

Procedimentos para programação em C para microcontroladores da linha AVR 8 bits da ATMEL

Introdução

O processo de escolha de um microcontrolador deve levar em conta vários fatores, entre eles o custo de aquisição, dificuldade de programação e recursos disponíveis. Os microcontroladores da linha AVR de 8 bits da ATMEL são uma boa escolha para as necessidades presentes do LMPT. São baratos (AT90S1200 \cong R\$6,00, AT90S8515 \cong R\$18,00 – maio 2003); sua programação é simples e há uma vasta gama de ferramentas para esse fim; possuem recursos poderosos, tais como temporizadores, contadores, conversores A/D, UARTS, etc; não necessitam de EPROMs para guardar os programas, pois possuem memória FLASH interna que é programada através de uma interface ISP, a qual é conectada na porta paralela ou USB de um micro, dependendo do modelo do microcontrolador. Essa última característica torna a utilização desses dispositivos muito interessante, pois diminui em muito o período de teste e prototipagem, quando comparados com microcontroladores que necessitam de EPROMs.

A ATMEL fornece kits de desenvolvimento para que o aprendizado sobre suas linhas de microcontroladores seja mais fácil; dois exemplos para a linha AVR de 8 bits são o STK 200 e o STK 300. Nesse curso introdutório será utilizado o STK 200, mas os conhecimentos que forem aqui adquiridos podem ser facilmente estendidos para os outros kits e microcontroladores da empresa. Há vários sites que disponibilizam uma grande quantidade de informações, sendo que, entre eles, considero estes três os mais interessantes:

www.atmel.com
www.avrfreaks.com
www.openavr.org

Ferramentas sugeridas para a programação

Da vasta gama de ferramentas que existem para o desenvolvimento de aplicações com os microcontroladores da ATMEL, sugiro que as seguintes sejam utilizadas, por serem de livre distribuição:

WinAVR: contém o compilador C e C++, assembler, linker e outros utilitários. Está se tornando uma suíte completa de desenvolvimento para os microcontroladores da ATMEL mas, no momento, ainda está em desenvolvimento. Será utilizado por causa do compilador.

AVR STUDIO (versão 3): é a suíte de desenvolvimento fornecida pela ATMEL. Contém o editor de textos, simulador dos microcontroladores e é capaz de compilar códigos em assembler.

PonyProg2000: Carrega o programa compilado no microcontrolador. Está disponível no site www.lancos.com.

Instalação das ferramentas

Todas essas ferramentas são distribuídas na forma de programas auto-instaladores. O processo de instalação é simples e basta seguir as instruções fornecidas durante o processo.

Criação de um novo projeto para código C / C++

A criação de um novo projeto é uma tarefa que envolve um certo número de passos, já que as ferramentas de edição / depuração e o compilador C / C++ não são integradas. No entanto, uma vez que o projeto seja criado o desenvolvimento do código é feito somente dentro do AVR Studio.

Para a criação de um novo projeto, os seguintes passos devem ser seguidos:

1- Após abrir o AVR Studio, clicar em *Project->New*. Na janela *Select New Project*, colocar o nome do novo projeto e a sua localização. É importante que cada projeto tenha o seu próprio diretório. Depois selecionar, em *Project type*, a opção *Generic 3rd party C compiler*.

2 – Na janela *Project*, há um item chamado *Target*. Clicar nesse item com o botão direito do mouse, e, após o aparecimento do menu de contexto, selecionar a opção *Targets->Add* (Figura 1). Na caixa de diálogo *Add New Target* digite *all* no campo *Name* e selecione *debug* no campo *copy settings from*. Repita esse procedimento mas digitando dessa vez *clear* no campo *Name*.

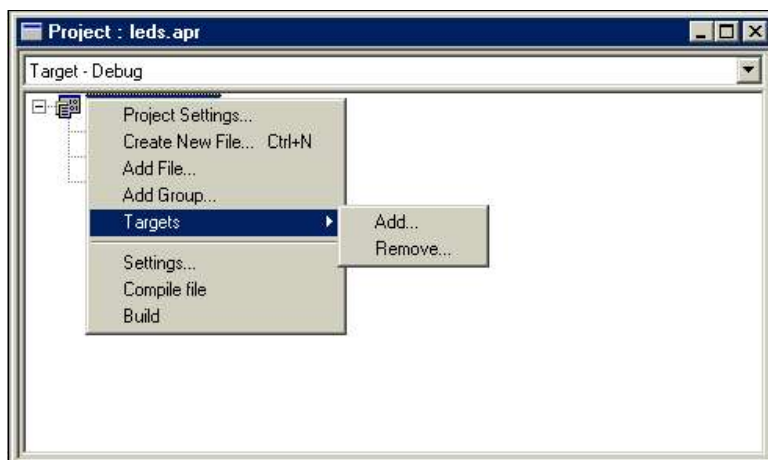


Figura 1: Adicionando um novo *Target*

4 – Na janela *Project*, selecionar *Target-all* (Figura 2). Clicando com o botão direito do mouse em *Target:all* e selecionando *Settings* surgirá uma nova janela, como a mostrada na Figura 3.

5 – A opção "*Run 'compile' on each file in Source Files group*" não deve estar selecionada. A opção "*Run linker/build stage tools*" deve estar selecionada.

6 – No grupo *Run Stage Settings*, a caixa *If output contains the following text* deve conter a mensagem: *Errors: none*.

7 – Selecionar a opção *Don't run the code*.

8 – Substituir o texto *obj* por *cof* na caixa *Extension of object file to load*.

9 – Na caixa *Command line*, escreva:

i) Windows NT, 2000 ou XP: *gcc.bat all* e *gcc.bat coff*

ii) Windows 95,98 ou ME: *make all* e *make coff*

Obs.: Digitar esses comandos em um editor de textos qualquer e colocar na caixa *Command line* com o recurso de copiar e colar. Isso deve ser feito dessa maneira porque a caixa *Command line* não aceita a inserção de novas linhas.

Clique no botão *OK*.



Figura 2: Selecionando um *Target*

10 – Repita os passos 4 a 8 para o *Target-clear*; o passo 9 deve ser executado de forma que a caixa *Command line* contenha as seguintes informações:

i) Windows NT, 2000 ou XP: *gcc.bat clean*.

ii) Windows 95,98 ou ME: *make clean*.

11 – Remover os *targets Debug* e *Release*.

12 – Se o sistema operacional for Windows NT, 2000 ou XP, copiar os arquivos *gcc.bat* e *gcc2.bat* do diretório *<diretório de instalação do WinAVR>\sample* para o diretório do projeto.

13 – Copiar o arquivo *Makefile*, fornecido juntamente com esse documento, para o diretório do seu projeto.

Esse arquivo informa ao compilador as condições para a compilação do código-fonte, tais como o microprocessador que irá rodar o programa, o nome do projeto, o nível de otimização desejado, etc. Para tanto, os seguintes parâmetros básicos devem estar contidos nesse arquivo:

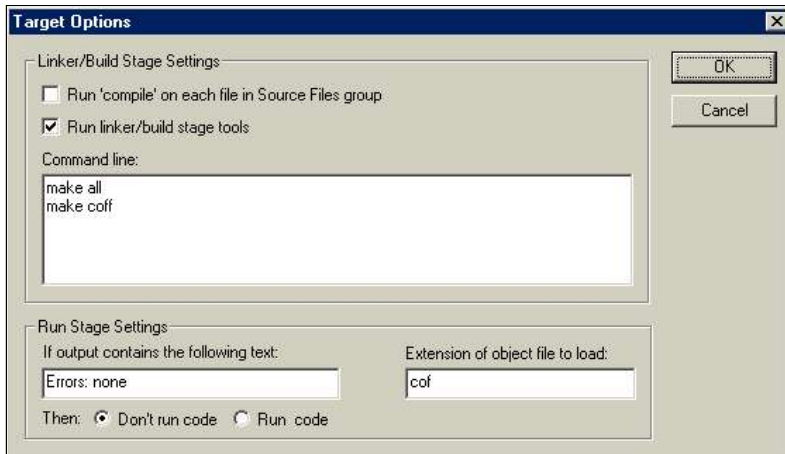


Figura 3: Opções do *Target*

```
PRG      = <nome do projeto>
OBJ      = <nome do projeto>.o
MCU_TARGET = <microcontrolador>
OPTIMIZE = -O2 (pode ser -O0, -O1, -O2, -O3 e -s,
conforme o grau de otimização desejado)
```

Por exemplo:

```
PRG      = leds
OBJ      = leds.o
MCU_TARGET = at90s8515
OPTIMIZE = -O0
```

Edição do código-fonte

Com o projeto devidamente criado, agora resta a edição do código-fonte. Para adicionar um arquivo que contenha o código a ser executado pelo microcontrolador,

clique com o botão direito do mouse em *Target:all* e selecione *Create New File* (Figura 4). Coloque o nome do arquivo, com a extensão *.c*, no campo *Name* e indique como sua localização o diretório do projeto no campo *Location*. Deixe a opção *Add to project* selecionada e clique em *OK*.

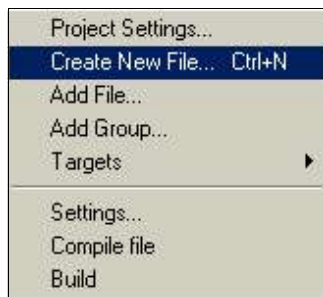


Figura 4: Adicionando o arquivo-fonte

Por exemplo, se *leds.c* for dado como nome do arquivo, o ambiente do *AVR Studio* terá a aparência do mostrado na Figura 5. Nessa figura vemos as janelas de gerenciamento do projeto e a de edição do código-fonte. Esses passos finalizam o processo de criação de um novo projeto. Resta agora colocar a “mão na massa” e começar a programação em si. Mostrarei um exemplo que proporcionará uma melhor compreensão das ferramentas de desenvolvimento em questão.

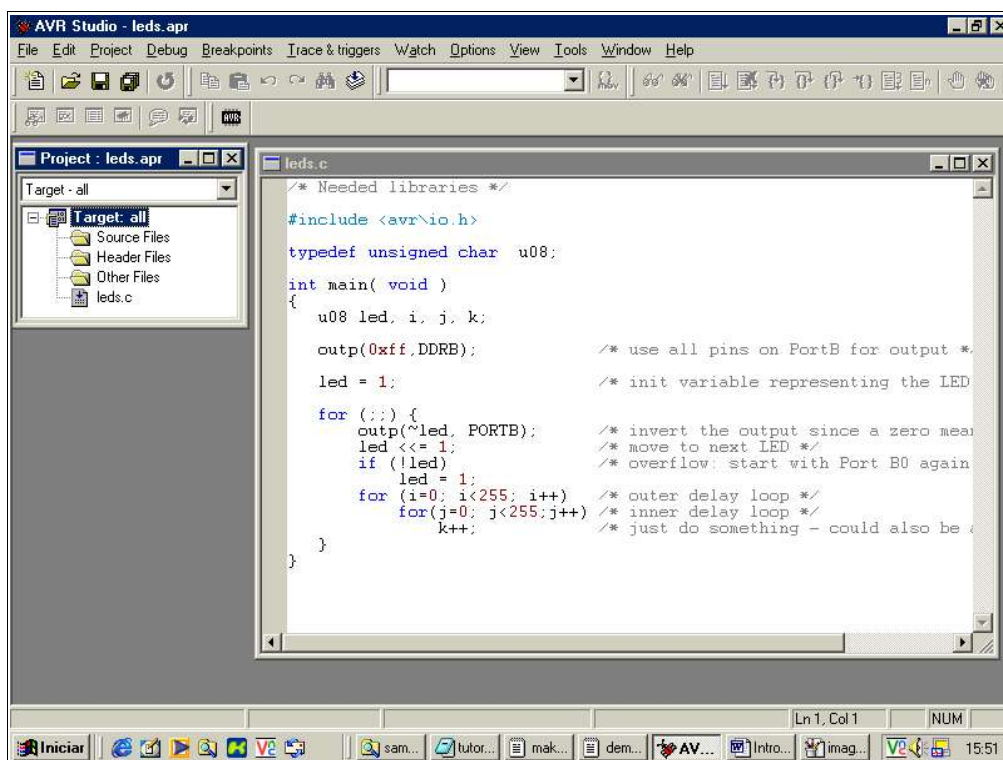


Figura 5: Edição do arquivo-fonte com o AVR Studio

Um pisca-pisca simples

a) Edição e compilação do código

Nesse exemplo, mostrarei como fazer um dos LEDs presentes no kit STK 200 piscar, considerando-se que o microcontrolador instalado no kit é o AT90S8515.

Os LEDs estão ligados à porta B do microcontrolador, conforme o manual do usuário do kit indica (disponível no site www.atmel.com). Com essa informação, é fácil perceber que para que um LED pisque basta alterar o estado de uma das saídas da porta B periodicamente. Um código que realiza essa tarefa é dado a seguir:

```

/* Needed libraries */

#include <avr\io.h>

int main( void )
{
    unsigned int i, j, k;

    struct bit_port_status    /* Bit field */
    {
        unsigned bit_port_0 :1;
    } bit_port;

    bit_port.bit_port_0 = 0;    /* Initialize variable */

    outp(0xff,DDRB);           /* use all pins on PortB for output */

    for (;;)
    {
        bit_port.bit_port_0 = ~bit_port.bit_port_0; /* Toggle the output bit */
        outp(~bit_port.bit_port_0, PORTB);    /* invert the output since a
                                                zero means: LED on */
        for (i=0; i<255; i++) /* outer delay loop */
            for(j=0; j<255;j++) /* inner delay loop */
                k++;           /* just do something - could also be a NOP */
    }
}

```

O início do programa contém a biblioteca necessária ao seu funcionamento (*avr\io.h*). Na realidade, essa biblioteca é básica e necessária a todos os programas, pois ela contém os endereços das portas de E/S, dos registradores, tamanho da memória RAM interna, etc. Segundo o *datasheet* do AT90S8515, para a utilização de uma porta de E/S devemos definir se ela vai ser entrada ou saída. Isso é feito através do registrador DDRx, onde *x* é a porta utilizada. Cada bit da porta pode ser definido como entrada ou saída de forma independente, sendo que com 0 determina-se o bit como entrada e com 1 como saída. Nesse exemplo, a porta é inteiramente definida como saída. O LED escolhido para piscar está ligado à porta PB0 do microcontrolador. Com a intenção de melhorar a clareza do código, escolhi utilizar o recurso de campo de bits que a linguagem C oferece. Esse recurso é extremamente útil, pois simplifica em muito o desenvolvimento de programas que necessitem acessar / verificar bits individuais em variáveis, fato que é praticamente obrigatório em microcontroladores. Sendo assim, dentro do programa está definida uma estrutura (*bit_port*) cujo bit 0 pode ser acessado de forma individual (*bit_port.bit_port_0*). O manual da biblioteca C (presente em <diretório de instalação do WinAVR>\doc\avr-libc\avr-libc-user-manual.pdf) do compilador gcc mostra que a função *main* deve conter um laço infinito. Isso se deve ao fato de que o microcontrolador deve executar um código explicitamente repetitivo. Se esse laço não existir, uma vez que o final do programa for atingido o microcontrolador executará instruções aleatórias, que dependerão do conteúdo da sua memória de programa. Dentro do laço infinito observa-se as instruções de inversão do estado

anterior do LED (aceso ou apagado) e de saída do novo estado. Há também atrasos para que seja possível observar esses estados, pois sem eles o LED piscaria com uma frequência demasiadamente elevada para podermos percebê-la.

Após o código ser devidamente editado, é hora de compilá-lo. Isso é feito selecionando-se *Project*→*Build* (Figura 6) no *AVR Studio* (com *Target:all* selecionado).

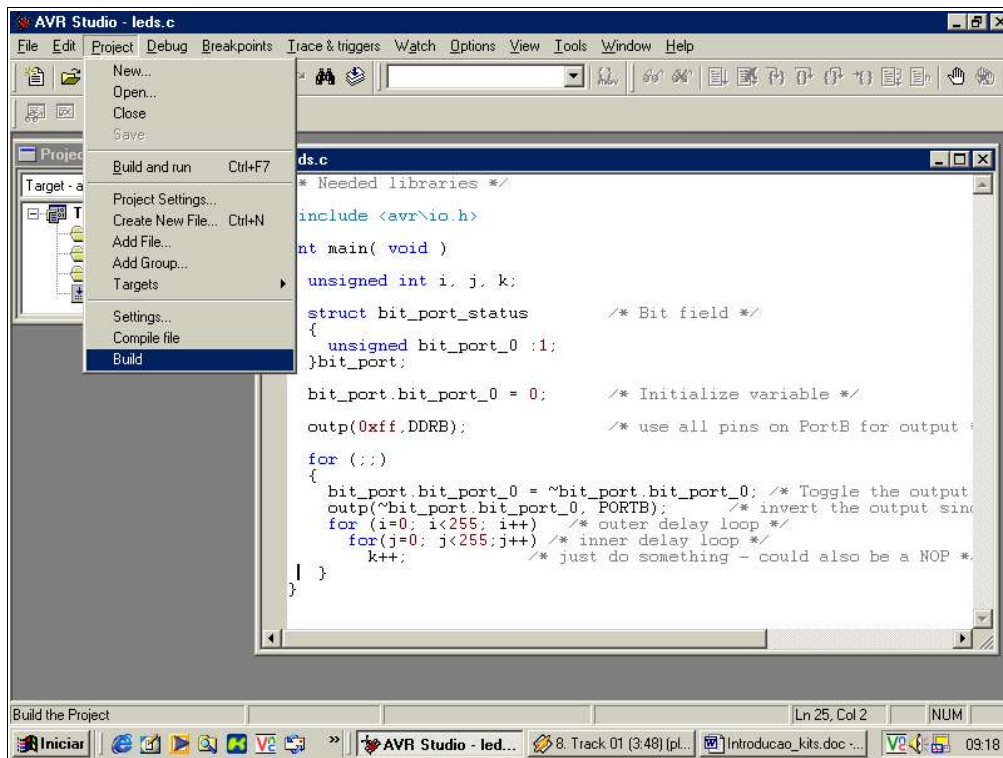


Figura 6: Compilando o programa

Se a digitação foi feita sem erros, uma janela cujo conteúdo será semelhante à da Figura 7 deverá aparecer.

b) Simulação do código

Com o programa compilado pode ser interessante executá-lo no simulador. Isso é interessante para a depuração de erros no código, já que quando ele é carregado no microcontrolador na maioria das vezes só é possível verificar se ele funciona ou não.

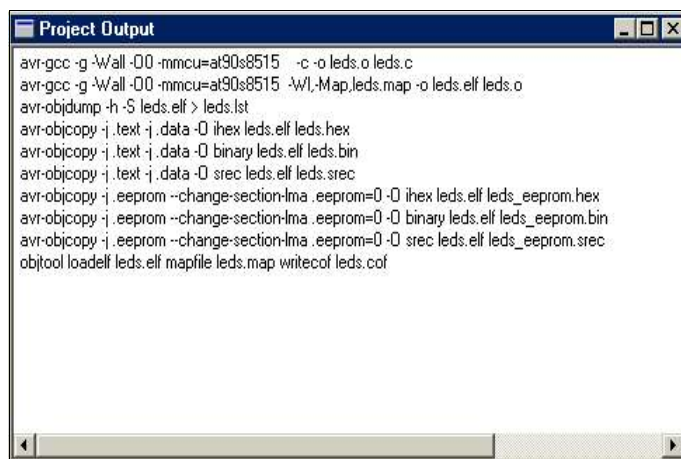


Figura 7: Log da compilação do programa

Para simular a execução do programa, basta selecionar *Project*→*Build and run*. Uma outra janela irá aparecer (Figura 8), para que o dispositivo a ser simulado seja selecionado. Apenas os campos *Device* e *Frequency* precisam ser alterados, para que reflitam as condições reais de funcionamento do programa. Nesse exemplo deve-se selecionar o dispositivo como sendo o AT90S8515, sem RAM

externa (caso a RAM externa estiver presente, a opção *AT90S8515 with External SRAM* pode ser selecionada), e a frequência em 4MHz (pois é o cristal que vem no kit STK 200).

Após pressionar OK, a janela com o código aparece novamente mas com uma seta amarela que indica a próxima instrução que vai ser executada. Para verificar se o microcontrolador está fazendo as tarefas da forma como as imaginamos, o

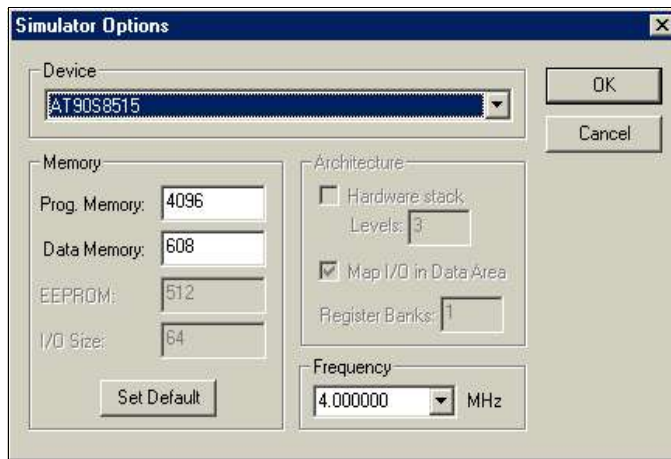


Figura 8: Opções para a simulação

simulador permite que observemos os estados das portas, dos registradores, memória, etc. Então, vamos configurar o ambiente de forma que o estado da porta B seja mostrado. Isso é feito selecionando-se *View→New IO View*. Uma nova janela, chamada *IO: 1(Standard)* é mostrada e, expandindo-se a visão da porta B e organizando as janelas, o ambiente de trabalho terá a aparência dada na Figura 9.

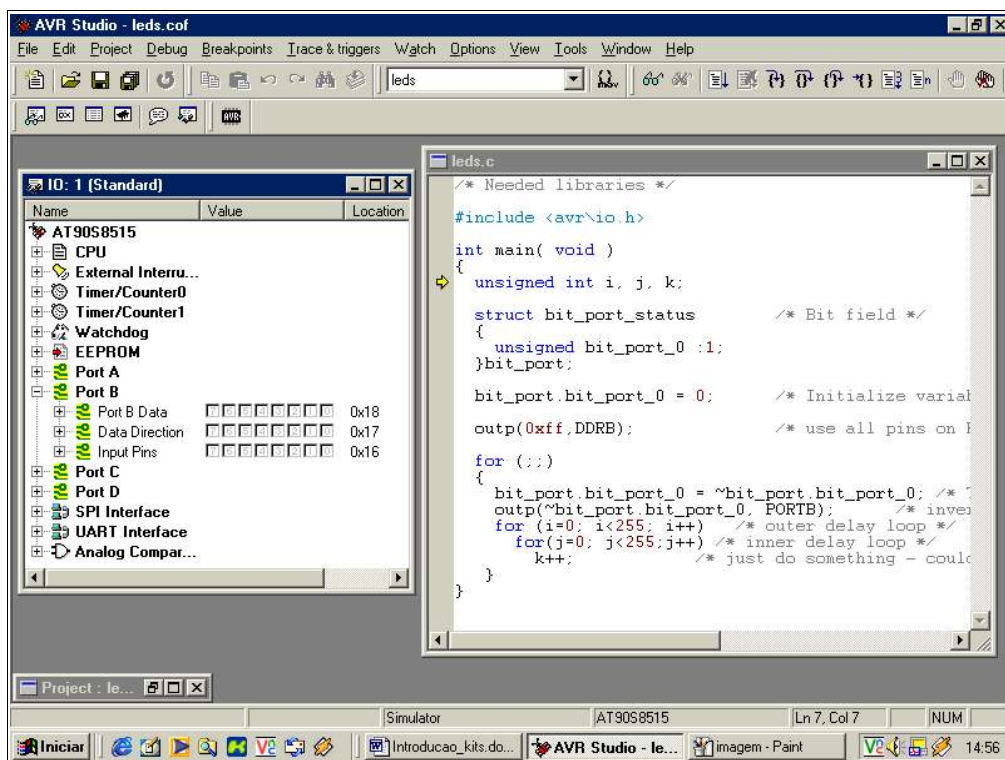


Figura 9: Simulação no *AVR Studio*

Observa-se que a seta amarela aponta para o início do programa. Agora, execute o programa passo-a-passo, pressionando-se a tecla F11 (*Trace into*) até que a seta aponte para a instrução *outp(0xff,DDRB)*. Nesse ponto, convém salientar que todos os bits do registrador DDRB estão com nível lógico 0 (nenhum dos bits do campo *Data Direction* da porta B está checado). Pressionando-se novamente F11, essa condição é modificada e todos esses bits são checados, mostrando que a instrução *outp(0xff,DDRB)*

funciona conforme o esperado. Avançando a execução do programa vemos que, já dentro do laço infinito, o estado do bit 0 da variável *bit_port* é invertido (*toggle*). E, na próxima instrução, o estado desse bit é colocado na porta B. Observe que há uma nova inversão, já que os LEDs do kit STK 200 seguem a lógica invertida (0 significa LED aceso). Ao executar essa instrução, somente a caixa que representa o bit 0 do campo *Port B Data* não é checado e, continuando-se a execução do código, verifica-se que essa caixa vai ter a sua condição invertida. Essa verificação torna-se mais simples com a colocação de um *breakpoint* na linha que tem a instrução *outp(~bit_port.bit_port_0, PORTB)* (posicionando-se o cursor nessa linha e pressionando F9). Ao pressionar F5 o programa passa a ser executado até que esse breakpoint seja atingido. Conclui-se, portanto, que o código passou pelo teste com o simulador.

c) Carregando o programa no microcontrolador

Com a fase de simulação terminada, é hora de passar o programa para o microcontrolador. Para tanto, o programa *PonyProg2000* deve ser aberto.

Caso o programa esteja sendo utilizado pela primeira vez, é necessário que seja feito um procedimento de calibração. Esse procedimento exige que nenhum outro aplicativo esteja sendo utilizado. Clique em *Setup*→*Calibration*. Clique em *Yes* na caixa de diálogo que se abrir e aguarde o término da calibração. Se tudo correr bem, uma outra caixa de diálogo aparecerá informando que o processo foi bem sucedido. Clique em *OK*. Clique agora em *Setup*→*Interface Setup*. Com essa janela informa-se para o

programa em qual interface o *kit* está conectado. Se ele estiver conectado na *LPT1*, selecione *Parallel* e *Avr ISP I/O*, conforme mostra a Figura 10. Clique em *OK*.

Após o processo de calibração, é só carregar o programa no microcontrolador. Clique no botão *Open Program Memory (FLASH) File* e selecione o arquivo *.hex* gerado pelo compilador (no caso do exemplo, *leds.hex*). Clicando em *OK*, o arquivo é carregado e o seu conteúdo é mostrado tanto em hexadecimal como em ASCII, como a Figura 11 mostra.

Agora o dispositivo a ser programado deve ser selecionado.

Clique em *Device*→*AVR micro*→*AT90S8515* e o dispositivo utilizado nesse exemplo será selecionado.

Agora resta somente programar o microcontrolador. Para verificar se o *kit* e as configurações estão corretos clique em *Command*→*Reset (Control+T)*. Se tudo correr bem, aparecerá uma caixa de diálogo informando que o comando foi executado com sucesso e os leds do kit piscarão. Clique em *OK*.

Clique em *Command*→*Erase (Control+E)* para apagar a memória FLASH do microcontrolador. Clique em *Command*→*Write Program (FLASH)* e o programa será carregado no microcontrolador.

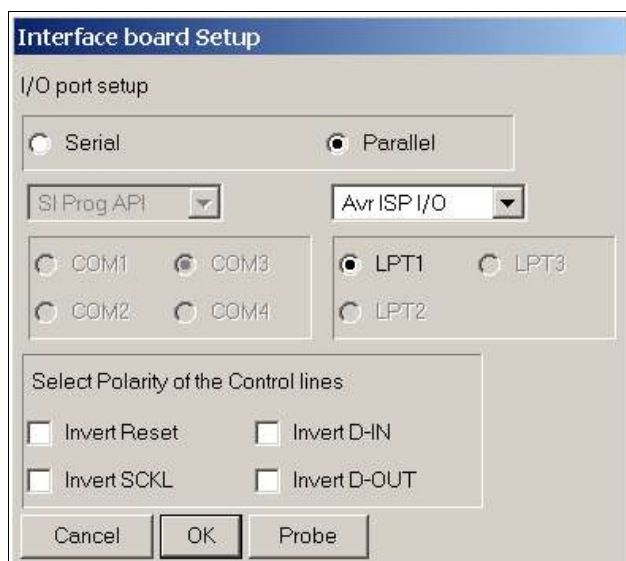


Figura 10: Configurando a interface

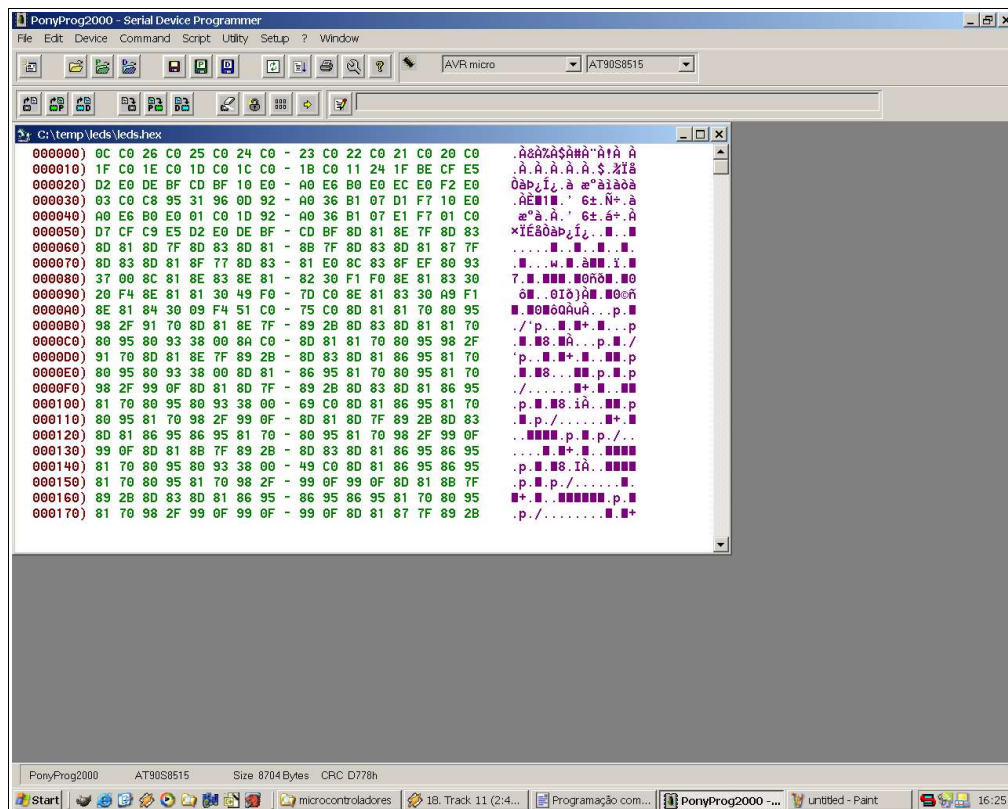


Figura 11: Exemplo carregado no *PonyProg2000*

Basta então verificar o correto funcionamento do dispositivo observando se o LED ligado à porta PB0 pisca periodicamente.