

### Aprendendo a programar com o BASIC Step M8

Chegou a hora de trabalharmos com o conversor analógico/digital e com UART para comunicação serial. Estes dois periféricos são muito interessantes pois permitem uma interação muito grande com o usuário. Se você não leu os artigos anteriores, faça isto agora.

#### Conversor analógico / Digital

O conversor analógico / digital do BASIC Step M8 é um dispositivo muito útil. Ele converte uma tensão analógica em um número digital.

O conversor é de 10 bits. O resultado é um número de 10 bits, então ele não cabe em uma variável byte. Nós devemos usar uma variável inteira (de 2 bytes). Na linguagem BASIC do BASIC Step M8 definimos uma variável inteira colocando o símbolo de porcentagem como por exemplo var%.

As variáveis inteiras não tem sinal, ou seja são sempre positivas. Elas podem variar de 0 a 65535 (&H0 a &HFFFF). Elas podem ser usadas nos comandos IF e em equações exatamente como fizemos com as variáveis byte. A diferença das variáveis byte e inteiras é que as inteiras são armazenadas na RAM estática do chip. Isto significa que são necessários 2 ciclos de clock para ler a variável e mais dois para armazenar o resultado. Sendo assim, as operações com variáveis inteiras são mais lentas do que as com variáveis byte.

Para realizar a conversão analógico / digital e armazenar o resultado em uma variável, nós usamos o comando A2D. A sintaxe do comando é A2D variável, multiplexador, opções.

O BASIC Step M8 possui 8 entradas analógicas que podem ser lidas. A parte “multiplexador” do comando A2D é um número de 0 a 7 que corresponde a entrada analógica que queremos ler. As entradas analógicas estão na porta C do processador.

Existe uma lista grande de opções para o comando A2D. Veja no arquivo de ajuda do compilador sobre todas as opções. Neste exemplo iremos usar a opção IDLE. Esta é uma opção que permite executar uma conversão muito precisa e sem ruídos. A opção IDLE faz com que o processador entre em estado de espera enquanto a conversão é feita e volte a operação assim que a conversão termine. Deste modo não há ruído digital do processador interferindo com o sinal analógico.

Vamos escrever um programa para fazer uma conversão A/D e mostrar o resultado.

Para mostrar o resultado iremos colocar o valor na porta B, que deve ser ligada aos LEDs da placa Super StepLab. O único problema é que temos 8 LEDs e 10 bits para mostrar, pois o conversor é de 10 bits. O meio mais fácil de resolver isto é descartando os dois bits menos significativos. Nós podemos fazer isto com o comando SHIFT.

## Conhecendo o BASIC Step M8 – parte 4

O princípio de operação do comando SHIFT é melhor entendido se olharmos a representação de um número em binário. A representação do número 18 em binário é:

```
00010010
```

Agora deslocando para a direita uma vez temos

```
00001001
```

Deslocando mais uma vez para a direita obteremos

```
00000100
```

Viu como funciona?

Então se deslocarmos o nosso número de 10 bits para a direita duas vezes ele se tornará um número de 8 bits e estará no byte menos significativo da variável inteira. Por exemplo, 0000001100110011 se tornará 0000000011001100. Isto pode ser feito com o comando:

```
SHIFT var% , 2 , RIGHT
```

A última coisa que precisamos fazer é passar o byte menos significativo da variável inteira para uma variável byte (para podermos colocar na porta B). Isto é feito simplesmente atribuindo o valor da variável inteira a uma variável byte. O programa ficará assim.

```
DIRPORT B , OUT  
  
DO  
    A2D x% , 0 , IDLE  
    SHIFT x% , 2 , RIGHT  
    temp = x%  
    OUTPORT B , temp  
LOOP
```

Para executar este programa na Super StepLab, ligue a saída do trimpot a entrada C,0 do BASIC Step M8. Ligue a porta B aos 8 LEDs da placa. Quando executar o programa verá que os LEDs mudam quando você gira o trimpot.

Quando você trabalha com dados do mundo real, é geralmente muito interessante filtrar os dados para remover ruídos indesejados. Para isto existe o comando FILTER.

Vamos modificar o programa acima para filtrar o sinal vindo do trimpot. No próximo programa iremos ler o valor da tensão no trimpot a cada décimo de segundo. Iremos filtrar o resultado com um valor de  $2^4$  (dois elevado a quarta potência) o que nos dará um filtro de 1,6 segundos. Execute o programa a seguir e mude o trimpot rapidamente. Você verá que a mudança nos LEDs se dará bem mais lentamente e irá se aproximando do valor final.

Este é o equivalente por software de usar um resistor e um capacitor para fazer um filtro do

tipo passa baixas. Você pode mudar o parâmetro 4 no comando FILTER para 3 e ver que o resultado fica duas vezes mais rápido.

```
DIRPORT B,OUT

DO
  A2D y%,0, IDLE
  FILTER x%,y%,4
  SHIFT x%,2,RIGHT
  temp = x%
  OUTPORT B,temp
  PAUSE 100
LOOP
```

### Transmissor-Receptor Universal Assíncrono (UART)

Uma das coisas mais úteis que podemos fazer com o BASIC Step M8 é conectá-lo a um computador PC. As informações podem transitar em ambas as direções. A primeira coisa que iremos fazer é transmitir o valor de nosso trimpot (espero que você não tenha desconectado o circuito ainda) para o PC.

A UART utiliza dois pinos específicos do processador, o D,0 para recepção ou RX e o D,1 para transmissão ou TX.

O cabo de gravação do BASIC Step M8 também funciona como cabo de comunicação serial. O compilador possui uma tela de terminal onde os dados recebidos pela porta serial podem ser visualizados. Vamos acrescentar um pouco mais de código ao nosso programa.

Existem dois comandos para controlar a UART. O comando XMIT controla a transmissão da UART e o comando RECV controla a recepção. A princípio nós iremos transmitir os dados lidos do trimpot para o PC com o programa a seguir. O primeiro comando XMIT configura a UART para a velocidade que queremos trabalhar. O segundo comando XMIT transmite os dados. Novamente, você pode ver no arquivo de ajuda mais sobre o comando XMIT e RECV.

```
DIRPORT B,OUT
XMIT INIT 9600

DO
  A2D y%,0, IDLE
  FILTER x%,y%,4
  SHIFT x%,2,RIGHT
  temp = x%
  OUTPORT B,temp
  XMIT OUT temp
  PAUSE 100
LOOP
```

Se olharmos no pino de saída da UART com um osciloscópio, veremos um trem de pulsos a cada décimo de segundo. É claro que é muito mais esclarecedor se pudermos ver os números na tela do PC.

Como foi dito antes, o compilador possui uma tela de terminal que pode ser usada para visualizar os dados recebidos pela porta serial. Abra a tela do terminal e habilite o reset do BASIC Step M8 e você poderá ver os dados relativos a posição do trimpot. Alterando a posição do trimpot você pode ver os valores mudando na tela.

Agora vamos fazer os nossos dados transitarem em ambas as direções. O programa a seguir irá receber um byte do PC, incrementá-lo e enviar de volta para o PC, ou seja o PC só irá receber um byte após enviar um. Se você digitar “A” na tela do terminal, deve receber o caracter “B”.

```
XMIT INIT 9600
RECV INIT 9600
RECV INTERRUPT ON
rflag = 0

DO
    IF rflag | 0 THEN
        INCR rbyte
        XMIT OUT rbyte
        rflag = 0
    END IF
LOOP

'=====
'   ROTINA DE INTERRUPTÃO PARA =
'   RECEBER DADOS DA UART     =
'=====
INTERRUPT RECV
    PUSHFLAGS
    PUSHREG
    rflag = 1
    RECV IN rbyte, errflag~
    POPREG
    POPFLAGS
END INTERRUPT
```

Existem diversos conceitos novos que merecem explicação. O comando RECV INTERRUPT ON habilita a interrupção da parte de recepção da UART. A qualquer momento que um byte seja recebido pela UART, uma interrupção é gerada. Isto significa que não importa o que o programa esteja fazendo, ele irá desviar para a rotina de tratamento de interrupção. No caso do comando RECV, ele irá desviar para a rotina INTERRUPT RECV.

Os dois comandos PUSH salvam o estado do processador na pilha. No fim da rotina de interrupção os dois comandos POP restauram o estado do processador de modo que ele possa continuar a executar do programa principal no mesmo estado que estava antes da interrupção.

Estes comandos devem ser colocados em todas as rotinas de interrupção que você escrever. O processamento real da rotina é feito pelas duas instruções do meio. Ele seta a variável rflag, que informa ao programa principal que um byte foi recebido pela UART. O comando RECV IN coloca o byte recebido pela UART na variável rbyte e também carrega o byte de erro da UART na variável errflag~. Este é um novo tipo de variável. Colocando o sinal de “til” no final do nome da variável, declaramos que a variável é armazenada em RAM e não em registrador. Lembre-se que o processador possui somente 28 variáveis de registrador. Todas as outras deve ser armazenadas em RAM e este é o modo correto de fazer isto.

De volta ao programa principal, nós simplesmente esperamos até que a variável rflag seja diferente de 0. Quando isto acontecer, iremos incrementar a variável rbyte (é o que o comando INCR faz) e transmiti-la de volta para o PC. Se o PC enviar um “A” retornaremos um “B”.

A UART é muito útil quando queremos escrever dados no Display Serial da Tato. A grande vantagem destes displays é que eles necessitam somente de um pino do processador.

Existe um outro comando para enviar dados pela UART que serve perfeitamente para os Displays Seriais, este comando é o PRINT UART.

O comando PRINT UART necessita que a UART seja configurada com o comando XMIT INIT antes de poder ser usado.

Para escrevermos um texto simples na primeira e segunda linha do display poderíamos usar:

```
XMIT INIT 9600    `configura a UART
XMIT OUT &H0D    `Inicializa o display
PRINT UART "linha 1";
XMIT OUT &HC0    `endereça linha 2
PRINT UART "linha 2";
```

Como podemos ver o comando PRINT facilita o nosso trabalho quando queremos enviar uma string para o display.

No próximo artigo iremos falar sobre estruturas de controle.