Rádio para micro controladores

Burkhard Kainka (Germany)

Experiências com módulos SDR de 868 MHz

Existe imenso cabo em torno de todo o mundo, a maior parte amontoado por trás de diversos equipamentos eléctricos/electrónicos. A alternativa a este amontoado de cabos é efectuar a transferência de dados entre equipamentos usando módulos de rádio de baixo custo, que são bastante fáceis de ligar a um microcontrolador. Tentámos implementar esta situação recorrendo a dois microcontroladores ATmega, programados com o Bascom-AVR, processando em simultâneo a transmissão e recepção de dados.

Os módulos de rádio de baixo custo da Hope RF [1] e da Elektor utilizam o transceiver universal IA4420 FSK, na banda ISM, da Integration Associates (agora, Silicon Laboratories, Inc.) [2]. O IA4420 pode ser configurado para utilizar as bandas de 315 MHz, 433 MHz, 868 MHz ou 915 MHz, embora na Europa apenas os 433 MHz e os 868 MHz possam ser utilizados. Para cada uma das diferentes bandas, a Hope RF disponibiliza um módulo de rádio com o respectivo circuito de antena ajustado (**Figura 1**). Para este artigo, seleccionámos o dispositivo de 868 MHz, dado que esta banda encontra-se menos congestionada do que a dos 433 MHz. O módulo pode também ser utilizado na banda dos 433 MHz, mas, uma vez que o circuito de antena não está adaptado para essa frequência, a distância máxima de alcance é drasticamente reduzida.

O módulo tem que ser utilizado em conformidade com os requisitos específicos para dispositivos SRD (Non-specific Range Devices), numa frequência entre os 868,0 MHz e os 868,6 MHz: em particular, o ciclo de trabalho máximo de transmissão permitido que é de 1%. Devido à modulação de banda relativamente larga vamos utilizar o módulo na frequência central da gama permitida (868,3 MHz).

O diagrama de blocos do transceiver (**Figura 2**) mostra os detalhes mais relevantes. O coração do receptor é constituído por um misturador I-Q, semelhante ao utilizado no circuito do famoso Software Defined Radio, publicado na Elektor em Setembro de 2007. O sinal em banda de base é processado através de um amplificador, filtro e desmodulador, para produzir um sinal de saída digital. No transmissor, a PLL (VFO) controla directamente o andar de saída.

O esquema de modulação usado é FSK (Frequency Shift Keying). O desvio de frequência e a largura de banda do receptor são ambas configuráveis: em contraste com os sistemas FM de banda estreita, amplamente usados, é possível obter desvios de frequência de ±15 kHz até ±240 kHz, com correspondentes larguras de banda de recepção até ±400 kHz.

Características do módulo

- Tensão de funcionamento: 2,2 V a 5,4 V.
- Consumo de corrente, em transmissão: 23 mA.
- Consumo de corrente, em recepção: 14 mA.
- Gama de frequência: 860,48 MHz a 879.51 MHz.
- Potência de transmissão: até 4 dBm (aproximadamente 2,5 mW).
- Sensibilidade: -100 dBm (aproximadamente 2 µV).
- Velocidade de transferência de dados: até 115,2 kbaud.
- Desvio de frequência: 15 kHz a 240 kHz.
- Largura de banda de recepção: 67 kHz a 400 kHz.
- FIFO de recepção de 16 bits.
- Dois registos de transmissão de dados de 8 bits.

Inicialização

Como se pode observar na **Figura 3**, os módulos RFM12 são controlados através de um barramento SPI, usando um total de quatro sinais: chip select (NSEL), relógio (SCL), e uma linha de dados para cada sentido (SDI e SDO). O hardware da porta SPI do microcontrolador ATmega pode ser usado, desde que se tenha em consideração que o módulo RFM12 espera ter mensagens com 16 bits. No caso do ATmega32, os pinos PB4 a PB7 são usados para implementar o porto SPI (/SS, MOSI, MISO e SCK). A listagem do programa mostra como é transferida uma palavra de 16 bits. A rotina, que funciona tanto para entrada como para saída, pode ser facilmente alterada para suportar outros microcontroladores AVR, com pinos diferentes para o barramento SPI.

```
Nsel Alias Portb.4
Sdi Alias Portb.5
Sdo Alias Portb.6
Sck Alias Portb.7

Function Spi16(byval Dout As Word) As Word
Local Nspi As Integer
Local Dspi As Integer
Local Dsdo As Word
  Nsel = 0
  Dsdo = 0
  For Nspi = 1 To 16
    Dspi = Dout And &H8000
    If Dspi = 0 Then
       Sdi = 0
    Else
       Sdi = 1
    End If
    Dout = Dout * 2
    Dsdo = Dsdo * 2
    Dsdo = Dsdo + Sdo
    Sck = 1
    Waitus 5
    Sck = 0
  Next Nspi
  Nsel = 1
  Spi16 = Dsdo
End Function
```

Assim que a alimentação é aplicada ao módulo, este tem que ser inicializado. Existe um número tão grande de bits de configuração que não é muito fácil configurá-los todos bem à primeira. A folha de características [2] lista todas as configurações existentes. A lista abaixo mostra os valores recomendados para funcionar a uma frequência de 868,3 MHz, com um desvio de frequência de ±90 kHz e uma taxa de transferência de dados de 2,4 kbaud. A subrotina Freq_rfm12 também é necessária como parte do processo

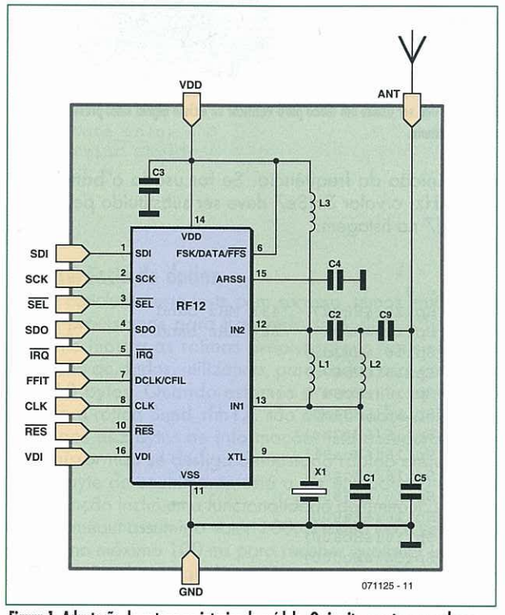


Figura 1. Adaptação da antena no interior do módulo. O circuito exacto e os valores dos componentes usados dependem da gama de frequências para a qual o módulo é projectado.

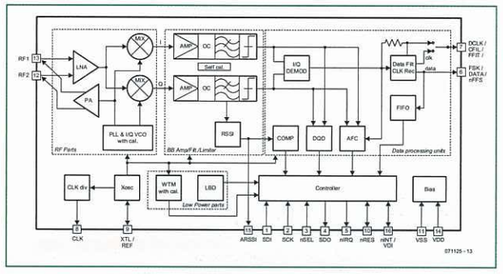


Figura 2. Diagrama de blocos do transceiver IA4420.

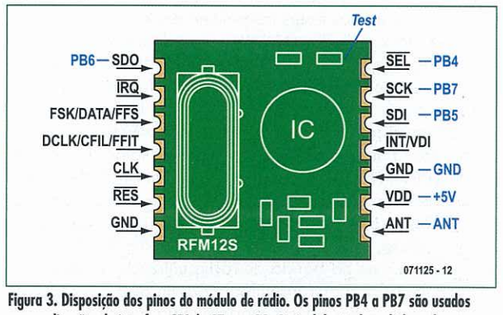


Figura 3. Disposição dos pinos do módulo de rádio. Os pinos PB4 a PB7 são usados para as ligações do interface SPI do ATmega32. O nível de sinal recebido pode ser monitorizado no ponto de teste.

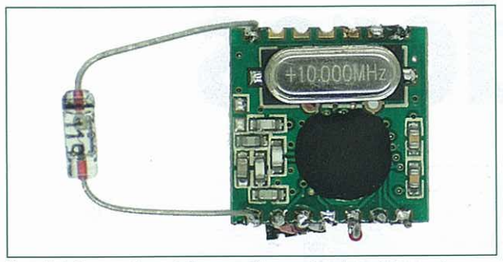


Figura 4. Pode ser usado um díodo para verificar se existe algum sinal presente na saída da antena.

de definição da frequência. Se for usada a banda dos 433 MHz, o valor &H8e7 deve ser substituído pelo valor &H80d7 na listagem.

```
  Nsel = 1
  Sck = 0
 'D = Spi16(&H80d7)  '433 MHz band
  D = Spi16(&H80e7)  '868 MHz band
  D = Spi16(&H82d9)
  D = Spi16(&Ha67c)
  D = Spi16(&Hc647)
  D = Spi16(&H94a0)
  D = Spi16(&Hc2ac)
  D = Spi16(&Hca81)
  D = Spi16(&Hc483)
  D = Spi16(&H9854)
  D = Spi16(&He000)
  D = Spi16(&Hc800)
  D = Spi16(&Hc000)

  Freq = 868.300
  Freq_rfm12
   ...

Sub Freq_rfm12
  If Freq < 800 Then Freq = Freq * 2
  Freq = Freq - 860
  D = Freq / 0.0050
  If D < 96 Then D = 96
  If D > 3903 Then D = 3903
  D = D + &HA000
  D = Spi16(d)
End Sub
```

Transmissão

São tantos os possíveis erros na utilização dos módulos que o melhor é começar com alguns testes iniciais mais simples. O programa RFM12.bas possui uma série de rotinas de teste que podem ser executadas recorrendo à instrução GOTO. Os nossos primeiros testes transmitem dados de uma forma contínua, pelo que não estão em concordância com as regulações de utilização da banda dos 868 MHz. Portanto, os testes devem ser realizados sem a antena ligada. A listagem abaixo mostra como efectuar o teste do transmissor sem aplicar um sinal modulador. O comando &H8238 liga o transmissor. Para ver se há algum sinal na saída da antena, pode ligar um díodo de germânio (ou um díodo Schottky) em paralelo para a massa (**Figura 4**). Deve aparecer uma tensão de aproximadamente 1 V aos terminais do díodo. Se, por acaso, tiver a sorte de ter um analisador de espectro, que funcione na banda de rádio utilizada, pode usá-lo para procurar o sinal transmitido: não em 868,3 MHz, mas em redor dos 868,21 MHz, dado que na ausência do sinal modulador a portadora é deslocada para este valor.

```
'start transmitter, no data
Test1:
  D = Spi16(&H8238)
  Do
  Loop
```

O nosso segundo teste produz um sinal de dados modulado (ver abaixo). O comando usado para isto é &Hb8xx, onde 'xx' corresponde ao byte de modulação. No nosso exemplo usámos o comando &Haa, que em binário corresponde a 10101010. Antes de cada byte ser transmitido, deve ser executada a função Wait_rfm12 para esperar até que o módulo esteja pronto: isto implica fazer com que o sinal NSEL vá para o nível lógico baixo e esperar que o sinal SDO venha para o nível lógico alto. Isto funciona quer o módulo esteja a funcionar como transmissor ou como receptor.

```
Sub Wait_rfm12
  Nsel = 0
  Do
  Loop Until Sdo = 1
End Sub

  'transmit data
Test2:
  D = Spi16(&H8238)
  Do
    Wait_rfm12
    D = Spi16(&Hb8aa)
  Loop
```

Agora, o analisador de espectro deve ser capaz de sintonizar portadoras tanto nos 868,21 MHz como nos 868,39 MHz, uma vez que o transmissor está continuamente a comutar entre estas duas frequências.

Recepção

Para receber dados é preciso um segundo sistema, constituído por um microcontrolador ATmega32 e um módulo RFM12. O nosso próximo teste demonstra como se pode receber 100 bytes de dados em modo de recepção contínua, com todos os bytes recebidos a serem transferidos, na íntegra, sobre o interface RS232.

```
'start receiver, all data
Test4:
  D = Spi16(&H82c8)
  D = Spi16(&Hca87)
  For N = 1 To 100
    Wait_rfm12
    D = Spi16(&Hb000)
    Data_in(n) = D
    Print Chr(d);
  Next N
  Do
  Loop
```

O receptor gera dados de saída mesmo quando não há sinal a ser transmitido na frequência utilizada. Neste caso, o teste anterior vai converte o receptor e o ruído da antena numa sequência de números aleatórios.
Quando o transmissor é ligado observa-se um efeito muito interessante. Por exemplo, o segundo teste acima envia sempre o byte &Haa. Como é que o receptor reage a este sinal? O fluxo de bytes na saída altera-se de imediato para um padrão regular, mas não necessariamente o correcto. Numa análise mais detalhada, verifica-se que a sequência de bits está correcta, mas o receptor não sabe identificar onde começa e termina cada byte.
Também é possível verificar o nível do sinal recebido. O IA4220 tem um pino ligado ao seu circuito de AGC (ARSSI, indicado como ponto de teste na **Figura 3**), que infelizmente não está disponível como pino de saída no módulo. No entanto, o ponto de teste é muito fácil de identificar e localizar na placa de circuito impresso, junto ao condensador no canto da placa. O nível de sinal DC, neste ponto, reflecte o nível do sinal recebido. Sem qualquer sinal, o nível de sinal DC situa-se tipicamente entre os 0,3 V e os 0,5 V, e com um forte sinal de entrada este nível de tensão DC pode chegar a atingir níveis acima de 1 V. Em condições de funcionamento normais, é sempre possível verificar o nível desta tensão DC, de modo a verificar se os dados enviados pelo transmissor estão a ser recebidos com nível de sinal suficiente.

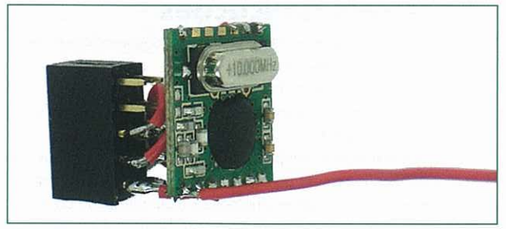
Sincronização

Em teoria, poderíamos retirar a sequência de bits (sem sincronismo) do receptor e extrair os dados válidos com alguma ginástica de software. No entanto, os projectistas do IA4420 incluíram uma funcionalidade no chip que evita isso: a sequência de bits recebidos alimenta de forma contínua um registo de deslocamento de 16 bits, e consoante cada bit é recebido o conteúdo do registo é comparado com um determinado padrão fixo. O número mágico em questão é 2DD4 em hexadecimal (atenção: não confundir com R2-D2!). O transmissor tem então de enviar na sequência de dados estes dois bytes, primeiro o valor hexadecimal 2D e em seguida o valor hexadecimal D4. Nessa altura, o receptor sabe então que o próximo bit corresponde ao primeiro bit do primeiro byte da mensagem propriamente dita. Na prática, primeiro, o transmissor envia uma sequência alternada de uns e zeros, por exemplo, três bytes com o valor hexadecimal AA. Isto permite ao receptor sincronizar a recepção do fluxo de dados e ajustar o seu nível de controlo automático. Depois deste padrão vêm os bytes mágicos e, em seguida, a mensagem propriamente dita. Neste teste, os dois bytes (2D e D4) são enviadas alternadamente.

```
'transmit key data
Test3:
  D = Spi16(&H8238)
  Do
    Wait_rfm12
    D = Spi16(&Hb82d)
    Wait_rfm12
    D = Spi16(&Hb8d4)
  Loop
```

Agora vamos efectuar o teste seguinte ao receptor, que difere do teste anterior apenas no segundo comando de inicialização. Uma vez mais, deverão ser recebidos 100 bytes. No entanto, neste caso, o programa vai ficar à espera na rotina Wait_rfm12 até que o transmissor seja ligado. A linha SDO no receptor só vai passar para o nível lógico alto quando os bytes 2D e D4 forem recebidos, e os bytes seguintes sejam válidos, com o transmissor e o receptor a acordarem quanto às posições e limites do byte.

```
'start receiver, matched data
Test5:
    D = Spi16(&H82c8)
    D = Spi16(&Hca83)
    For N = 1 To 100
      Wait_rfm12
      D = Spi16(&Hb000)
```



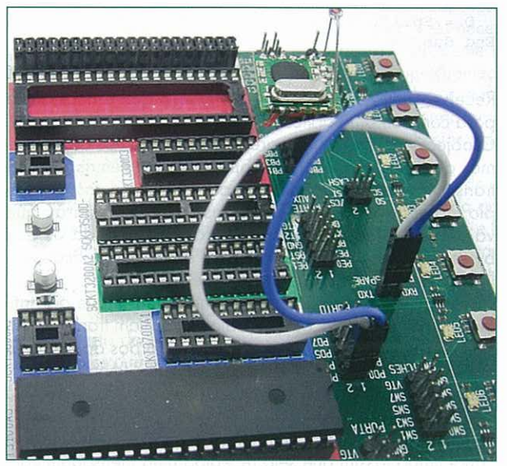
Neste caso, o autor soldou o módulo a um conector...

```
    Data_in(n) = D
    Print Chr(d);
  Next N
Do
Loop
```

Transferência de dados

Tendo concluído este teste com sucesso, temos então tudo o que precisamos para transferir os dados. A listagem seguinte mostra as rotinas próprias para transmissão e recepção de dados, utilizando, para cada caso, um buffer com 10 bytes. Quando estamos a transmitir os dados, usando a rotina Send_rfm12, são adicionados dois bytes extra aos dez bytes de informação: isto assegura que o transmissor não se desliga demasiado rápido enquanto o último byte de enchimento está a ser enviado. O código de recepção inclui uma funcionalidade de timeout: se a variável Timeout assumir o valor 100, a rotina Receive_rfm12 espera no máximo 100 ms para receber qualquer informação. Se não chegar qualquer informação válida, o buffer de recepção permanece inalterado.

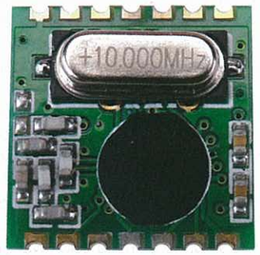
```
Sub Send_rfm12
  D = Spi16(&H8238)
  Wait_rfm12
  D = Spi16(&Hb8aa)
  Wait_rfm12
  D = Spi16(&Hb8aa)
  Wait_rfm12
  D = Spi16(&Hb8aa)
  Wait_rfm12
```



... para permitir realizar experiências com o kit de desenvolvimento STK 500 AVR Starter Kit.

Possíveis aplicações

- Dispositivos comutados como por exemplo lâmpadas e rádios.
- Controlo remoto para robôs.
- Sistemas de abertura para garagens.
- Sistema geral de interruptores para quarto de crianças, para assegurar que tudo fica desligado à noite.
- Sistemas de alarme.
- Envio de mensagens simples (como, por exemplo, o jantar está pronto).
- Monitorização remota do estado do sistema de aquecimento.
- Monitorização de valores analógicos (com alteração do software).
- Sistema de monitorização remoto para baterias e carregadores.
- Estações meteorológicas.




```
  D = Spi16(&Hb82d)
  Wait_rfm12
  D = Spi16(&Hb8d4)
  For N = 1 To 10
    Wait_rfm12
    D = &HB800 + Data_out(n)
    D = Spi16(d)
  Next N
  Wait_rfm12
  D = Spi16(&Hb8aa)
  Wait_rfm12
  D = Spi16(&Hb8aa)
  Wait_rfm12
  D = Spi16(&H8208)
End Sub


Sub Receive_rfm12
  Tt = Timeout * 10
  D = Spi16(&H82c8)
  D = Spi16(&Hca83)
  For N = 1 To 10
    Nsel = 0
    T = 0
    Do
      T = T + 1
      Waitus 100
      If T > Tt Then Goto Nosignal
    Loop Until Sdo = 1
    D = Spi16(&Hb000)
    Data_in(n) = D
  Next N
Nosignal:
  D = Spi16(&H8208)
End Sub
```

Receive_rfm12 e Send_rfm12 é tudo o que é necessário para construir uma aplicação para enviar dados via rádio. O objectivo é que o firmware seja o mesmo nos dois extremos da ligação rádio, para que os dois sistemas possam transmitir e receber dados alternadamente. No nosso exemplo, colocámos no buffer de transmissão um padrão que vai incrementando. O receptor coloca na saída os dez bytes no interface RS232, pelo que é simples verificar se a ligação está a trabalhar correctamente.
Contudo, temos ainda um problema: como é que garantimos que as duas estações de rádio não tentam transmitir em simultâneo, fazendo com que a recepção dos dados falhe numa e noutra estação? Esta questão é facilmente solucionada: alteramos o valor de timeout aleatoriamente entre 400 ms e 1400 ms. Depois de possivelmente ocorrerem algumas tentativas de transmissão, a estação rádio vai encontrar o canal limpo; assim que seja recebida uma mensagem com sucesso as duas estações de rádio podem então começar a transmitir alternadamente, uma de cada vez.

```
Do
    For N = 1 To 10
      Data_out(n) = N
    Next N
    Send_rfm12
    Waitms 500
    For N = 1 To 10
       Data_in(n) = 0
    Next N
    Timeout = 400 + Rnd(1000)
    Receive_rfm12
    For N = 1 To 10
       Print Data_in(n);
       Print " ";
    Next N
    Waitms 700
 Loop
```

Uma aplicação prática deste sistema pode ser ler um byte de um dos portos de um microcontrolador e transmitir esse byte para o outro lado, que pode depois enviar o byte recebido para um dos seus portos de saída. Em teoria, isto permite controlar remotamente até oito dispositivos. Uma das formas de aumentar a fiabilidade da ligação seria enviar o byte em primeiro lugar com o seu valor real e depois invertido: isto permitiria detectar erros de transmissão que ocorressem apenas num único bit. Também é possível ligar a saída do microcontrolador do receptor de volta para um porto de entrada do mesmo microcontrolador, que pode depois transmitir de volta uma confirmação dos dados recebidos para a estação emissora.

O módulo de rádio pode ser obtido através do Serviço Elektor (**Refª 071125-71**). O programa em BASCOM pode também ser obtido através do site da Elektor, na página dedicada a este artigo.

(071125-1)

Artigo original: Radio for Microcontrollers – January 2009

Internet

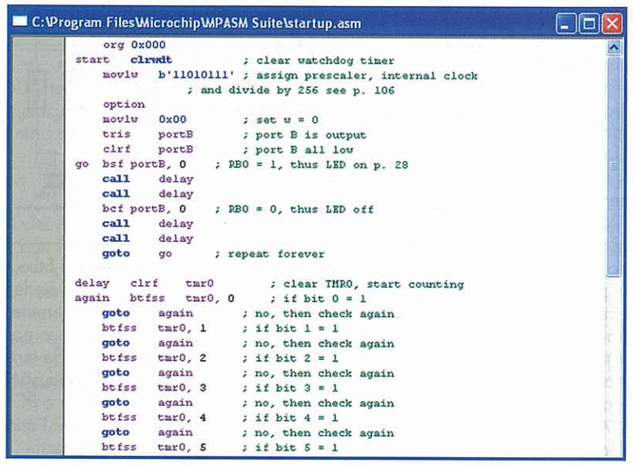
[1] http://www.hoperf.com

[2] http://www.silabs.com/Support%20Documents/TechnicalDocs/IA4420-DS%20v1.7r.pdf

Quando o meu microcontrolador não arranca...

É impossível imaginar a electrónica actual sem microcontroladores. Tudo aquilo que antigamente era feito com lógica combinatória ou sequencial é hoje realizado com um único circuito integrado, que executa um programa, que faz exactamente aquilo que se pretende. Ou melhor, que devia fazer exactamente aquilo que se pretende...

Luc Lemmens



Especialmente durante as primeiras fases de desenvolvimento do hardware e software, é frequente depararmo-nos com problemas enigmáticos. O mais frustrante de todos é quando o microcontrolador não faz absolutamente nada (ou melhor, parece não fazer absolutamente nada). Nos microcontroladores mais antigos, em que a memória era externa, era possível usar um osciloscópio ou ponta de prova lógica para obter algumas respostas. Assim, observando o barramento de dados ou de endereços era possível perceber se o microcontrolador se encontrava a funcionar. Mas hoje em dia, na maioria dos casos, toda a memória está integrada no próprio microcontrolador, pelo que apenas nos portos de E/S se pode tentar encontrar alguns sinais de vida.

Felizmente, a maioria dos controladores utilizam ainda um oscilador externo, pelo que é fácil verificar se pelo menos o sinal de relógio está presente. Contudo, alguns microcontroladores mais pequenos dispõem de um oscilador interno, pelo que esta opção de teste também não está disponível.

O método mais simples para garantir que o oscilador está activo e que o software, em princípio, está pronto a funcionar, é incluir uma pequena rotina no início do programa que faz piscar uma das saídas. Preferencialmente um pino de saída onde esteja ligado um LED, o que dá um retorno visual imediato, não sendo necessário um osciloscópio. Não faça o programa esperar por qualquer acção externa, como por exemplo um botão de pressão ou a recepção de dados pela porta série, mas simplesmente faça qualquer coisa num pino qualquer. Se ainda assim não houver qualquer indicação de actividade (e assumindo que as verificações mais básicas como a presença da tensão de alimentação e do nível correcto no pino de reset já foram feitas), há uma hipótese de dez para um de que algo possa ter corrido mal durante o processo de programação. Os bits de configuração do microcontrolador são os principais suspeitos.

Os microcontroladores modernos têm um conjunto de bits (fusíveis) que definem o comportamento do microcontrolador. Quando um microcontrolador não arranca, os primeiros parâmetros a conferir na configuração são os referentes ao oscilador (interno ou externo, gama de frequências). Se estes estiverem incorrectos, na maioria dos casos o microcontrolador não arranca. O suspeito seguinte é, caso esta funcionalidade exista no controlador em questão, o temporizador watchdog (WDT). Se este tiver sido activado na programação e a sua aplicação não fizer um reset ao temporizador, o microcontrolador vai estar continuamente a fazer reset, e em termos práticos nunca vai chegar a executar o seu programa (ou não vai passar da fase inicial).

O terceiro parâmetro da configuração a considerar é o associado ao circuito de reset. Em alguns microcontroladores o sinal de reset pode ser configurado como externo ou interno. Consulte a folha de características do microcontrolador e verifique o resto do circuito, para confirmar se o nível de tensão e os tempos de subida (ou descida) são os correctos. Ás vezes é preciso perder algum tempo a investigar o software de programação para descobrir onde estão escondidas as configurações, na medida em que por vezes estas não têm os nomes que seria de esperar. Em caso de desespero, pode sempre tentar todas as configurações possíveis, na esperança que uma delas funcione. Isto parece algo absurdo, mas infelizmente pode vir a ser necessário noutra altura. Felizmente que hoje em dia a (re)programação de um microcontrolador já não é a tarefa morosa, sendo actualmente um processo bastante expedito.

Uma indicação de arranque simples no início da aplicação poupa imenso tempo e frustração, na medida em que sabe à partida que o hardware está a funcionar e nada correu mal na programação dos bits de configuração. Numa fase posterior do desenvolvimento pode retirar a rotina de arranque, mas apenas se for estritamente necessário.

(070101-1)

Artigo original: My microcontroller doesn't go... - April 2008