra melhor a soldagem da placa da gravadora BSD com o conector DB-25.
04/06/2006	Revisão geral do texto, inclusão de material sobre gravadora BSD, LCD e terminal.
17/01/2005	Primeira versão da nota técnica sobre AVR

Sumário

1 Introdução

Em sistemas de Controle e Automação, são raros os exemplos em que não há necessidade do uso de alguma unidade de processamento. De fato, a não ser que o sistema possua uma lógica extremamente simples, se faz necessário o uso de uma ou mais CPUs (*Central Processing Unit*). Isso inclui sistemas que apresentam desde uma simples interface com o usuário por meio de um display LCD (*Liquid Cristal Display*) até o complexo controle de um veículo aeroespacial.

Entretanto, definida a necessidade de se empregar um elemento processador no sistema, resta o problema de definir qual plataforma utilizar. Entre as diversas disponíveis, há CLPs (Controladores Lógico-Programáveis), FPGAs (*Field-Programmable Gate Array*), DSPs (*Digital Signal Processing*), PCs (embarcados ou não), entre outras. Assim, a escolha do *"cérebro"* do sistema deve se dar a partir de um compromisso entre as diversas necessidades do projeto e as características apresentadas pelo elemento processador, como custo, capacidade de processamento, memória, linguagem de programação disponível, capacidade de atuar em sistemas de controle em tempo real, consumo de energia, etc. Nesse contexto, os microcontroladores geralmente se apresentam como uma das alternativas de menor custo, confiabilidade satisfatória, simplicidade, menor tempo de desenvolvimento e menor consumo; porém com limitada capacidade de processamento e memória.

Em termos gerais, os microcontroladores podem ser definidos como processadores que foram encapsulados com memória, interface de entrada/saída de dados e dispositivos periféricos. Entre os periféricos estão conversores A/D (analógico/digital), temporizadores/contadores, interface para comunicação serial, *watchdog* programável, etc. Em outras palavras, são computadores encapsulados em um único invólucro. Tornaram-se comuns em diversos ramos da indústria a partir do final da década de 70 e atualmente existe um número crescente de opções disponíveis no mercado.

Esta nota técnica pretende apresentar uma introdução ao desenvolvimento de projetos empregando a família de microcontroladores AVR da Atmel, em especial o modelo ATmega8, muito utilizado em pequenos projetos do Grupo de Robótica, Automação e Visão Computacional (GRAV) da Universidade de Brasília. Uma ênfase será dada à utilização da linguagem C na programação do microcontrolador.

Para os que têm alguma experiência com o AVR, ou mesmo para aqueles mais apressados, o Anexo B apresenta os procedimentos para uma iniciação rápida.

2 O microcontrolador Atmel AVR ATmega8

Nesta seção, se pretende apresentar brevemente as características principais do funcionamento deste microcontrolador. Grande parte das informações aqui fornecidas foram obtidas do manual do ATmega8 [Atmel 2004], doravante denominado AVR, ATmega8 ou simplesmente microcontrolador. Este documento deve ser, de fato, o ponto de partida na busca de informações que não se encontram nesta nota técnica. Porém, as informações estão aqui transcritas com uma maior clareza na exposição, bem como acompanhadas por algumas alterações visando facilitar a iniciação ao uso do dispositivo.

O ATmega8 é um microcontrolador 8-bit de tecnologia CMOS e arquitetura RISC. Uma característica interessante deste dispositivo é sua capacidade de executar uma instrução por ciclo de relógio. Esta taxa de execução de instruções foi possível de ser alcançada em virtude da conexão direta de seus 32 registradores de propósito geral com a unidade lógica aritmética. Esta caracterísa o deixa em vantagem quando comparado com os concorrentes da família PIC da *Microchip Technology*. Além do mais, apesar de ser RISC, possui um grande número de instruções (130), o que permite melhor 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. Por outro lado, sempre que se usar linguagem C para sistemas embarcados, deve-se ter em mente que os recursos de memória são bastante limitados, de forma que boas práticas para sistemas embarcados devem ser usadas.

Com o objetivo de maximizar o desempenho e o paralelismo, o AVR segue arquitetura Harvard, em que os barramentos associados às memórias de dados e do programa são distintos. Além disso,

utiliza-se a técnica do *pipeline*, em que, enquanto uma instrução começa a ser executada, uma outra já é buscada na memória de programa para que a mesma possa ser executada no próximo ciclo de relógio. A Figura 1 ilustra o diagrama de blocos do microcontrolador [Atmel 2004].

3 Ferramentas de desenvolvimento

3.1 Hardware

Para desenvolvimento com o ATmega8, o hardware mínimo sugerido é dado por um circuito de referência e pela gravadora BSD, mostrados na Figura 2. O circuito de referência contém o mínimo para o o ATmega8 funcionar apropriadamente, podendo ser conectado diretamente a uma protoboard para desenvolvimento. A gravadora possui um conector para o circuito de referência e um outro para porta paralela de microcomputadores PC. As seções seguintes apresentam seus circuitos e descrevem os procedimentos de montagem.

3.1.1 Gravadora BSD

A gravadora é um circuito faz a interface entre o microcomputador e o microcontrolador, realizando as tarefas de programação e configuração. Elas são projetadas de acordo com as particularidades técnicas de cada microcontrolador e utilizam geralmente as portas serial ou paralela do PC. Para o caso do AVR, existem alguns modelos comerciais, mas também existem muitos outros gratuitos, que podem ser montados sem muita dificuldade. No caso do LCVC, costuma-se utilizar a gravadora BSD. Seu diagrama esquemático está disponível na Figura 3. A gravadora é conectada ao microcomputador pela porta paralela, e ao microcontrolador por meio de um *flat cable* com conector de 10 pinos na extremidade. Sua alimentação é proveniente do circuito em que a gravadora será conectada. Os pinos 9 e 10 do conector da gravadora não devem ser usados. desta forma, se o usuário cometer o erro de inverter o conector, isto não deve causar a queima de componentes.

Para montar a gravadora, são necessários os seguintes componentes (*c.f.* Figura 4):

- 1 placa de circuito impresso confeccionada para a gravadora BSD;
- 1 circuito integrado 74LS367;
- 1 resistor de 4k7 a 6k8, 1/8W;
- 1 capacitor eletrolítico 10uF 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*, normalmente são fios de instalação telefônica ou de cabo de rede;

Para a BSD, sugere-se o *layout* da Figura 5(a). Essa placa é soldada diretamente a um conector DB-25 macho, utilizado para conectar à porta paralela de um PC. Conforme as setas em azul na Figura 5(b), os quatro furos mais à esquerda são para conexões por meio de fios com os pinos 7, 8 , 9 e 10 do DB-25. Já nos oito furos à direita, são soldados cada fio do flat cable, tomando-se o cuidado que *o terceiro fio é trocado com o quinto*. Ainda na Figura 5(b), a linha vermelha vertical é um fio que une os dois furos nos extremos da placa e também fica na parte superior da mesma. O 74LS367 é soldado direto na placa, sem soquete, sendo que o pino 1 é o que está mais próximo do resitor de 6k8.

Para a soldagem do conector DB-25, deve-se colocar a placa entre os pinos inferiores e superiores do conector, conforme mostras as Figuras 5(c)-(e). A face do cobre está voltada para os pinos inferiores

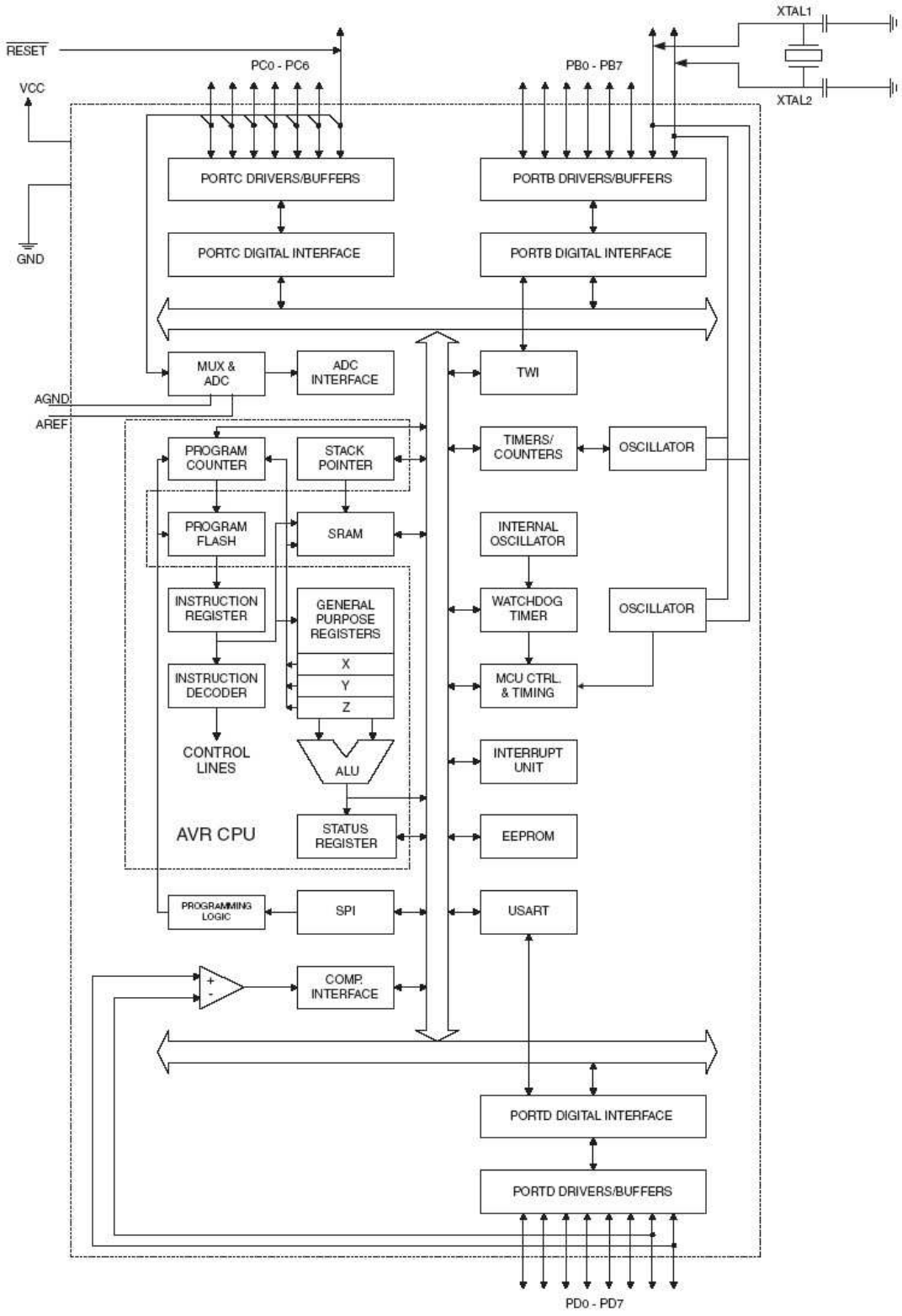
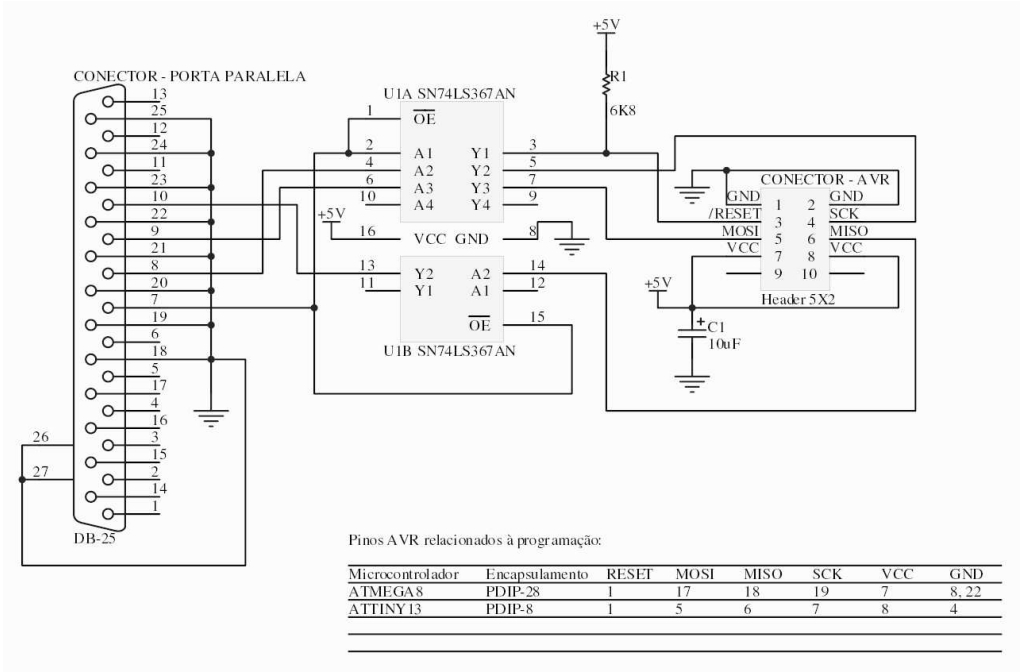


Figura 1: Diagrama de blocos do microcontrolador [Atmel 2004].

Figura 2: Foto do circuito de referência para protoboard e gravadora BSD.



Microcontrolador	Encapsulamento	RESET	MOSI	MISO	SCK	VCC	GND
ATMEGA8	PDIP-28	1	17	18	19	7	8, 22
ATTINY13	PDIP-8	1	5	6	7	8	4

Figura 3: Diagrama esquemático da gravadora BSD.

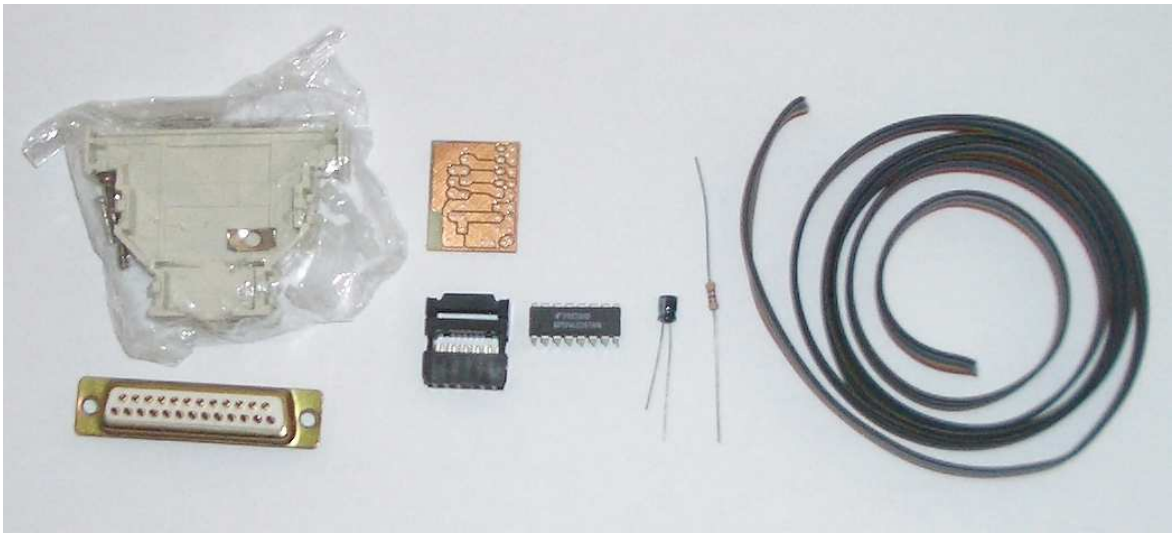


Figura 4: Material para montagem da gravadora BSD

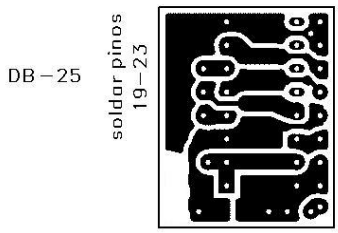
(14 a 25). Certifique-se de que a placa está bem centralizada no mesmo e solde os pinos de 18 a 25 uns aos outros, conforme a Figura 5(c). Na placa, a face do cobre deve alcançar do pino 19 ao 23, que devem ser soldados à placa. Por último, para colocação do conector no *flat cable*, deve-se tomar cuidado para que o fio 1 do cabo corresponda ao pino 1 do conector, indicado por uma pequena seta em relevo, conforme mostra a Figura 5(f).

3.1.2 Circuito de referência

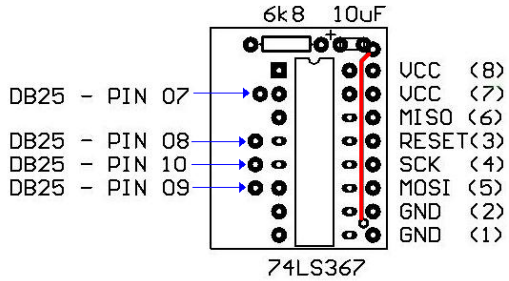
O circuito de referência é um circuito que serve de base para o desenvolvimento com o ATmega8. Nele estão os capacitores de desacoplamento para alimentação e conversor A/D, assim como um conector *header* para a gravadora BSD. Neste documento propõe-se o esquemático da Figura 6. Neste circuito, o cristal é opcional, devendo este ser usado quando a fonte de relógio da CPU for cristal externo.

De forma similar à gravadora BSD, sugere-se uma placa de circuito impressso para o circuito de referência. Esta placa foi concebida de modo a permitir que o circuito de referência seja inserido em protoboard, facilitando assim o desenvolvimento com o ATmega8. Para a montagem desse módulo, são necessários os seguintes componentes (*c.f.* Figura 7(a)):

- 1 placa de circuito impresso confeccionada para o circuito de referência;
- 2 soquetes DIP14 torneados. Em conjunto, os dois soquetes formam um soquete de 28 pinos estreito, difícil de ser encontrado no Brasil;
- 1 cristal de quatzo de no máximo 16MHz (opcional, se utilizar cristal externo como fonte de relógio);
- 1 barra de soquete torneado com 3 pinos (suporte do cristal);
- 2 capacitores de 100 nF, cerâmico ou poliéster;
- 2 capacitores cerâmicos de 22 pF;
- 2 barras de pinos simples header, 14 pinos cada. Cada barra deve ter seus pinos rebaixados, ficando rentes ao plástico de suporte, usando um alicate de bico como mostrado na Figura 7(b);
- 1 barra de pinos dupla, 2 x 5 pinos, para receber o conector da gravadora BSD;
- fios para *jumpers*, normalmente são fios de instalação telefônica.

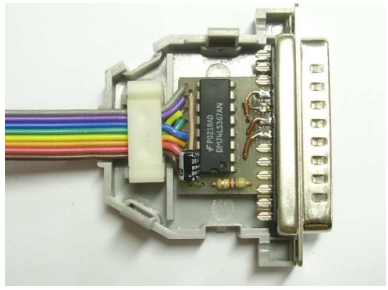


(a) *Layout* inferior

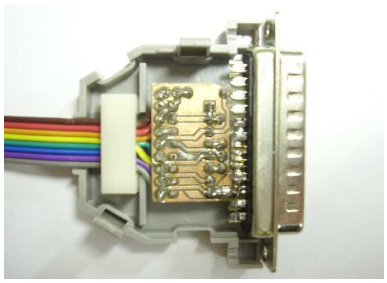


(b) Face superior (fazer atenção com a seqüência dos fios do *flat cable*)

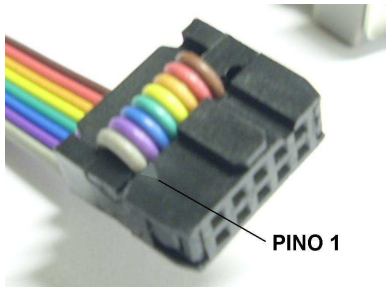
(c) Inserção da placa entre os pinos do conector DB-25



(d) Montagem superior



(e) Montagem inferior



(f) Conector

Figura 5: Placa da gravadora BSD e sua montagem no conector DB-25.

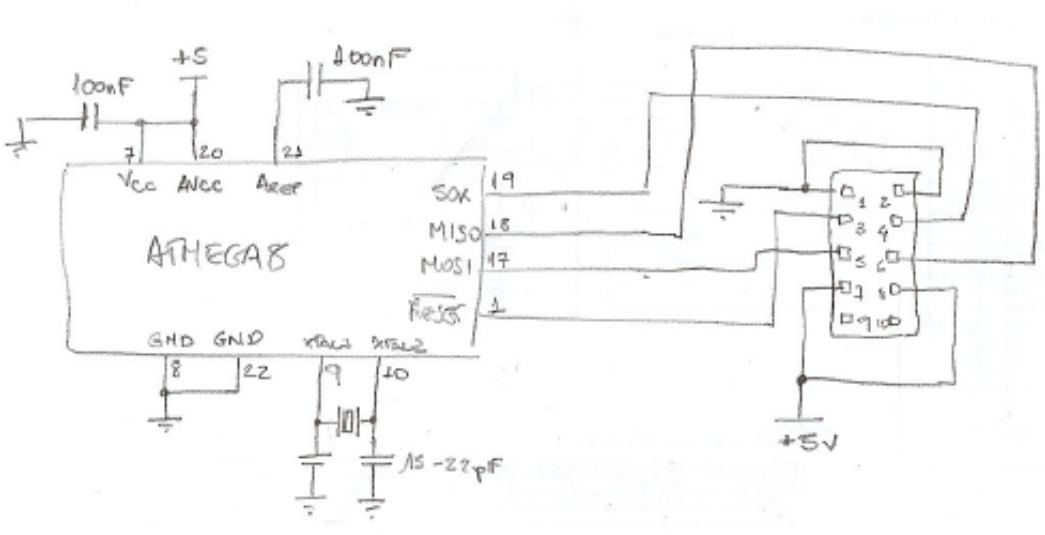
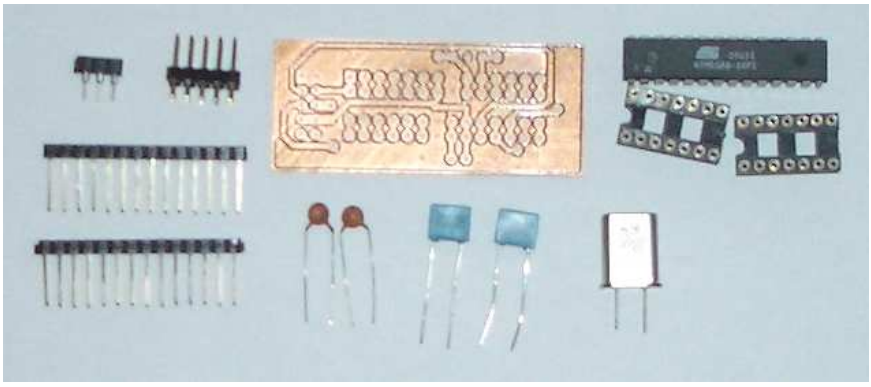
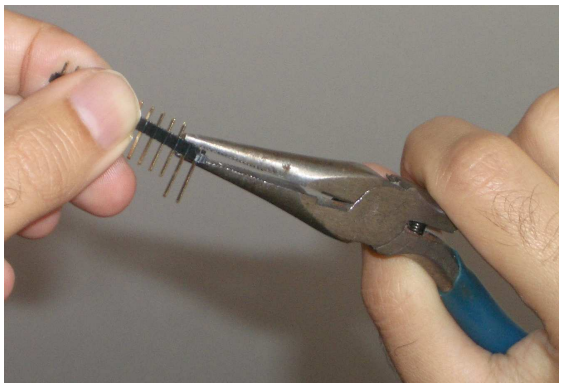


Figura 6: Circuito de referência para o ATmega8



(a) Componentes



(b) Alteração das barras de terminais de 14 pinos

Figura 7: Material para montagem do circuito de referência ATmega8

O *layout* da placa de referência é mostrado na Figura 8(a). A disposição dos componentes é mostrada na Figura 8(b). Deve ser observado que tem uma barra de pinos logo abaixo do soquete do microcontrolador, devendo esta barra ser soldada antes dos soquetes DIP14. Para a montagem, a dica é soldar primeiro os componentes mais baixos e do centro, deixando por último os componentes próximo às bordas. Assim, deve-se começar soldando os fios de ponte, que são as linhas vermelhas na Figura 8(b). Depois, devem ser soldadas as barras de pinos simples. , porém, antes de soldá-las empurre todos os pinos de modo que fiquem rentes ao plástico. Finalmente solde os soquetes, os capacitores e a barra de pinos dupla.

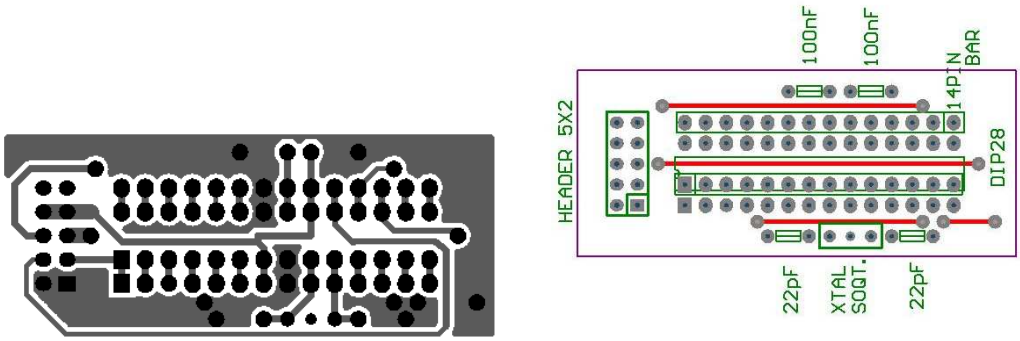
As Figuras 8(c)-(d) mostram a placa finalizada. É interessante usar uma caneta de retroprojetor para marcar o pino 1 do *header* para conexão com a gravadora. Nesse caso, o pino 1 é o que está mais próximo do pino 1 do ATmega8.

Conforme mencionado antes, a placa do circuito de referência foi concebida para ser inserida em um protoboard. Para poder usar a placa, ainda faz-se necessário conectar a alimentação. Com a placa no protoboard, deve-se conectar VCC (5V) ao pino 7 e GND (0V) ao pino 8 do ATmega8. Estes pinos são conectados à protoboard por meio das barras de terminais de 14 pinos.

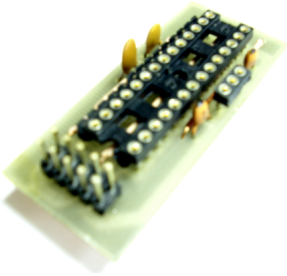
3.1.3 Teste do hardware

Para testar a gravadora e o circuito de referência, sugere-se seguir os seguintes passos:

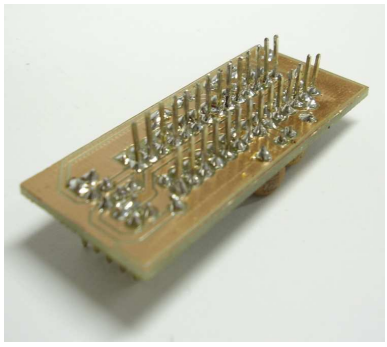
- Coloque a placa de referência a uma protoboard, e conecte a alimentação. Não ligar ainda a fonte de alimentação;
- Conecte a gravadora BSD na placa de referência;



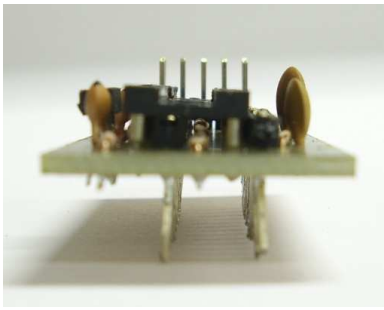
(a) *Layout* inferior (b) Face superior



(c) Vista superior



(d) Vista inferior



(e) Vista lateral

Figura 8: Montagem do circuito de referência ATmega8

- Certifique-se que a porta paralela está operando no modo padrão (SPP, ou Standard Parallel Port). Para verificar isto, deve-se reiniciar o microcomputador e entrar no programa de configuração da BIOS. Em geral esta informação encontra-se em um menu de periférios. Se a porta paralela estiver configurada para qualquer outro modo, deve-se alterar a configuração para SPP, sair do programa monitor da BIOS salvando a nova configuração, e reiniciar o microcomputador;
- Conecte a gravadora BSD na porta paralela do microcomputador;
- Ligue a fonte de alimentação do circuito de referência. A gravadora tem sua alimentação proveniente do circuito de referência.
- Certifique-se que o programa Avrdude está configurado corretamente (ver Seção 3.2.3);
- Execute `avrdude.exe` que está no diretório `WinAVR/bin` com as seguintes opções:

  ```
  avrdude.exe -p atmega8 -c bsd -P lpt1 -t -u
  ```

  Se aparecer logo no início a mensagem avrdude: `AVR device not responding` ou algo similar, então algo está errado. Reveja todos os passos de montagem da gravadora e do circuito de referência.

3.1.4 Observações muito importantes

Nunca se deve desconectar a gravadora do circuito (ou do PC) com o Atmega8 ligado. Este procedimento pode gerar gravação do registro LFUSE para o valor 00h (hexadecimal), que leva o microcontrolador a operar somente com fonte de relógio externa. Isto impossibilita a gravação do ATmega8 com qualquer outra fonte de relógio, inclusive a que vinha sendo usada. Para recuperar o microcontrolador, deve-se 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).

3.2 Software

Tendo disponíveis a gravadora e o circuito do microcontrolador, deve-se providenciar o ambiente computacional que se comunicará com o dispositivo de acordo com o protocolo estabelecido pelo fabricante. De fato, necessita-se ao menos de um compilador e/ou um montador para o microcontrolador, e um programa responsável pela gravação do dispositivo por meio do hadware da gravadora.

No caso do LCVC, utiliza-se largamente o WinAVR, um pacote de programas em código aberto para desenvolvimento com microcontroladores da família AVR da Atmel no ambiente Windows, que inclui um editor de texto, *debugger*, bibliotecas em C, entre outros. Entre os programas mais importantes que acompanham o pacote, são descritos a seguir:

- Programmer's Notepad: Ambiente Integrado de Desenvolvimento, que é gratuito, e permite ser configurado para trabalhar com as ferramentas de desenvolvimento AVR;
- Avrdude: programa de gerenciamento da gravadora em linha de comando DOS, que permite gravar o dispositivo, ler seu conteúdo e alterar os registros de configuração da família AVR;
- avr-toochain: Na verdade, isto não é um programa mas um conjunto de ferramentas necessárias para compilar um programa para o AVR utilizando ferramentas GNU (software livre).

3.2.1 WinAVR

Como mencionado anteriormente, o WinAVR é um pacote de vários softwares utilitários para desenvolvimento com a família AVR. O site da distribuição é `http://winavr.sourceforge.net/`. Sua instalação é trivial, bastando seguir os passos indicados pelo programa de instalação. Entretanto, no caso de Windows 2000/XP, faz-se necessário *ter privilégios de administrador do sistema.* Após instalação, é criado o diretório `WinAVR` na raiz de algum disco do microcomputador, assim como `\WinAVR\bin;\WinAVR\utils\bin;` é inserido na variável de ambiente `PATH`. Se os programas que compõem o WinAVR não funcionarem após instalação, deve-ser verificar se a variável `PATH` foi realmente alterada. Caso não, sua alteração deve ser feita manualmente.

Ap[os a instalação do WinAVR, recomenda-se uma leitura do seu manual, acessível no menu de programas.

3.2.2 Programmer's Notepad

Programmer's Notepad 2 (PN) é uma ambiente de desenvolvimento integrado (IDE, do inglês) que acompanha o pacote. Através do PN é possível criar um projeto, que é composto de um ou mais arquivos, editar os arquivos e compilar o projeto. A Figura 9 mostra o PC aberto com um projeto, em que se vê os arquivos que compõem o projeto (`main.c` e `makefile`), o arquivo `main.c` aberto para edição com diferentes cores para as estruturas do programa em linguagem C, e uma janela de saída, que contém mensagens de compilação.

Quando instalado, o PN deve ser o programa default para abrir arquivos com a extensão `.pnproj`, que é a extensão do arquivo principal de um projeto. Os passos para criar um novo projeto são os seguintes:

- Execute pn.exe, que está no diretório `\WinAVR\pn` ou é acessível através da barra de ferramentas WinAVR do Windows;
- No PN, crie um novo projeto através do menu File -> New -> Project. Uma janela deve abrir para o usuário entrar com o nome do projeto, que receberá a extensão `.pnproj`, assim como sual locação;
- No espaço do projeto (lado esquerdo da interface) pode-se incluir arquivos no projeto clicando no botão direito do mouse que deve estar posicionado sobre o nome do projeto. Suger-se que sejam incluídos todos os arquivos `.c` e `.h` que compõem o projeto, e o arquivo `makefile`. Ao

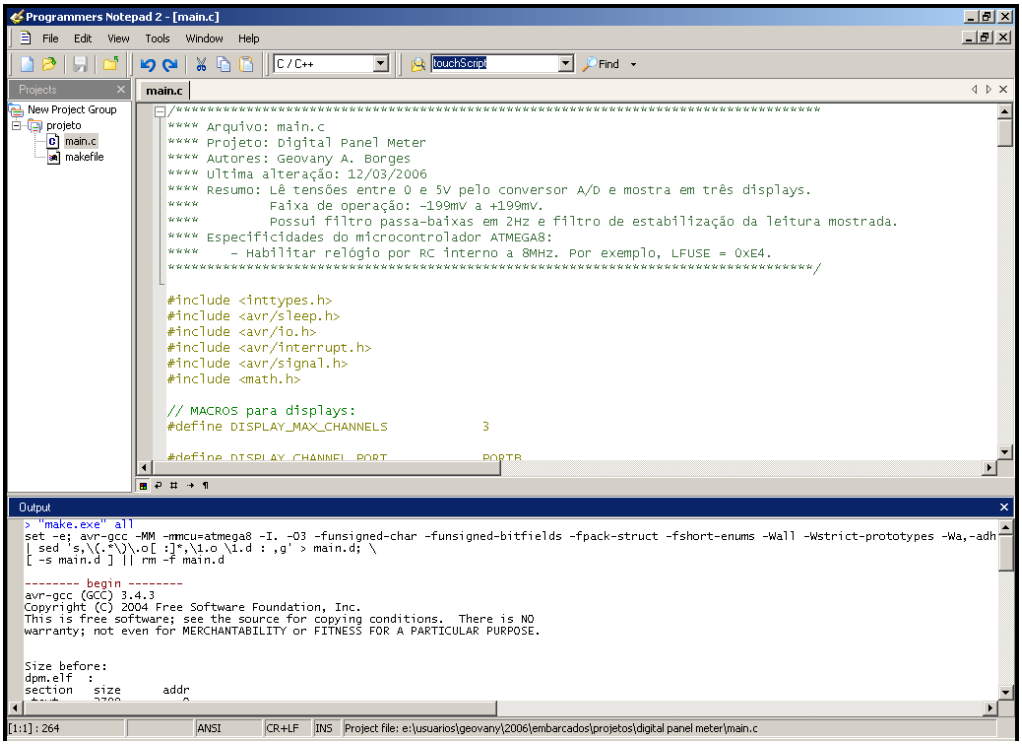


Figura 9: Programmer's Notepad 2 com um projeto aberto.

estarem presentes no espaço do projeto, estes arquivos podem ser facilmente abertos clicando com o botão direito do mouse sobre seus nomes;

- Com um arquivo em linguagem C aberto para edição, o menu Tools exibe as opções [WinAVR] Make All, [WinAVR] Make Clean e [WinAVR] Make Program. Ao clicar em qualquer uma destas opções, o programa make.exe é chamado com as opções "`make all`", "`make clean`" e "`make program`", respectivamente. A primeira destas opções inicia os procedimentos de compilação guiada pelo arquivo `makefile`. Um exemplo de arquivo `makefile` é apresentado e discutido no Anexo D. Sua compreensão é essencial para iniciar o desenvolvimento de novos projetos. O resultado da compilação é um arquivo `.hex` que é a imagem que deve ser gravada no microcontrolador pelo utilitário avrdude.

3.2.3 Avrdude

A gravadora BSD nada mais é do que uma peça de hardware que precisa ser comandada por um programa. Um destes programas é o Avrdude, que é um programa que permite gerenciar várias gravadoras, entre elas a BSD. O Avrdude é disponibilizado no diretório `WinAVR/bin` como `avrdude.exe`, que opera em linha de comando. Além de poder lere escrever programas na memória *flash* do microcontrolador, o Avrdude também permite alterar registros de configuração. A Seção 4.2 descreve com mais detalhe alguns procedimentos básicos de uso do `avrdude.exe`. Ao leitor recomenda-se consultar o manual do `avrdude.exe` que acompanha o CD de iniciação ao AVR (ver Anexo A).

Para qualquer versão do Avrdude, se estiver usando qualquer versão do Windows diferente do Windows 98, o *driver* GiveIO deverá está instalado. Instale GuiveIO executando `install_giveio.bat` do diretório `WinAVR/bin`.

3.2.4 avr-toochain

O projeto GNU (GNU is not GNU)[1] foi iniciado em 1984 por Richard Stallman com o objetivo de criar um sistema operacional totalmente livre, além de ferramentas para desenvolvimento. O projeto iniciou com as ferramentas, e abraçou o sistema operacional Linux em 1991 como alvo dos desenvolvimentos. Entretanto, muitas das ferramentas também funcionam no Windows, permitindo assim uma maior difusão do software livre. Desta forma, é possível desenvolver programas para vários microprocessadores e microcontroladores sem a necessidade de comprar ferramentas comerciais. Para o caso do AVR, tem-se, entre outras ferramentas, o avr-gcc (compilador C), avr-g++ (compilador C++), avr-as (assembler) e avr-ld (linker). No WinAVR, estas ferramentas estão instaladas no diretório `\WinAVR\bin`.

As ferramentas GNU não são descritas em detalhe nesta versão do documento. Uma descrição detalhada é feita no site `http://www.gnu.org/manual/manual.html`, em que tem-se apenas o nome da ferramenta sem o prefixo avr-.

Com a instalação do WinAVR, na barra de ferramentas é criada uma pasta WinAVR com alguns links, entre eles o *avr-libc manual*. Este link aponta para o arquivo `\WinAVR\doc\avr-libc\avr-libc-user-manual\index.html` que é o manual da biblioteca LibC para a família AVR. LibC é a na verdade um conjunto de bibliotecas básicas para se trabalhar com linguagem C/C++ para a família AVR. O manual da biblioteca LibC é sempre bastante consultado para desenvolvimento.

4 Procedimentos básicos de desenvolvimento

4.1 O primeiro projeto

Cabe dizer, neste momento, que esta nota técnica trata apenas do desenvolvimento de projetos em linguagem C. Para aqueles que gostariam de trabalhar com linguagem *assembly*, a principal diferença está que os arquivos fonte possuem extensão `.s`, mudando muito pouco a nível do processo de construção da imagem hexadecimal a ser programada no microcontrolador. Porém, cabe colocar que a escolha pela programação em *assembly* não gera tantos benefícios em termos de desempenho quanto se obtém em outros microcontroladores, principalmente devido (i) ao grande número de instruções RISC da família AVR e (ii) porque o compilador da linguagem C (*i.e.*, gcc) é bastante eficiente e foi amplamente testado. Entretanto, existe espaço para que trechos de código sejam escritos em assembly, principalmente se for desejado fazer manipulações diretamente em registradores da CPU ou realizar operações de muito baixo nível, não permitidas pela linguagem C.

Considerando o circuito da Figura 12 no Anexo E, sugere-se um simples programa para acender intermitentemente um LED ligado a uma porta de E/S do microcontrolador. O algoritmo é extremamente simples e não requer maiores explicações além daquelas disponíveis no código, listado logo abaixo. De fato, o programa simplesmente liga e desliga o LED a um período que depende da constante *TROCA*. As montagem do circuito necessário para a visualização do resultado é composto unicamente pela ligação do LED e de um resistor de 220Ω entre o pino 5 (PD3) e +5V, conforme mostrado no circuito 12.

```
#include <inttypes.h>
#include <avr/io.h>
#define TROCA 10000

void main (void) {
   int cont = 0;

   DDRD |= _BV(DDD3); /* Neste comando seta-se o pino correspondente
                         como pino de saída. Poderia-se
                         obter o mesmo efeito com o comando
```

[1] Para entender melhor o porquê desta sigla, sugere-se uma consulta ao site `http://pt.wikipedia.org/wiki/GNU`

```
                        DDRD |= 0x04; */
   for(;;){
      cont++;
      if (cont == TROCA){
         cont = 0;
         PORTD = ~PORTD;
      }
   }
}
```

Se denominarmos o programa acima de `teste.c`, ele deve fazer parte de um projeto do Programmer's Notepad, conforme explicado na Seção 3.2.2. Um exemplo de makefile é mostrado no Anexo D. Entretanto, aquele arquivo precisa sofrer uma alteração para funcionar corretamente com o projeto:

- O comando `TARGET = testportb` deve ser alterado de modo a conter o nome do projeto, do será gerado o aquivo `.hex`. Não é necessário que o nome do projeto seja o mesmo do arquivo fonte principal. Por exemplo, se fizer `TARGET = projeto`, o resultado final da compilação será o arquivo `projeto.hex`;
- O comando `OPT = s` indica que o arquivo final gerado será o de menor tamanho possível (*size optimization*), sem no entanto ser mais rápido. Entretanto, `OPT = 0` corresponde a não haver otimização de código e `OPT = 2` resulta em geral em código mais rápido;
- O comando `SRC = main.c` deve conter o nome do arquivo principal do projeto. No caso em questão, deve-se fazer `SRC = teste.c`. Se o projeto for modular, todos os arquivos fonte devem ser enumerados aqui.

Com as alterações acima, o programa está pronto para ser compilado através da opção "[WinAVR] Make All" do menu Tools. Se nenhum erro ocorrer, o resultado `projeto.hex` poderá ser gravado no microcontrolador usando o programa Avrdude. Para tanto, basta chamar a opção "[WinAVR] Program" do menu Tools.

4.2 Registros de configuração

O ATmega8 possui alguns registros de configuração que não são acessíveis pelo programa gravado na memória flash. Estes registros definem a configuração de hardware do microcontrolador, habilitando ou não algumas de suas características. Estes registros são descritos na seção *Memory Programming* do manual [Atmel 2004]. São registros de proteção de código e dados (*Lock bit byte*), fusíveis de configuração (*Fuse High byte* e *Fuse Low Byte*), bytes de assinatura do dispositivo e byte de calibração do oscilador interno. Destes registros, um dos mais usados é o *Fuse Low byte* pois determina principalmente a principal fonte de relógio da CPU.

Para alterar qualquer um dos registros de configuração, isto somente pode ser feito através do programa `avrdude.exe` operando no modo *unsafe* (opção -u). Por exemplo, a seqüência de operações abaixo permite obter os valores gravados nos registros *Fuse High byte* (`hfuse` no avrdude) e *Fuse Low byte* (`lfuse` no avrdude):

```
C:\>avrdude.exe -p atmega8 -c bsd -P lpt1 -t -u
avrdude.exe: AVR device initialized and ready to accept instructions
Reading | ################################################## | 100% 0.00s
avrdude.exe: Device signature = 0x1e9307
avrdude> d hfuse
>>> d hfuse
0000 d9 |+ |
avrdude> d lfuse
>>> d lfuse
```

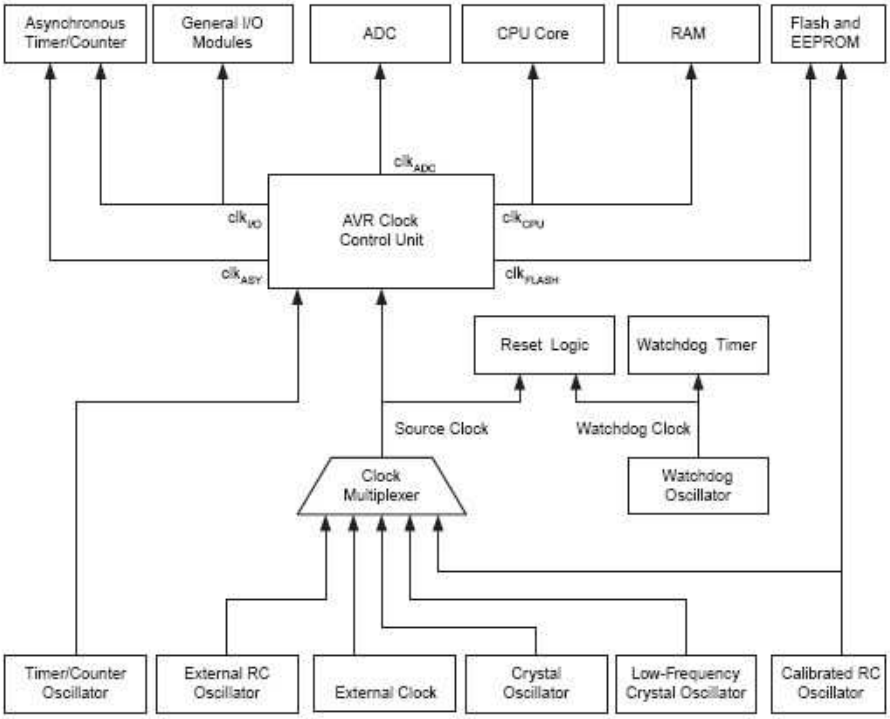


Figura 10: Distribuição dos relógios [Atmel 2004].

```
0000 e1 |b |
avrdude> quit
>>> quit
avrdude.exe done. Thank you.
```

No exemplo acima, a opção `-t` permite entrar no modo terminal, em que o usuário usa alguns comandos para manipular os registros. Para conhecer melhor o avrdude, sugere-se uma leitura no manual `avrdude.pdf` que se encontra no diretório `Docs\Avrdude` do CD de iniciação.

Um procedimento comum consiste em alterar o valor do *Fuse Low byte* para redefinir a fonte de relógio do microcontrolador. Isto pode ser feito no modo terminal como no exemplo acima, mas também usando linha de comando através de

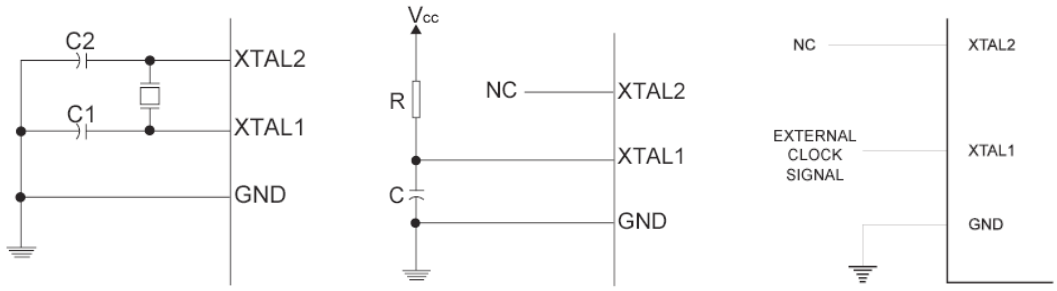
```
C:\>avrdude.exe -p atmega8 -c bsd -P lpt1 -t -U lfuse:w:0xef:m
```

Este comando muda a fonte de relógio para cristal externo, o que permite fazer o ATmega8 operar a até 16MHz. A fonte de relógio é definida por alguns bits do registro `lfuse`, conforme mostrado na próxima seção.

4.3 Seleção da freqüência do relógio

A freqüência do relógio determina, em última instância, a velocidade de execução do programa no microcontrolador. No microcontrolador AVR, o sinal de relógio pode ser obtido através de cinco diferentes fontes (ver Figura 10):

- *Circuito RC interno calibrado*, que é um circuito que gera algumas freqüências padrão utilizando um oscilador interno baseado em resistor e capacitor. A precisão é da ordem de $\pm 3\%$;
- *Cristal externo*, segundo conexão mostrada na Figura 11(a). Esta fonte de relógio é a de maior precisão, podendo chegar a até 16MHz. Quando o bit CKOPT do registro *Fuse High byte* é programado em 0, a saída XTAL2 possui faixa de variação de 0 a VCC, tornando o



(a) (b) (c)

Figura 11: Conexões de fontes externas de relógio: (a) Cristal/ressonador externo, C1 = C2 = 22pF, (b) Circuito RC externo com freqüência dada por f = 1/(3RC) e (c) Oscilador externo [Atmel 2004].

microcontrolador mais imune a interferência externa e permitindo XTAL2 ser conectado a uma porta CMOS/TTL. Por outro lado, com CKOPT programado em 1, a faixa de variação de XTAL2 é bem pequena, o que implica também em menor consumo de energia;

- *Circuito RC externo*, que requer um resistor e capacitor ligados externamente conforme mostrado na Figura 11(b). A freqüência do relógio passa a ser f = 1/(3RC), mas com pequena precisão. Esta configuração é interessante quando se deseja trabalhar com baixas freqüências, o que implica em muito baixo consumo;

- *Oscilador externo*, que gera uma onda quadrada a ser aplicada ao pino 9 (XTAL1) do Atmega8, conforme mostrado na Figura 11(c). O período da onda quadrada não pode sofre alteração de mais de 2% de seu período entre dois ciclos consecutivos. Se isto ocorrer, o comportamento do microcontrolador será imprevisível;

- *Ressonador cerâmico externo*, que apresenta o mesmo tipo de conexão que o modo cristal externo. Ressonadores cerâmicos são mais baratos, embora não tão precisos quanto cristais de quartzo.

Ainda de acordo com a Figura 10, cada um dos sinais de relógio gerados pode ser desligado independentemente (ver seção *Power Managament and Sleep Modes*, em [Atmel 2004]). Além disso, é importante destacar que o *Watchdog Timer*, que funciona como um dispositivo de proteção que reinicia o microcontrolador após o não cumprimento de determinada condição em um intervalo de tempo especificado, é conectado diretamente à fonte de relógio, e não à Unidade de Controle do Relógio, para proporcionar maior confiabilidade ao sistema.

Deve ser observado que o ATmega8 já vem de fábrica configurado para fonte de relógio dada pelo circuito RC interno a 1MHz [Atmel 2004]. Para alterar a fonte de relógio a ser utilizada, deve-se escrever no registro de configuração *Fuse Low Byte*, conforme ilustrado pela Seção 4.2. A Tabela 1 apresenta as fontes de relógio e o conteúdo que deve conter os bits CKOPT (registro *Fuse High Byte*) e CKSEL3, CKSEL2, CKSEL1 e CKSEL0 (registro *Fuse Low Byte*). Algumas vezes, alguma efeito externo provoca a gravação do *Fuse Low Byte* para 00h, que implica no uso de fonte externa de relógio. Se isto ocorrer, um procedimento que vem sendo usado no LCVC consiste em aplicar uma onda quadrada de aproximadamente 1MHz em XTAL1 e refazer a programação dos registros usando o `avrdude.exe`.

Uma vez alterada a fonte do relógio, a alteração somente tem efeito após reiniciar o microcontrolador. Isto significa que da próxima vez que avrdude for usado para gravação, a nova fonte de relógio já passa a ser utilizada. O mesmo ocorre com o programa da Flash, que passa a operar com a nova fonte de relógio selecionada.

Tabela 1: Seleção da fonte de relógio e bits CKOPT, CKSEL3..0 do registro *Low Fuse Byte*.

		Bit	**CKSEL**			
Fonte de relógio	**Freqüência**	**CKOPT**	3	2	1	0
Oscilador externo no pino 9 (XTAL1)	0 - 16 MHz	X	0	0	0	0
Circuito RC interno calibrado	1,0 MHz	X	0	0	0	1
Circuito RC interno calibrado	2,0 MHz	X	0	0	1	0
Circuito RC interno calibrado	4,0 MHz	X	0	0	1	1
Circuito RC interno calibrado	8,0 MHz	X	0	1	0	0
Circuito RC externo	$<$ 0,9 MHz	X	0	1	0	1
Circuito RC externo	0,9 MHz - 3,0 MHz	X	0	1	1	0
Circuito RC externo	3,0 MHz - 8,0 MHz	X	0	1	1	1
Circuito RC externo	8,0 MHz - 12,0 MHz	X	1	0	0	0
Cristal externo (baixa freqüência)	alguns KHz	X	1	0	0	1
Cristal externo (média freqüência)	0,9 MHz - 3,0 MHz	X	1	1	0	1
Cristal externo (alta freqüência)	3,0 MHz - 16,0 MHz	X	1	1	1	1
Ressonador cerâmico externo	0,4 MHz - 0,9 MHz	1	1	0	1	0

5 Funcionalidades particulares

Esta seção pretende ilustrar o uso de algumas das principais funções disponibilizadas pelo AVR. Essas funções são, em alguns casos, implementações em hardware de funcionalidades que até poderiam ser executadas por software, mas que, dessa forma, poderiam resultar em um código bastante complexo, além de um custo computacional consideravelmente maior.

5.1 Interrupções

Um aspecto muito importante na utilização de um microcontrolador é o tratamento das interrupções, que podem ser tanto internas quanto externas. Pode-se descrever interrupções como sendo desvios condicionais efetuados pelo programa em função da ocorrência de um fenômeno prioritário ocorrido em um determinado instante. As interrupções são muito importantes, pois elas permitem o processamento adequado em resposta a eventos internos ou externos, aumentado a possibilidade de interação com o meio.

De acordo com [Atmel 2004], como medida de segurança, existe para cada interrupção um vetor de registradores único. Essa configuração impede a execução de uma nova instância da mesma interrupção quando outra estiver sendo executada. Um outro fato importante é a existência de prioridades na execução das interrupções, ou seja, caso a prioridade de uma interrupção seja maior que a de outra, o seu endereço é colocada no vetor geral de interrupção como prioritária e conseqüentemente será processada primeiro em caso dos eventos ocorrerem simultaneamente.

Para cada fonte de interrupção, existe um bit responsável por sua habilitação. Este bit fica em algum dos registros do periférico associado à interrupção. Mas também existe o bit I do registro SREG (registro de estado do microcontrolador) que, quando setado em 1, habilita a ocorrência de qualquer interrupção que tenha sido habilitada em seu periférico reponsável. Se este bit for colocado em 0, então nenhuma solicitação de interrupção será atendida, mesmo se a interrupção for habilitada em seu periférico. Se ocorrer uma solicitação de interrupção habilitada com o bit I em 0, então o atendimento somente ocorrerá quando o bit I for colocado em 1. Para garantir atomicidade no procedimento de alteração do bit I em linguagem C, deve-se usar as funções `sei()` e `cli()`, que correspondem às instruções *assembly* `sei` (*set I bit*) e `cli` (*clear I bit*). Asism, `sei()` habilita globalmente as interrupções, enquanto que `cli()` desabilita globalmente as interrupções.

Nesta seção serão tratadas apenas as interrupções externas, visto que a maior parte dos outras interrupções disponíveis estão relacionadas às funcionalidades descritas nas próximas seções. Para

configurar uma interrupção externa, é necessário atentar aos registradores MCUCR e GICR. O registrador MCUCR contém os bits que configuram o modo de ativação das interrupções 0 e 1, que são associadas a eventos externos. Em outras palavras, alterando os bits ISCxx, segundo o *External Interrupts* de [Atmel 2004], pode-se configurar a interrupção para ocorrer quando há transição de subida ou de descida do sinal, ou mesmo quando ele se mantém em nível baixo. Já o registrador GICR é aquele em que se permite a ocorrência de determinada interrupção, por meio dos bits INT1 e INT0.

O trecho de código abaixo ilustra um simples exemplo de utilização de interrupções: utiliza-se um botão para alterar o estado de um LED.

```
...
SIGNAL(SIG_INTERRUPT0){ // Funcao que responde aa interrupcao externa 0
   PORTB = ~PORTB;
}
void main(void){
   // Configurando a interrupcao
   MCUCR = _BV(ISC01) | _BV(ISC00); // Transicao positiva em PD2 (INT0)
   GICR = _BV(INT0); // habilita interrupcao INT0
   sei(); // habilitacao global das interrupcoes

   // Definindo Port B como saída
   DDRB = _BV(DDB1);
   for(;;){
   }
}
```

Um outro uso das interrupções externas é a emulação da instrução assembly `SWI` (*Software Interrupt*), que não existe no conjunto de instruções do AVR. Para tal, basta setar o pino PD2 como saída e ativar a interrupção INT0, por exemplo. Nesta situação, poderá executar o código da interrupção controlando o estado do pino PD2 por meio de instruções de escrita a este pino.

No programa acima, a macro `SIGNAL(SIG_INTERRUPT0)` permite definir a função que fará o processamento da interrupção externa 0 quando ela ocorrer. De acordo com o manual da biblioteca LibC, disponível em `\WinAVR\doc\avr-libc\avr-libc-user-manual\index.html`, existem duas macros para definir a função que irá tratar uma determinada interrupção: `SIGNAL()` e `INTERRUPT()`. A principal diferença entre as duas é que `INTERRUPT()` define uma função de processamento de interrupção que é executada com a máscara global de interrupções ativada, o que permite que outras interrupções ocorram durante a execução da rotina de interrupção. Se isto ocorrer, diz-se que o sistema permite ninho de interrupções. Com `SIGNAL()`, a máscara global de interrupções é desativada, impedindo assim o atendimento de outras interrupções durante a execução da rotina de interrupção. Se ocorrer uma solicitação de interrupção durante a execução de uma rotina `SIGNAL()`, a mesma somente será atendida quando a rotina atual se encerrar. A desvantagem do uso de `SIGNAL()` está em um maior atraso ao atendimento de interrupções, mas isto apresneta vantagem pois reduz a necessidade de quantidade de memória RAM necessária para armazenar o contexto das instâncias interrompidas.

5.2 Temporizadores/Contadores (não revisado)

O microcontrolador possui três temporizadores, sendo dois de 8 bits e um de 16 bits. O mais simples deles é o *timer/counter* 0. Ele não possui módulo comparador, o que não possibilita que seja utilizado um evento externo para modificar o estado do contador. Um exemplo da aplicação deste módulo é a geração de sinais PWM (*Pulse Width Modulation*), muito utilizado para o controle de velocidade de motores DC. Essa operação pode, entretanto, ser efetuada por software, acarretando maior custo computacional.

Cada um desses dispositivos pode ser controlado tanto por uma fonte interna ou externa de relógio. Na maior parte dos casos, entretanto, em que se deseja alterar a base de tempo do timer, utiliza-se a mesma fonte de relógio do programa conjuntamente com o chamado *prescaler*, que se trata basicamente de um divisor de freqüência que altera o período de atualização do timer. A definição do valor do *prescaler* é feita por meio do registrador TCCRX. Caso o relógio de referência seja $8,000$ MHz e o prescaler seja configurado em 8, por exemplo, a freqüência resultante será dada por:

$$f = \frac{8\ MHz}{8} = 1\ MHz$$

5.2.1 Funções Periódicas

Uma das interessantes aplicações dos *timers/counters* disponíveis no AVR é a geração de funções periódicas. Essas funções podem ser utilizadas tanto para aquisição de dados e atuação em sistemas de controle de acordo com taxas de amostragem de período fixo, quanto para contagem de pulsos em determinado intervalo de tempo para medição de velocidade de rotação de um motor, por exemplo.

Mais especificamente, a função periódica nada mais é do que um interrupção gerada pelo temporizador quando este atinge o valor máximo de contagem (*overflow*). Para o caso do *timer* 0, por exemplo, para ativar esta função, basta garantir que o bit TOIE0 do registrador TIMSK esteja escrito com o valor 1. Desta forma, no momento em que o contador ultrapassar seu valor máximo, será executado o código contido na macro SIGNAL(TIMER _OVERFLOW0). Caso o relógio de referência seja $8.000KHz$ e o prescaler seja configurado em 8, por exemplo, o período de amostragem da função será dado por:

$$T_a = \left(\frac{8000}{8 \cdot 256}\right)^{-1} = 0,256\ ms$$

Caso seja do interesse utilizar taxas de amostragem mais precisas, além de alterar os valores do relógio e do *prescaler*, pode-se, por exemplo, escrever determinado valor no registrador TCNT0 toda vez em que ocorrer o sinal de *overflow*. Assim, cada contagem se dará a partir do valor escrito, e não do zero, que é o padrão.

A configuração destas funcionalidades nos outros dois timers é similar. Basta consultar [Atmel 2004] para verificar os registradores envolvidos.

5.2.2 *PWM*

Com exceção do *timer/counter* 0, os outros temporizadores trazem consigo a funcionalidade da configuração do PWM. Como já foi dito, este tipo de modulação é muito utilizado para o acionamento de motores de corrente contínua. Um exemplo de circuito utilizado para tal finalidade com o uso do AVR e do circuito integrado L298 encontra-se disponível no circuito do Anexo E. Neste exemplo, implementou-se um controle bidirecional de velocidade (um ciclo de trabalho de 50 % mantém o motor parado e o sinal do pino 14 deve ser o complemento do sinal do pino 15).

A geração do sinal PWM com o AVR se baseia na comparação dos estados dos contadores com valores pré-determinados. Assim, o resultado dessa comparação é utilizado para alterar a saída do PWM e, em última instância, definir o ciclo de trabalho do sinal PWM gerado.

Neste contexto, o ATmega8 permite a geração de três sinais PWM simultâneos. Dois deles são gerados com o *timer/counter* 1, que permite resolução de 16 bits, e o outro com o *timer/counter* 2, cuja resolução permitida é de 8 bits. A configuração dessa função é relativamente simples, bastando atentar às tabelas de configuração referentes aos registradores TCCR1A, TCCR1B ou TCCR2, dependendo do caso, disponíveis em [Atmel 2004]. Abaixo demonstra-se um exemplo de função para inicializar a geração de dois sinais PWM simultâneos a partir do *timer/counter* 1. Para alterar o valor da taxa de trabalho, basta escrever nos registradores OCR1AH e OCR1AL para o sinal A e OCR1BH e OCR1BL para o sinal B.

```
void PWM_Iniciar (void){
   // valores iniciais dos sinais PWM
   OCR1AH = 0x03;
   OCR1AL = 0xFF;
   OCR1BH = 0x03;
   OCR1BL = 0xFF;
   // configuracao do PWM
   TCCR1A = _BV(COM1A1) | _BV(COM1A0) | _BV(COM1B1) | _BV(COM1B0) | _BV(WGM10) | _BV(WGM11);
   TCCR1B = _BV(CS11) | _BV(WGM12);
   // set PB1 e PB2 as output
   DDRB = _BV(DDB1) | _BV(DDB2);
}
```

Na configuração acima descrita, o *prescaler* foi configurado para 8 (CS11) e é utilizado o *Fast PWM* de 10 bits (WGM10,11,12). Os bits COMXXX são utilizados para definir a saída da operação de comparação.

5.3 Conversão A/D

Muitos sistemas que utilizam microcontroladores requerem a medição de variáveis de natureza analógica. Alguns exemplos são a temperatura medida por um termômetro, a posição angular medida por um potenciômetro, entre outros. Neste contexto, para que essas informações possam ser utilizadas pelo microcontrolador, há a necessidade de se converter tais sinais para seus correspondentes digitais. O dispositivo utilizado para esta operação é o conversor analógico/digital, ou conversor A/D.

O AVR possui um conversor A/D com resolução de 10 bits. Esse mesmo conversor pode ser acessado por seis diferentes canais nos encapsulamentos PDIP. Ou seja, com um AVR ATmega8 neste encapsulamento pode-se medir facilmente seis sinais analógicos, porém não simultaneamente. Isso se dá porque existe um único conversor A/D com entradas multiplexadas. Além do mais o AVR possui um circuito *sample-and-hold* que mantém o valor analógico da entrada constante durante o processo de conversão, o que quer dizer que o projetista não precisa se preocupar quanto a esse aspecto no projeto do circuito. De fato, está disponível em anexo um simples circuito para conexão dos valores a serem medidos nos pinos correspondentes. Este circuito oferece certa proteção ao conversor A/D contra sub- e sobre-tensões. Além disso, a conversão A/D se utiliza uma fonte de tensão distinta (AVCC), cujo valor não pode diferir muito do valor de VCC, sendo na verdade a alimentação VCC passada por um filtro para minimizar flutuações na tensão de alimentação do conversor. O conversor A/D ainda requer uma tensão de referência analógica (AREF), que pode ser tanto interna como externa. De fato, o máximo valor obtido na conversão, que seria 1023 visto que o conversor A/D é de 10 bits, ocorrerá quando a entrada se igualar ao valor de AREF. Recomenda-se a consulta ao manual do microcontrolador [Atmel 2004] para melhor compreender o funcionamento do conversor A/D e atentar para restrições de operação, sob o risco de danificar permanentemente o conversor.

A utilização das funções de conversão A/D também não é complicada, conforme mostra o exemplo a seguir:

```
void AD_Iniciar (void) {
   // enable ADC e prescaler com division factor 64(ADPS2 e ADPS1)
   ADCSRA = _BV(ADEN) | _BV(ADPS2) | _BV(ADPS1);
   // setar referência
   ADMUX = _BV(REFS0);
}

void AD_Converter(unsigned char canal, unsigned char *pdatah, unsigned char *pdatal){
   ADMUX  = _BV(REFS0) | (canal & 0x0F);
   ADCSRA |= (0x01<<ADSC);
   // esperar fim de conversao
   while((ADCSRA & (0x40)));
```

```
   // copiar resultado
   *pdatal = ADCL; // deve ser lido antes de ADCH
   *pdatah = ADCH;
}

void main (void) {
   unsigned char canal,convh, convl;
   ...
   AD_Iniciar();
   ...
   while(true){
      ...
      canal = 1;
      AD_Converter(canal,&convh,&convl);
      ...
   }
}
```

Deve ser observado que o programa acima pode ser usado com o circuito de teste da seção E, que faz uso do canal 1 do conversor. No CD, o projeto exemplo `voltmeter` é mais completo pois corresponde a um voltímetro com display de LEDs. No entanto, o projeto `voltmeter` usa o canal 0 do conversor A/D.

5.4 Comunicação Serial (não revisado)

A fim de que o AVR envie e receba dados por meio de sua interface serial, basta configurar os registradores corretos. Neste sentido, os principais bits a serem ativados são o RXEN e o TXEN do registrador UCSRB, que liberam a recepção e a transmissão, respectivamente. Além disso, é preciso indicar o *baud rate* utilizado por meio dos registradores UBRRH e UBRRL (ver páginas 156-159 de [Atmel 2004] para tabela de valores), o tamanho da palavra, por meio dos bits UCSZ0, UCSZ1 e UCSZ2 do registrador UCSRC e o bit de parada por meio do bit USBS do mesmo registrador. A seguir disponibiliza-se exemplos de função para configurar a comunicação serial, enviar e receber dados:

```
void USART_Iniciar (void) {
   //Setar baud rate 38400
   UBRRH = (unsigned char)(25>>8);
   UBRRL = (unsigned char)(25);
   //Enable TXD, RXD e ativar RXCIE(enable interrupt on the RCX flag)
   UCSRB = _BV(TXEN) | _BV(RXEN); //| _BV(RXCIE);
   //Setar tamanho palavra (formatação da comunicação serial)
   UCSRC = _BV(URSEL) | _BV(UCSZ1) | _BV(UCSZ0) | _BV(USBS);
   //Ativar double speed
   UCSRA = _BV(U2X);
}

void USART_Transmitir (unsigned char c) {
   //Verificar se o buffer da serial está vazio
   while ( !(UCSRA & (1<<UDRE)) );
   //coloca o dado no buffer e envia
   UDR = c;
}

unsigned char USART_Receber (void) {
   //Esperar até que seja feita comunicação em RXD
   while ( !(UCSRA & (1<<RXC)) );
   //Recebe o dado do buffer
```

```
   return UDR;
}
```

Dos códigos acima, percebe-se que, para enviar e receber, basta ler ou escrever no registrador UDR, a partir do momento em que ele se encontra preenchido ou não, de acordo com o caso. Mais especificamente, UDRE é um bit que informa se o registrador em questão está vazio e RXC o bit que informa se a recepção se foi finalizada.

Para compatibilidade com o padrão RS-232, é necessária uma interface, que compõe a camada física do protocolo. Isto é devido ao fato de o protocolo RS-232 trabalhar com faixas de tensão distintas do padrão 0-5V da interface de comunicação serial. Com este objetivo, utiliza-se o circuito integrado MAX-232 e um conjunto de capacitores de $10\mu F$. O circuito correspondente encontra-se disponível no Anexo E.

Para testar o funcionamento do sistema de comunicação serial, recomenda-se a utilização de programas como o HyperTerminal, disponível no Windows ou o Terminal, disponível online.

5.5 Usando o PC como terminal de teclado/vídeo

Quando as funções da biblioteca STDIO direcionam seus canais *stdin* e *stdout* à porta serial do microcontrolador, pode-se utilizar funções tais como `printf` ou `scanf` para imprimir mensagens na tela ou ler teclas do teclado de um microcomputador PC. Nesta configuração, diz-se que o microcomputador está operando apenas como terminal. Para tanto, faz-se necessário conectar o ATmega8 à porta serial do PC por meio de um conversor de nível RS-232, tal como o MAX232. Isto é feito no circuito da Figura 12. No PC, faz-se necessário ter um programa terminal rodando, tal como o Hyperterminal. Em um programa terminal, tudo que for teclado é enviado à porta serial na forma de caractere ASCII. E também, todo dado que chegar pela porta serial é impresso na tela também como caractere ASCII.

O programa exemplo abaixo faz com que o ATmega8 reenvie ao PC tudo que for recebido pela porta serial. O exemplo está também disponível no CD de iniciação na forma do exemplo `terminal`.

```
#include <avr/io.h>
#include <stdio.h>

void USART_Init(void);
int USART_Transmit( char data );
int USART_Receive( void );

int main(void)
{
  char ch;

  USART_Init();

  /*registrando as funções de leitura e escrita de um byte pela serial*/
  fdevopen(USART_Transmit, USART_Receive, 0);

  //escrevendo "Ola, mundo!" pela serial
  printf ("Ola, mundo");

  while(1){
    // As duas linhas abaixo fazem a mesma coisa: o caracter que chegar
    // pela serial será ecoado por ela. Entretanto, as funções usadas são diferentes.
    // A diferença fundamental está na compexidade de cada uma delas.
    // Para ter uma melhor compreensão disto, descomente apenas uma das linhas abaixo,
    // e compile o programa para ver a diferença de tamanho do programa compilado.
    scanf ("%c", &ch);  printf ("%c", ch); // Solução 1.
//    putchar(getchar()); // Solução 2.
  }
```

```
}

/*
Função que faz a inicializcao da usart para comunicacao assincrona com o PC
a 9600 baud rate com 16MHz de cristal externo.
A recepção e transmissão são habilitadas, sobrepondo as funções dos pinos TxD e RxD.
Nenhuma interrupção é habilitada.
*/
void USART_Init(void)
{
  /*zerando o conteúdo do registrador de dados da serial*/
  UDR=0;
  //Seta baud rate de 9600 para fosc= 16MHz
  UBRRH = 0;
  UBRRL = 103;
  //Habilita transmissão e recepção serial
  UCSRB = _BV(TXEN)| _BV(RXEN);
  //Setar tamanho palavra
  //URSEL = 1 -> seleciona acesso para UCSRC (0 is UBRRH) reading and writing
  //UMSEL = 0 -> Asynchronous Operation
  //UPM1 e UPM0 = 00 -> sem paridade
  //USBS = 1 -> seleciona 2 bits de parada a ser inserido pelo transmissor
  //UCSZ 2 1 0 = 011 -> escolhe tamanho de caractere de 8 bits p/ dado
  UCSRC = _BV(URSEL) | _BV(UCSZ1) | _BV(UCSZ0);
}

// Faz a leitura de um byte da serial usando pooling
int USART_Receive( void )
{
  while ( !(UCSRA & (1<<RXC)) ); // Wait for data to be received
  return UDR; // Get and return received data from buffer
}

// Faz a transmissao de um byte pela serial
int USART_Transmit(  char data )
{
  while ( !( UCSRA & (1<<UDRE)) ); // Wait for empty transmit buffer
  UDR = data; // Put data into buffer, sends the data
  return 0;
}
```

Caso o usuário deseje imprimir no terminal números em ponto flutuante com a função printf, ele deve descomentar a seguinte linha do arquivo makefile (Anexo D):

```
#LDFLAGS += -Wl,-u,vfprintf -lprintf_flt
```

5.6 Display LCD

Uma das maneiras mais interessantes de apresentar informações relativas ao sistema utiliza displays de cristal líquido (LCDs). Neste sentido, esta subseção disponibiliza um exemplo que pode auxiliar aqueles que pretendem utilizar este dispositivo. Este exemplo faz uso da bilioteca `lcd.c` disponível no site `http://homepage.hispeed.ch/peterfleury/avr-lcd44780.html`. O exemplo está também disponível no CD de iniciação na forma do exemplo `lcdmsg`.

```
#include <inttypes.h>
#include <string.h>
#include <stdio.h>
#include <avr/io.h>
```

```
#define F_CPU 16.0E6 // relógio com cristal externo de 16MHz
#include <avr/delay.h>

#include "lcd.h"

void delay_us(unsigned int timedelay_us);
void delay_ms(unsigned int timedelay_ms);

int main (void)
{
   unsigned int segundos = 0, milisegundos = 0;
   char s[10];

   lcd_init(LCD_DISP_ON);
   lcd_clrscr();
   lcd_home();
   lcd_puts("Iniciando...");
   delay_ms(1000);
   lcd_clrscr();
   lcd_home();
   for(;;){
      delay_ms(100);
      if((milisegundos+=100)>=1000){
         milisegundos = 0;
         ++segundos;
      }
      sprintf(s,"%d:%2d",segundos, milisegundos/10);
      lcd_gotoxy(7-strlen(s)/2,0);
      lcd_puts(s);
   }
}

void delay_us(unsigned int timedelay_us)
{
   while(timedelay_us>1000){
      _delay_loop_2(4000); // 1ms
      timedelay_us-=1000;
   }
   timedelay_us = (4*timedelay_us);
   _delay_loop_2(timedelay_us);
}

void delay_ms(unsigned int timedelay_ms)
{
   while(--timedelay_ms>0)
      delay_us(1000); // 1ms;
}
```

6 Conclusões

Esta nota técnica apresentou algumas ferramentas básicas para desenvolvimento de projetos com microcontroladores AVR. A partir das informações disponibilizadas, o leitor é convidado a buscar as mais diversas aplicações, bem como novas funcionalidades não descritas neste documento. Este é, de fato, o maior objetivo daqueles que contribuíram para a elaboração deste documento e que foram efetivamente usuários do microcontrolador ATmega8.

Referências

[Atmel 2004]ATMEL. *ATmega8(L) Complete Datasheet*. [S.l.], 2004.

A CD de iniciação ao AVR

Para melhor acompanhar esta nota técnica, está sendo disponibilizado no site `http://www.ene.unb.br/~gaborges/recursos/embarcados/index.htm` um CD com todas as ferramentas de desenvolvimento, exemplos e documentação. Como o CD está disponível na internet, qualquer outra pessoa interessada pode baixá-lo. O arquivo `leiame.txt` detalha o conteúdo do CD e dá várias dicas de como fazer melhor uso do microcontrolador. No CD também existem vários exemplos de programas.

B Iniciação rápida

Para aqueles que querem se iniciar rapidamente com o desenvolvimento com AVR, mesmo que correndo todos os riscos associados à compreensão limitada do que está sendo realizado, sugere-se seguir os seguintes passos:

- Instalar o WinAVR (`Ferramentas - software\WinAVR`). Se houver uma versão anterior no microcomputador, a mesma deve ser desinstalada;
- Construir uma gravadora BSD (`Ferramentas - hardware\BSD prog simples`). Não esquecer de configurar na BIOS do seu computador que a porta paralela deve operar em modo Standard (geralmente usa-se a sigla SPP);
- Construir um circuito de referência (`Ferramentas - hardware\Circuito de referência ATMEGA8`);
- Testar o circuito de referência e a gravadora seguindo o procedimento da Seção 3.1.3.
- Usar o Programmer's Notepad para desenvolver o projeto (instalado com o WinAVR). Se o microcontrolador é novo, deve-se lembrar que sua configuração de fábrica implica que a fonte de relógio é um circuito RC interno a 1MHz. Uma recomendação para os novatos consiste em partir de um dos exemplos do CD de iniciação ao AVR;
- Verificar se o Makefile está adaptado ao projeto. Não esquecer de especificar o microcontrolador (no caso, ATmega8), a gravadora (no caso, bsd) e a porta em que ela está conectada (lpt1, se porta paralela 1);
- Ligue a alimentação do circuito com o microcontrolador. A gravadora BSD deve estar conectada ao microcontrolador e ao microcomputador;
- Use as opções [WinAVR] Make Clean, [WinAVR] Make All e [WinAVR] Make Program do menu Tools do Programmer's Notepad para limpar o projeto, compilar e gravar no microcontrolador.

Se tudo der certo, ao final dos procedimentos acima o programa deverá estar sendo executado no microcontrolador.

Estes passos foram escritos considerando que o usuário dispõe do CD de iniciação (Anexo A);

C Referências na Internet

- `http://www.atmel.com/` : página do fabricante, com manuais e notas de aplicação;
- `http://www.avrfreaks.net/` : a principal fonte para aqueles que buscam auxílio para o desenvolvimento com microcontrolador da série AVR©da Atmel;

- http://www.microschematic.com/ : apresentação em Flash indicada para aqueles que desejam programar em assembly;

- http://winavr.sourceforge.net/ : site oficial do pacote de desenvolvimento WinAVR;
- http://www.bsdhome.com/avrdude/ : site do programador AVRDUDE (BSD);
- http://www.freertos.org/ : site do FreeRTOS, um kernel que possui suporte à linha de microcontroladores AVR;
- http://members.home.nl/jmnieuwkamp1/AVRlib/ : site da biblioteca AVRlib, que contém um grande número de funções par uso e interface para microcontroladores AVR;
- http://homepage.hispeed.ch/peterfleury/index.html : site do Peter Fleury com projetos, exemplos. Tem uma placa de desenvolvimento e algumas bibliotecas.

D Exemplo de makefile

O arquivo `makefile` (sem extensão) é essencial para projetos em que o processo de compilação é mais complexo, embora seja também bastante útil para projetos simples. O `makefile` é um arquivo texto com diretivas que determinam como cada arquivo do projeto deve ser compilado, linkado e como o resultado final deve ser apresentado. Este arquivo é usado pelo utilitário `make.exe`, devendo estar no mesmo diretório da chamada, quando se executa algo do tipo `make procedimento` em que `procedimento` é uma palavra reservada que caracteriza o procedimento a ser realizado. De acordo com o arquivo `makefile` abaixo, estão definidos procedimentos `all`, `clean` e `program`, entre outros procedimentos intermediários:

- `make all`: procedimento de compilação de todos os arquivos fonte que foram alterados desde a última compilação, e geração de vários arquivos, entre eles a imagem `.hex` a ser gravada no microcontrolador.
- `make clean`: apaga todos os arquivos intermediários e resultantes de compilação, deixando apenas os arquivos fonte de um projeto. É um procedimento geralmente usado quando se deseja guardar apenas o necessário de um projeto para, talvez, enviar para alguém.
- `make program`: grava o programa no microcontrolador utilisando a gravadora configurada no `makefile`.

O arquivo `makefile` usado como exemplo foi extraído de um exemplo do pacote WinAVR. Para compreendê-lo, faz-se necessário saber como o utilitário `make.exe` funciona. Para tanto, sugere-se o site `http://www.gnu.org/software/make/manual/make.html`. Entretanto, o arquivo está muito bem comentado.

```
# Hey Emacs, this is a -*- makefile -*-
#----------------------------------------------------------------------------
# WinAVR Makefile Template written by Eric B. Weddington, Jörg Wunsch, et al.
#
# Released to the Public Domain
#
# Additional material for this makefile was written by:
# Peter Fleury
# Tim Henigan
# Colin O'Flynn
# Reiner Patommel
# Markus Pfaff
# Sander Pool
```

```
# Frederik Rouleau
# Carlos Lamas
#
#-----------------------------------------------------------------------------
# On command line:
#
# make all = Make software.
#
# make clean = Clean out built project files.
#
# make coff = Convert ELF to AVR COFF.
#
# make extcoff = Convert ELF to AVR Extended COFF.
#
# make program = Download the hex file to the device, using avrdude.
#                Please customize the avrdude settings below first!
#
# make debug = Start either simulavr or avarice as specified for debugging,
#              with avr-gdb or avr-insight as the front end for debugging.
#
# make filename.s = Just compile filename.c into the assembler code only.
#
# make filename.i = Create a preprocessed source file for use in submitting
#                   bug reports to the GCC project.
#
# To rebuild project do "make clean" then "make all".
#-----------------------------------------------------------------------------


# MCU name
MCU = atmega8


# Processor frequency.
#     This will define a symbol, F_CPU, in all source code files equal to the
#     processor frequency. You can then use this symbol in your source code to
#     calculate timings. Do NOT tack on a 'UL' at the end, this will be done
#     automatically to create a 32-bit value in your source code.
#     Typical values are:
#         F_CPU =  1000000
#         F_CPU =  1843200
#         F_CPU =  2000000
#         F_CPU =  3686400
#         F_CPU =  4000000
#         F_CPU =  7372800
#         F_CPU =  8000000
#         F_CPU = 11059200
#         F_CPU = 14745600
         F_CPU = 16000000
#         F_CPU = 18432000
#         F_CPU = 20000000


# Output format. (can be srec, ihex, binary)
FORMAT = ihex


# Target file name (without extension).
TARGET = voltmeter


# Object files directory
#     To put object files in current directory, use a dot (.), do NOT make
```

```
#     this an empty or blank macro!
OBJDIR = .


# List C source files here. (C dependencies are automatically generated.)
SRC = main.c


# List C++ source files here. (C dependencies are automatically generated.)
CPPSRC =


# List Assembler source files here.
#     Make them always end in a capital .S.  Files ending in a lowercase .s
#     will not be considered source files but generated files (assembler
#     output from the compiler), and will be deleted upon "make clean"!
#     Even though the DOS/Win* filesystem matches both .s and .S the same,
#     it will preserve the spelling of the filenames, and gcc itself does
#     care about how the name is spelled on its command-line.
ASRC =


# Optimization level, can be [0, 1, 2, 3, s].
#     0 = turn off optimization. s = optimize for size.
#     (Note: 3 is not always the best optimization level. See avr-libc FAQ.)
OPT = 2


# Debugging format.
#     Native formats for AVR-GCC's -g are dwarf-2 [default] or stabs.
#     AVR Studio 4.10 requires dwarf-2.
#     AVR [Extended] COFF format requires stabs, plus an avr-objcopy run.
DEBUG = dwarf-2


# List any extra directories to look for include files here.
#     Each directory must be seperated by a space.
#     Use forward slashes for directory separators.
#     For a directory that has spaces, enclose it in quotes.
EXTRAINCDIRS =


# Compiler flag to set the C Standard level.
#     c89   = "ANSI" C
#     gnu89 = c89 plus GCC extensions
#     c99   = ISO C99 standard (not yet fully implemented)
#     gnu99 = c99 plus GCC extensions
CSTANDARD = -std=gnu99


# Place -D or -U options here for C sources
CDEFS = -DF_CPU=$(F_CPU)UL


# Place -D or -U options here for ASM sources
ADEFS = -DF_CPU=$(F_CPU)


# Place -D or -U options here for C++ sources
CPPDEFS = -DF_CPU=$(F_CPU)UL
#CPPDEFS += -D__STDC_LIMIT_MACROS
#CPPDEFS += -D__STDC_CONSTANT_MACROS
```

```
#---------------- Compiler Options C ----------------
#  -g*:          generate debugging information
#  -O*:          optimization level
#  -f...:        tuning, see GCC manual and avr-libc documentation
#  -Wall...:     warning level
#  -Wa,...:      tell GCC to pass this to the assembler.
#    -adhlns...: create assembler listing
CFLAGS = -g$(DEBUG)
CFLAGS += $(CDEFS)
CFLAGS += -O$(OPT)
CFLAGS += -funsigned-char
CFLAGS += -funsigned-bitfields
CFLAGS += -fpack-struct
CFLAGS += -fshort-enums
CFLAGS += -Wall
CFLAGS += -Wstrict-prototypes
#CFLAGS += -mshort-calls
#CFLAGS += -fno-unit-at-a-time
#CFLAGS += -Wundef
#CFLAGS += -Wunreachable-code
#CFLAGS += -Wsign-compare
CFLAGS += -Wa,-adhlns=$(<:%.c=$(OBJDIR)/%.lst)
CFLAGS += $(patsubst %,-I%,$(EXTRAINCDIRS))
CFLAGS += $(CSTANDARD)


#---------------- Compiler Options C++ ----------------
#  -g*:          generate debugging information
#  -O*:          optimization level
#  -f...:        tuning, see GCC manual and avr-libc documentation
#  -Wall...:     warning level
#  -Wa,...:      tell GCC to pass this to the assembler.
#    -adhlns...: create assembler listing
CPPFLAGS = -g$(DEBUG)
CPPFLAGS += $(CPPDEFS)
CPPFLAGS += -O$(OPT)
CPPFLAGS += -funsigned-char
CPPFLAGS += -funsigned-bitfields
CPPFLAGS += -fpack-struct
CPPFLAGS += -fshort-enums
CPPFLAGS += -fno-exceptions
CPPFLAGS += -Wall
CFLAGS += -Wundef
#CPPFLAGS += -mshort-calls
#CPPFLAGS += -fno-unit-at-a-time
#CPPFLAGS += -Wstrict-prototypes
#CPPFLAGS += -Wunreachable-code
#CPPFLAGS += -Wsign-compare
CPPFLAGS += -Wa,-adhlns=$(<:%.cpp=$(OBJDIR)/%.lst)
CPPFLAGS += $(patsubst %,-I%,$(EXTRAINCDIRS))
#CPPFLAGS += $(CSTANDARD)


#---------------- Assembler Options ----------------
#  -Wa,...:   tell GCC to pass this to the assembler.
#  -adhlns:   create listing
#  -gstabs:   have the assembler create line number information; note that
#             for use in COFF files, additional information about filenames
#             and function names needs to be present in the assembler source
#             files -- see avr-libc docs [FIXME: not yet described there]
#  -listing-cont-lines: Sets the maximum number of continuation lines of hex
```

```
#       dump that will be displayed for a given single line of source input.
ASFLAGS = $(ADEFS) -Wa,-adhlns=$(<:%.S=$(OBJDIR)/%.lst),-gstabs,--listing-cont-lines=100


#---------------- Library Options ----------------
# Minimalistic printf version
PRINTF_LIB_MIN = -Wl,-u,vfprintf -lprintf_min

# Floating point printf version (requires MATH_LIB = -lm below)
PRINTF_LIB_FLOAT = -Wl,-u,vfprintf -lprintf_flt

# If this is left blank, then it will use the Standard printf version.
PRINTF_LIB =
#PRINTF_LIB = $(PRINTF_LIB_MIN)
#PRINTF_LIB = $(PRINTF_LIB_FLOAT)


# Minimalistic scanf version
SCANF_LIB_MIN = -Wl,-u,vfscanf -lscanf_min

# Floating point + %[ scanf version (requires MATH_LIB = -lm below)
SCANF_LIB_FLOAT = -Wl,-u,vfscanf -lscanf_flt

# If this is left blank, then it will use the Standard scanf version.
SCANF_LIB =
#SCANF_LIB = $(SCANF_LIB_MIN)
#SCANF_LIB = $(SCANF_LIB_FLOAT)


MATH_LIB = -lm


# List any extra directories to look for libraries here.
#     Each directory must be seperated by a space.
#     Use forward slashes for directory separators.
#     For a directory that has spaces, enclose it in quotes.
EXTRALIBDIRS =



#---------------- External Memory Options ----------------

# 64 KB of external RAM, starting after internal RAM (ATmega128!),
# used for variables (.data/.bss) and heap (malloc()).
#EXTMEMOPTS = -Wl,-Tdata=0x801100,--defsym=__heap_end=0x80ffff

# 64 KB of external RAM, starting after internal RAM (ATmega128!),
# only used for heap (malloc()).
#EXTMEMOPTS = -Wl,--section-start,.data=0x801100,--defsym=__heap_end=0x80ffff

EXTMEMOPTS =



#---------------- Linker Options ----------------
#  -Wl,...:     tell GCC to pass this to linker.
#    -Map:      create map file
#    --cref:    add cross reference to  map file
LDFLAGS = -Wl,-Map=$(TARGET).map,--cref
LDFLAGS += $(EXTMEMOPTS)
LDFLAGS += $(patsubst %,-L%,$(EXTRALIBDIRS))
LDFLAGS += $(PRINTF_LIB) $(SCANF_LIB) $(MATH_LIB)
#LDFLAGS += -T linker_script.x
```

```
#---------------- Programming Options (avrdude) ----------------

# Programming hardware
# Type: avrdude -c ?
# to get a full listing.
#
AVRDUDE_PROGRAMMER = bsd

# com1 = serial port. Use lpt1 to connect to parallel port.
AVRDUDE_PORT = lpt1    # programmer connected to parallel port

AVRDUDE_WRITE_FLASH = -U flash:w:$(TARGET).hex
#AVRDUDE_WRITE_EEPROM = -U eeprom:w:$(TARGET).eep

# Uncomment the following if you want avrdude leave with /RESET=1 (microcontroller running).
AVRDUDE_EXIT_NO_RESET = -E noreset

# Uncomment the following if you want avrdude's erase cycle counter.
# Note that this counter needs to be initialized first using -Yn,
# see avrdude manual.
#AVRDUDE_ERASE_COUNTER = -y

# Uncomment the following if you do /not/ wish a verification to be
# performed after programming the device.
#AVRDUDE_NO_VERIFY = -V

# Increase verbosity level.  Please use this when submitting bug
# reports about avrdude. See <http://savannah.nongnu.org/projects/avrdude>
# to submit bug reports.
#AVRDUDE_VERBOSE = -v -v

AVRDUDE_FLAGS = -p $(MCU) -P $(AVRDUDE_PORT) -c $(AVRDUDE_PROGRAMMER)
AVRDUDE_FLAGS += $(AVRDUDE_NO_VERIFY)
AVRDUDE_FLAGS += $(AVRDUDE_VERBOSE)
AVRDUDE_FLAGS += $(AVRDUDE_ERASE_COUNTER)
AVRDUDE_FLAGS += $(AVRDUDE_EXIT_NO_RESET)


#---------------- Debugging Options ----------------

# For simulavr only - target MCU frequency.
DEBUG_MFREQ = $(F_CPU)

# Set the DEBUG_UI to either gdb or insight.
# DEBUG_UI = gdb
DEBUG_UI = insight

# Set the debugging back-end to either avarice, simulavr.
DEBUG_BACKEND = avarice
#DEBUG_BACKEND = simulavr

# GDB Init Filename.
GDBINIT_FILE = __avr_gdbinit

# When using avarice settings for the JTAG
JTAG_DEV = /dev/com1

# Debugging port used to communicate between GDB / avarice / simulavr.
DEBUG_PORT = 4242
```

```
# Debugging host used to communicate between GDB / avarice / simulavr, normally
#     just set to localhost unless doing some sort of crazy debugging when
#     avarice is running on a different computer.
DEBUG_HOST = localhost



#============================================================================


# Define programs and commands.
SHELL = sh
CC = avr-gcc
OBJCOPY = avr-objcopy
OBJDUMP = avr-objdump
SIZE = avr-size
AR = avr-ar rcs
NM = avr-nm
AVRDUDE = avrdude
REMOVE = rm -f
REMOVEDIR = rm -rf
COPY = cp
WINSHELL = cmd


# Define Messages
# English
MSG_ERRORS_NONE = Errors: none
MSG_BEGIN = -------- begin --------
MSG_END = --------  end  --------
MSG_SIZE_BEFORE = Size before:
MSG_SIZE_AFTER = Size after:
MSG_COFF = Converting to AVR COFF:
MSG_EXTENDED_COFF = Converting to AVR Extended COFF:
MSG_FLASH = Creating load file for Flash:
MSG_EEPROM = Creating load file for EEPROM:
MSG_EXTENDED_LISTING = Creating Extended Listing:
MSG_SYMBOL_TABLE = Creating Symbol Table:
MSG_LINKING = Linking:
MSG_COMPILING = Compiling C:
MSG_COMPILING_CPP = Compiling C++:
MSG_ASSEMBLING = Assembling:
MSG_CLEANING = Cleaning project:
MSG_CREATING_LIBRARY = Creating library:




# Define all object files.
OBJ = $(SRC:%.c=$(OBJDIR)/%.o) $(CPPSRC:%.cpp=$(OBJDIR)/%.o) $(ASRC:%.S=$(OBJDIR)/%.o)

# Define all listing files.
LST = $(SRC:%.c=$(OBJDIR)/%.lst) $(CPPSRC:%.cpp=$(OBJDIR)/%.lst) $(ASRC:%.S=$(OBJDIR)/%.lst)


# Compiler flags to generate dependency files.
GENDEPFLAGS = -MMD -MP -MF .dep/$(@F).d


# Combine all necessary flags and optional flags.
# Add target processor to flags.
ALL_CFLAGS = -mmcu=$(MCU) -I. $(CFLAGS) $(GENDEPFLAGS)
ALL_CPPFLAGS = -mmcu=$(MCU) -I. -x c++ $(CPPFLAGS) $(GENDEPFLAGS)
```

```
ALL_ASFLAGS = -mmcu=$(MCU) -I. -x assembler-with-cpp $(ASFLAGS)




# Default target.
all: begin gccversion sizebefore build sizeafter end

# Change the build target to build a HEX file or a library.
build: elf hex eep lss sym
#build: lib


elf: $(TARGET).elf
hex: $(TARGET).hex
eep: $(TARGET).eep
lss: $(TARGET).lss
sym: $(TARGET).sym
LIBNAME=lib$(TARGET).a
lib: $(LIBNAME)



# Eye candy.
# AVR Studio 3.x does not check make's exit code but relies on
# the following magic strings to be generated by the compile job.
begin:
@echo
@echo $(MSG_BEGIN)

end:
@echo $(MSG_END)
@echo


# Display size of file.
HEXSIZE = $(SIZE) --target=$(FORMAT) $(TARGET).hex
ELFSIZE = $(SIZE) --mcu=$(MCU) --format=avr $(TARGET).elf

sizebefore:
@if test -f $(TARGET).elf; then echo; echo $(MSG_SIZE_BEFORE); $(ELFSIZE); \
2>/dev/null; echo; fi

sizeafter:
@if test -f $(TARGET).elf; then echo; echo $(MSG_SIZE_AFTER); $(ELFSIZE); \
2>/dev/null; echo; fi



# Display compiler version information.
gccversion :
@$(CC) --version



# Program the device.
program: $(TARGET).hex $(TARGET).eep
$(AVRDUDE) $(AVRDUDE_FLAGS) $(AVRDUDE_WRITE_FLASH) $(AVRDUDE_WRITE_EEPROM)


# Generate avr-gdb config/init file which does the following:
#     define the reset signal, load the target file, connect to target, and set
```

```
#     a breakpoint at main().
gdb-config:
@$(REMOVE) $(GDBINIT_FILE)
@echo define reset >> $(GDBINIT_FILE)
@echo SIGNAL SIGHUP >> $(GDBINIT_FILE)
@echo end >> $(GDBINIT_FILE)
@echo file $(TARGET).elf >> $(GDBINIT_FILE)
@echo target remote $(DEBUG_HOST):$(DEBUG_PORT)  >> $(GDBINIT_FILE)
ifeq ($(DEBUG_BACKEND),simulavr)
@echo load  >> $(GDBINIT_FILE)
endif
@echo break main >> $(GDBINIT_FILE)

debug: gdb-config $(TARGET).elf
ifeq ($(DEBUG_BACKEND), avarice)
@echo Starting AVaRICE - Press enter when "waiting to connect" message displays.
@$(WINSHELL) /c start avarice --jtag $(JTAG_DEV) --erase --program --file \
$(TARGET).elf $(DEBUG_HOST):$(DEBUG_PORT)
@$(WINSHELL) /c pause

else
@$(WINSHELL) /c start simulavr --gdbserver --device $(MCU) --clock-freq \
$(DEBUG_MFREQ) --port $(DEBUG_PORT)
endif
@$(WINSHELL) /c start avr-$(DEBUG_UI) --command=$(GDBINIT_FILE)




# Convert ELF to COFF for use in debugging / simulating in AVR Studio or VMLAB.
COFFCONVERT = $(OBJCOPY) --debugging
COFFCONVERT += --change-section-address .data-0x800000
COFFCONVERT += --change-section-address .bss-0x800000
COFFCONVERT += --change-section-address .noinit-0x800000
COFFCONVERT += --change-section-address .eeprom-0x810000



coff: $(TARGET).elf
@echo
@echo $(MSG_COFF) $(TARGET).cof
$(COFFCONVERT) -O coff-avr $< $(TARGET).cof


extcoff: $(TARGET).elf
@echo
@echo $(MSG_EXTENDED_COFF) $(TARGET).cof
$(COFFCONVERT) -O coff-ext-avr $< $(TARGET).cof



# Create final output files (.hex, .eep) from ELF output file.
%.hex: %.elf
@echo
@echo $(MSG_FLASH) $@
$(OBJCOPY) -O $(FORMAT) -R .eeprom -R .fuse -R .lock -R .signature $< $@

%.eep: %.elf
@echo
@echo $(MSG_EEPROM) $@
-$(OBJCOPY) -j .eeprom --set-section-flags=.eeprom="alloc,load" \
--change-section-lma .eeprom=0 --no-change-warnings -O $(FORMAT) $< $@ || exit 0
```

```
# Create extended listing file from ELF output file.
%.lss: %.elf
@echo
@echo $(MSG_EXTENDED_LISTING) $@
$(OBJDUMP) -h -S -z $< > $@

# Create a symbol table from ELF output file.
%.sym: %.elf
@echo
@echo $(MSG_SYMBOL_TABLE) $@
$(NM) -n $< > $@



# Create library from object files.
.SECONDARY : $(TARGET).a
.PRECIOUS : $(OBJ)
%.a: $(OBJ)
@echo
@echo $(MSG_CREATING_LIBRARY) $@
$(AR) $@ $(OBJ)


# Link: create ELF output file from object files.
.SECONDARY : $(TARGET).elf
.PRECIOUS : $(OBJ)
%.elf: $(OBJ)
@echo
@echo $(MSG_LINKING) $@
$(CC) $(ALL_CFLAGS) $^ --output $@ $(LDFLAGS)


# Compile: create object files from C source files.
$(OBJDIR)/%.o : %.c
@echo
@echo $(MSG_COMPILING) $<
$(CC) -c $(ALL_CFLAGS) $< -o $@


# Compile: create object files from C++ source files.
$(OBJDIR)/%.o : %.cpp
@echo
@echo $(MSG_COMPILING_CPP) $<
$(CC) -c $(ALL_CPPFLAGS) $< -o $@


# Compile: create assembler files from C source files.
%.s : %.c
$(CC) -S $(ALL_CFLAGS) $< -o $@


# Compile: create assembler files from C++ source files.
%.s : %.cpp
$(CC) -S $(ALL_CPPFLAGS) $< -o $@


# Assemble: create object files from assembler source files.
$(OBJDIR)/%.o : %.S
@echo
@echo $(MSG_ASSEMBLING) $<
$(CC) -c $(ALL_ASFLAGS) $< -o $@
```

```
# Create preprocessed source for use in sending a bug report.
%.i : %.c
$(CC) -E -mmcu=$(MCU) -I. $(CFLAGS) $< -o $@


# Target: clean project.
clean: begin clean_list end

clean_list :
@echo
@echo $(MSG_CLEANING)
$(REMOVE) $(TARGET).hex
$(REMOVE) $(TARGET).eep
$(REMOVE) $(TARGET).cof
$(REMOVE) $(TARGET).elf
$(REMOVE) $(TARGET).map
$(REMOVE) $(TARGET).sym
$(REMOVE) $(TARGET).lss
$(REMOVE) $(SRC:%.c=$(OBJDIR)/%.o)
$(REMOVE) $(SRC:%.c=$(OBJDIR)/%.lst)
$(REMOVE) $(SRC:.c=.s)
$(REMOVE) $(SRC:.c=.d)
$(REMOVE) $(SRC:.c=.i)
$(REMOVEDIR) .dep


# Create object files directory
$(shell mkdir $(OBJDIR) 2>/dev/null)


# Include the dependency files.
-include $(shell mkdir .dep 2>/dev/null) $(wildcard .dep/*)


# Listing of phony targets.
.PHONY : all begin finish end sizebefore sizeafter gccversion \
build elf hex eep lss sym coff extcoff \
clean clean_list program debug gdb-config
```

E Circuito de teste

O circuito da Figura 12 é utilizado por alguns exemplos desta nota técnica para ilustrar funcionalidades do ATmega8. O mesmo pode ser montado em protoboard aproveitando-se o circuito de referência da Seção 3.1.2.

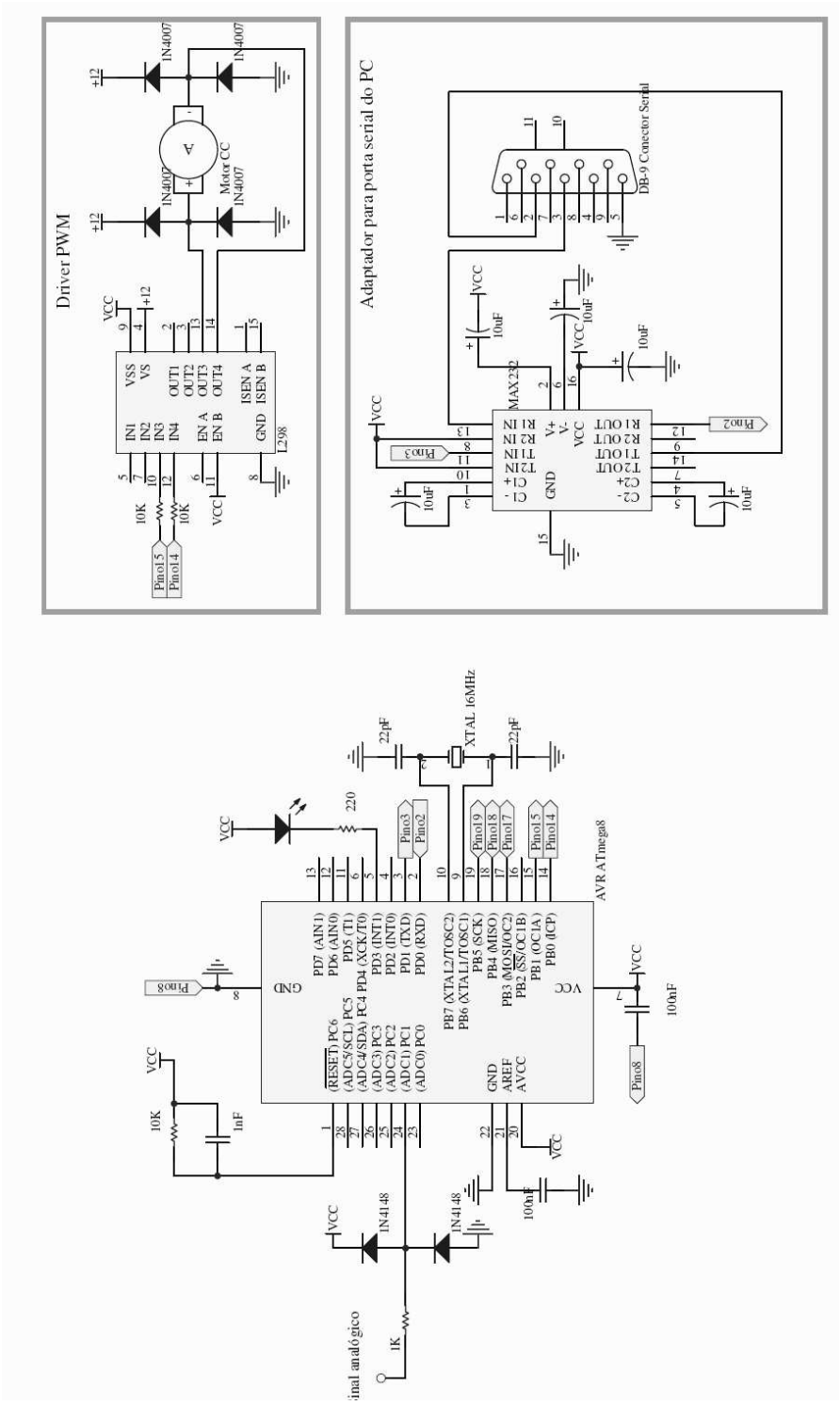


Figura 12: Circuito de teste com o microcontrolador, interface serial para PC e acionador por modulação em largura de pulso para motor de corrente contínua com escovas