Microaltímetro

Escrito por: Jamille Rocha Alves Rodrigues
Colaboradores: Thiago Barros e Yan Coutinho

Introdução

O objetivo desse projeto é registrar dados da trajetória dos minifoguetes, mais especificamente os dados de altura. Devido ao diâmetro do foguete, que pode variar entre e 28mm, projetamos um altímetro com as menores dimensões possíveis para atender as dimensões do minifoguete. A partir das dimensões requisitadas optamos por um projeto com base em dois dispositivos: o ATtiny85 e o BMP180. O BMP180 é um sensor de temperatura e pressão que será especificado logo a seguir, basicamente através dele poderemos obter valores de altura. Para comandar o BMP180 usaremos o microcontrolador ATtiny85, no qual também registramos os dados do vôo. ATtiny85 e BMP180 estarão soldados em lados opostos de uma placa de circuito impresso de formato circular (com diâmetro interno igual ao do minifoguete) que poderá ser prendida a coifa ou ser colocada acima do motor.

Componentes

Arduino

O arduino é uma placa que permite a automação de projetos eletrônicos e robóticos por profissionais e amadores. É uma plataforma de prototipagem eletrônica de hardware livre e de placa única, projetada com um microcontrolador Atmel AVR com suporte de entrada/saída embutido, uma linguagem de programação padrão, a qual tem origem em Wiring, e é essencialmente C/C++. O microcontrolador ATmega328 da Atmel é utilizado nos Arduinos mais recentes. É um microcontrolador de 8 bits, com arquitetura Harvard modificada. O ATmega328 pertence à família AVR da Atmel. Todos os modelos desta família compartilham uma arquitetura e conjunto de instruções básicas, particularmente os grupos tinyAVR (microcontroladores ATtiny), megAVR (os ATmega) e XMEGA (os Atxmega).Os primeiros modelos de Arduino usavam o ATmega8 (com 8K de memória Flash), que posteriormente foi substituído pelo ATMega168 (com 16K de Flash e maiores recursos de entrada e saída) e finalmente pelo ATMega328 (com 32K de Flash).

A figura abaixo, extraída do datasheet, mostra os principais blocos do microcontrolador ATMega328:

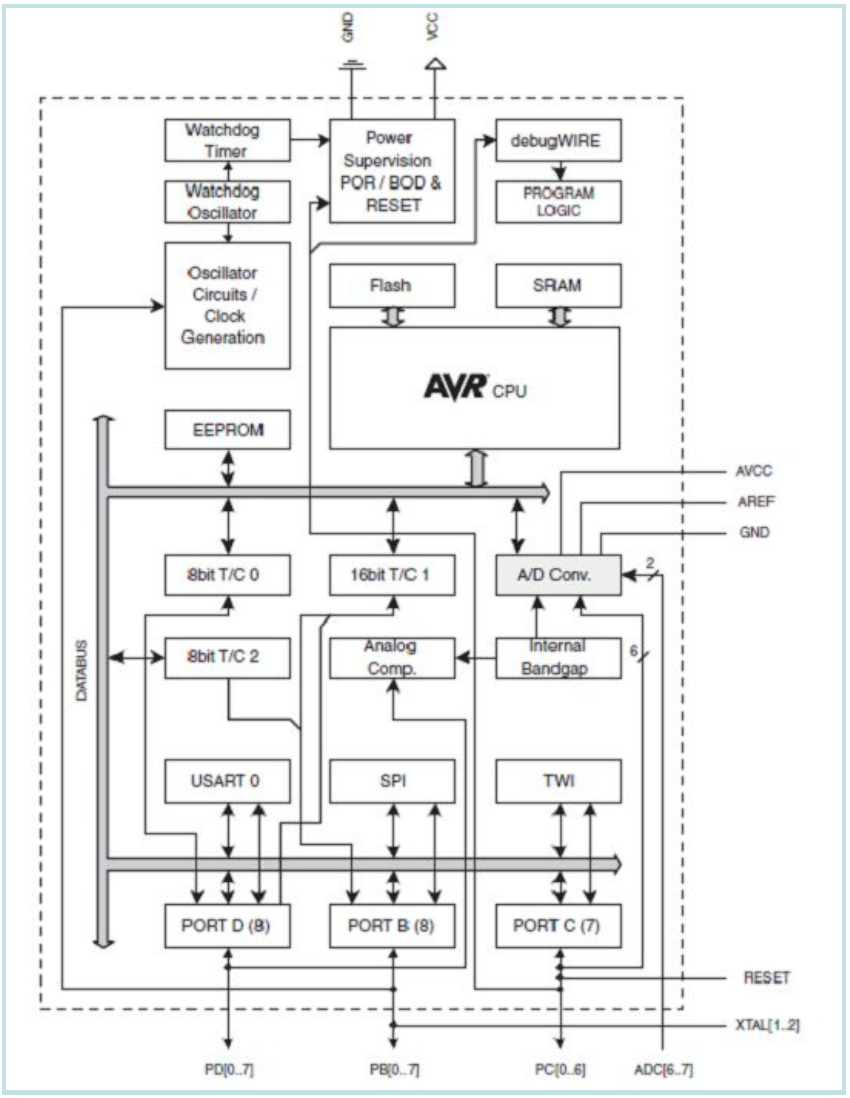

Figura 1 - Blocos do microcontrolador Atmega328

Reparar as ligações separadas entre a CPU e as memórias Flash e SRam. O uso de vias de dados separadas para programa e dados é uma característica da arquitetura Harvard. Na família AVR as duas vias tem largura de 8 bits e a memória Flash pode ser usada para armazenar dados constantes (daí ser uma arquitetura Harvard modificada). Entretanto, somente instruções armazenadas na Flash podem ser executadas (não é possível executar código que esteja na SRam).

Como outros microcontroladores AVR, o ATMega328 possui ainda uma memória do tipo EEProm, porém esta memória está ligada na via de conexão aos periféricos e portanto não é acessada pelas instruções normais de acesso a memória.

Além da EEProm podem ser vistos as três portas de E/S digital (PORTs B, C e D), os três timers (TCx, dois de 8 bits e um de 16 bits), o conversor A/D, o comparador analógico e as interfaces seriais SPI, TWI (compatível com I2C) e USART.

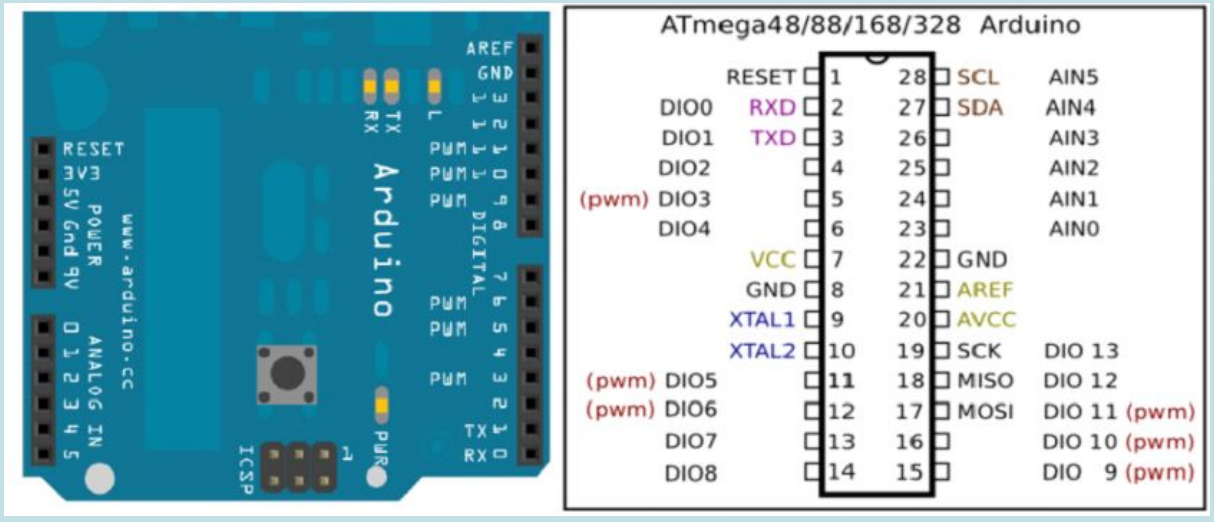

Figura 2 - Imagem do arduino e figura da pinagem do atmega328

Dos 28 pinos do ATmega328, 23 podem ser usados como entrada ou saída digital (inclusive os que estão marcados como entradas analógicas na placa do Arduino). Todos os pinos possuem funções alternativas, por exemplo no Arduino utiliza funções como Reset, XTAL1 e XTAL2, limitando o uso destes pinos para E/S digital. Os pinos de E/S digital estão organizados em três portas (PB, PC e PD), mas cada pino pode ser configurado independentemente como entrada e saída. Todos os pinos possuem um resistor de pull-up (também controlado independentemente) e diodos de proteção. Quando operando como saída, os pinos podem tanto gerar como receber uma corrente suficiente para acender um LED (até 40mA por pino). Os 23 pinos de E/S digital podem também ser gerar uma interrupção quando ocorre mudança de sinal. É possível controlar a geração de interrupção pino a pino e ela independe do pino ter sido configurado como entrada ou saída. O conversor analógico digital (ADC) possui 10 bits de resolução e possui 8 opções de entrada (6 no encapsulamento DIP normalmente usado no Arduino). Existem três opções de referência: a tensão de alimentação (fornecida em um pino separado, AVcc), uma referência interna de 1,1V ou um tensão externa (fornecida no pino ARef).

O arduino necessita ter um software chamado bootloader gravado no microcontrolador e o código executável (programa) gravado na flash.. Vamos agora juntar estas duas informações e ver em mais detalhes o que é e como funciona o Bootloader. Uma vez que todo software a ser executado no AVR precisa estar na Flash, o programa feito na IDE do Arduino (sketch) precisa ser gravado lá. Para dispensar o uso de um gravador externo, a gravação da Flash é feita por um software que reside nela própria, o Bootloader. O Bootloader é o primeiro software executado pelo microcontrolador após um Reset (Boot) e carrega na Flash um software que recebe pela serial (loader).

O ATmega328 possui alguns recursos que facilitam isto:

 A memória Flash pode ser dividida em duas seções, um bootloader (no final da memória, com 512, 1024, 2048 ou 4096 bytes) e uma aplicação (com o restante). Na seção do bootloader pode ser usada a instrução SPM, capaz de reprogramar toda a Flash.

 Independente disto, existe uma outra divisão em duas seções de tamanho fixo, a RWW (Read While Write, os primeiros 28K) e a NRWW (No Read While Write, os últimos 4K). Enquanto está sendo feito um apagamento ou escrita na região RWW, código pode estar

executando na região NRWW. No caso contrário, região NRWW sendo atualizada, o processador fica parado durante a operação.
 Para fins de apagamento e gravação a Flash é dividida em páginas de 128 bytes. As operações afetam sempre uma página inteira.

Em termos práticos, um software na seção bootloader pode executar sem parada enquanto a seção de aplicação é atualizada. A seção de bootloader pode se auto-atualizar, porém o processador ficará parado enquanto a operação de completa (e você não vai querer atualizar a página onde está sendo executada a instrução SPM).

No que diz respeito ao protocolo para receber o programa pela serial, o Bootloader utiliza um subconjunto do protocolo usado por um programador externo, o STK-500. Isto permite usar softwares padrão de programação no PC, a IDE do Arduino usa o Avrdude.

Ao longo dos anos o bootloader do Arduino passou por alterações, a versão atual é o chamado Optiboot (bootloader "otimizado"). A otimização consistiu em reduzir o tamanho para caber em 512 bytes, aumentar a velocidade na serial para 19200 e receber os dados de uma página enquanto a apaga.

Existe ainda um último ponto importante no bootloader: o seu disparo e o disparo da aplicação. Nas primeiras versões, o bootloader começava a sua execução quando era feito um reset e esperava por um certo tempo uma comunicação antes de iniciar a aplicação. Isto trazia dois inconvenientes: era preciso apertar o botão de reset antes de enviar um sketch e aguardar alguns segundos ao final da carga para ele começar a ser executado.

A primeira limitação foi contornada por hardware: ao detectar uma conexão na serial o microcontrolador é ressetado no Duemilinove e no Uno. Para o segundo inconveniente, o bootloader passou a distinguir a iniciação pelo sinal de reset da re-iniciação ao final da carga de um sketch. No segundo caso o bottoloader passa diretamente para a aplicação, não tentando estabelecer comunicação.

Com estas alterações podem ocorrer problemas quando a aplicação do Arduino utiliza comunicação serial. De um lado pode ocorrer uma re-iniciação devido a uma conexão (retirando o controle da aplicação e passando ao bootloader) e por outro o bootloader pode ficar tentando tratar uma comunicação destinada à aplicação e nunca passar o controle para ela.

ATtiny85

O attiny85 é um microcontrolador atmel de 8 pinos, o usaremos para controlar e gravar os dados do bmp180, as gravações no attiny85 são feitas a partir do arduino por SPI. O attiny85 trabalha com tensão de 2.7V a 5.5V, possui 8kb de memória flash (armazena códigos) 512bytes de memória EEPROM e 512bytes de memória SRAM(processa informações).

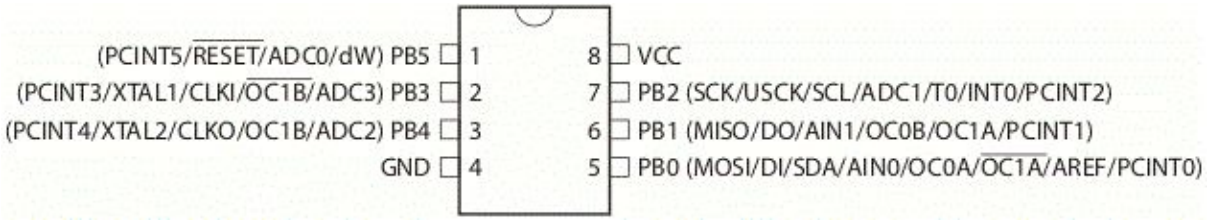

Figura 3 - Pinos attiny85

SPI: A Serial Peripheral Interface é um protocolo de dados seriais síncronos utilizados em microcontroladores para comunicação entre o microcontrolador e um ou mais periféricos. Também pode ser utilizado entre dois microcontroladores. A comunicação SPI sempre tem um master. Isto é, sempre um será o master e o restante será slave. Por exemplo, o arduino é o master e os outros periféricos são slave. Esta comunicação contém 4 conexões: MISO (Master IN Slave OUT) - Dados do Slave para Master; MOSI (Master OUT Slave IN) - Dados do Master para Slave; SCK (Serial Clock) - Clock de sincronização para transmissão de dados entre o Master e Slave; SS (Slave Select) - Seleciona qual Slave receberá os dados.

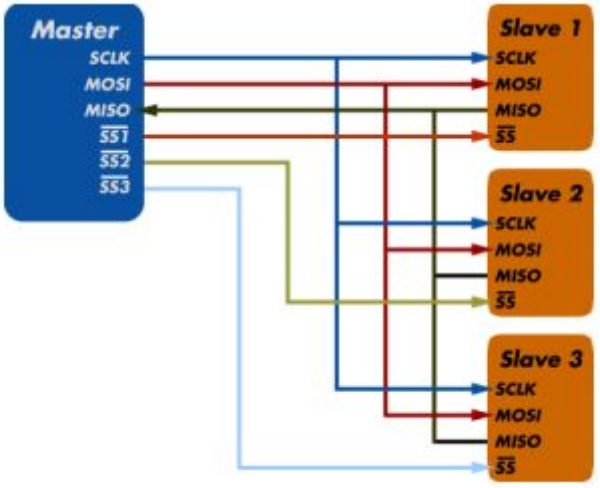

Figura 4 - Diagrama SPI

Para esse projeto tivemos que fazer algumas alterações em fuse bits do attiny85, já que esse vem configurado para apagar sua memória EEPROM toda vez que um novo código é carregado e será na EEPROM que gravaremos os dados de voo, ele também vem configurado para dividir o clock internamente por 8. Os procedimentos para essas configurações serão listados na fase de montagem.

BMP180

O sensor BMP180, fornece valores da pressão atmosférica, operando em uma faixa de 300 hPa a 1.100 hPa (altitude +9000m a -500m). Além de ser um sensor de pressão barométrica, o BMP180 também é capaz de efetuar medidas da temperatura, com a sua medição de pressão atmosférica é possível chegar a uma altura por meio de cálculos que

converte a pressão atmosférica em alturas em relação ao nível do mar, dessa forma sabendo a altura do ponto de partida é possível saber o quão alto o projétil foi.

O sensor propriamente dito é a cápsula de 5 mm x 5 mm que se encontra no centro da placa de circuito impresso, o sensor digital BMP180 é baseado no efeito piezoresistivo, onde a pressão aplicada sobre certos materiais, semicondutores, afeta a resistência elétrica. A figura 1 mostra sensor BMP180.

Figura 5 – lado superior do sensor BMP180 e lado inferior

Especificações: Por ser um CI que o setor da eletrônica já está manuseando, ter uma resolução de 0,6hPa (50cm) e por ser um circuito que usa poucas portas, apenas 4, como mostra figura 2, é uma boa opção para ser o altímetro dos foguetes.

- Tensão de entrada (VIN);
- Aterramento (GND);
- Transferência de dados (SDA – Serial Data – porta analógica);
- Temporiza a ligação entre os CIs (SCL – Serial Clock - porta analógica);

Características:

- Tensão de operação: 1,8 – 3,6V;
- Consumo de corrente: 5uA;
- Tempo de reação: 7,5ms;
- Dimensões 14 x 12 mm;
- Peso: 1,2g;

A comunicação com o sensor é feita por I2C. I2C é a sigla de Inter-Integrated Circuit, e basicamente é um protocolo de comunicação entre dispositivos que “falam” I2C. O I2C

trabalha no modelo master-slave, com pelo menos um dispositivo atuando como master, e os demais dispositivos atuando como slave. A função do master é coordenar a comunicação, sendo que é ele quem envia informações a determinado slave ou consulta informações. O Arduino vem com pinos próprios para a conexão I2C, no caso do Uno e derivados os pinos são sempre o 4 (SDA) e 5(SCL). No caso do Arduino Mega, os pinos utilizados são o 20 (SDA) e 21 (SCL). SDA significa Serial Data e SCL significa Serial Clock. SDA é o pino que efetivamente transfere os dados, e SCL serve para temporização entre os dispositivos, de modo que a comunicação pela DAS possa ter confiabilidade. Como podem observar tanto o envio quanto a recepção de dados é realizada utilizando a linha SDA, ou seja, é uma linha bi-direcional de comunicação, ora estamos enviando dados por este pino, ora estamos recebendo dados.

Ruído bmp180 : 0.25m

Conversion time pressure	tc_p_low	ultra low power mode	3ms	4.5ms
	tc_p_std	standard mode	5ms	7.5ms
	tc_p_hr	high resolution mode	9ms	13.5ms
	tc_p_luhr	ultra high res. mode	17ms	25.5ms
	tc_p_ar	Advanced res. mode	51ms	76.5ms
Start-up time after power-up, before first communication	tStart			10ms

Bibliotecas

Wire

A biblioteca Wire permite a comunicação I2C/TWI entre dispositivos. A sigla I2C faz referência a “Inter-Integrated Circuit” e é basicamente um protocolo de comunicação entre dispositivos que utilizam I2C respectivamente. O protocolo I2C utiliza o modelo master-slave, normalmente com uma CPU trabalhando como master e um ou mais dispositivos trabalhando como slaves, sendo esses dispositivos chamados periféricos. A função dos master é coordenar a
comunicação, sendo que ele envia informações a determinado periférico ou consulta informações. Tal consulta é feita normalmente por meio de interrupções, onde o periférico passa as informações ou os resultados da atividade que estava sendo executada para o master. O I2C é composto de dois barramentos principais para o estabelecimento da comunicação.

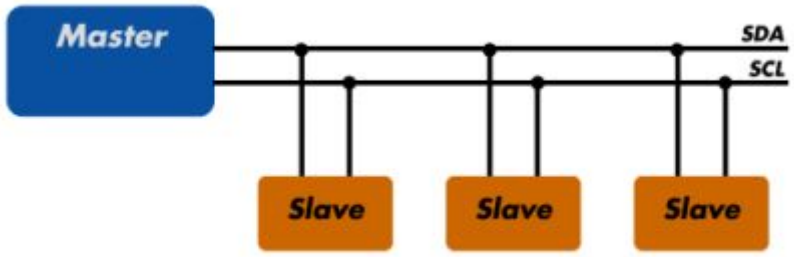

Figura 6 - Diagrama de funcionamento biblioteca Wire

O barramento SDA (Serial Data), o qual efetivamente transfere os dados e o barramento SCL (Serial Clock), que serve para ajustar a temporização entre os dispositivos. O barramento SDA é utilizado tanto para envio quanto para recepção dos dados, ou seja, é um barramento bi-direcional. Vários dispositivos utilizam a comunicação I2C, como o próprio Arduino, memórias externas como EEPROM, visores e sensores de diversos tipos, como o barômetro utilizado no projeto do altímetro.

TinyWireM

Problemas utilizando o ATtiny: A biblioteca Wire utiliza uma grande quantidade de memória para buffers, são três buffers de byte diferentes, cada um com 32 bytes de tamanho. Pode não parecer muito para microcontroladores mais potentes, porém o ATtiny possui menos RAM disponível. Seria necessário, talvez, alterar o tamanho do buffer para 6 para que tenha o funcionamento desejado. Todavia não basta apenas alterar na IDE #define TWI_BUFFER_LENGHT 6 pois não funcionaria. A maneira como o Arduino compila o código não é tão óbvia. Necessitaria, portanto, mudar o tamanho do buffer no cabeçalho da biblioteca, o que também inclui a fonte da biblioteca (twi.c). A biblioteca TinyWire é muito semelhante à Wire, porém utilizada para fazer a comunicação I2C entre dispositivos como o ATtiny85 e o sensor BMP180 utilizados no projeto do altímetro.

SoftwareSerial

Quando utilizando o Arduíno com a IDE sozinho, normalmente é necessário apenas utilizar Serial.print e o monitor serial, para obter alguma informação vinda do mesmo. Porém, quando utilizando um “tiny”, não é possível fazer a comunicação dessa forma. Com o TinyDebugSerial, é possível apenas fazer a comunicação em um sentido, ou seja, do ATtiny para o Arduíno, porém isso é o bastante, visto que as informações estarão contidas no ATtiny85 e queremos mostra-las no monitor. O TinyDebugSerial funciona para clocks inferiores a 8MHz porém apenas em algumas taxas de transmissão que não incluem 19200, logo teremos que optar por SoftwareSerial que trabalha nessa faixa porém só em 8MHz.

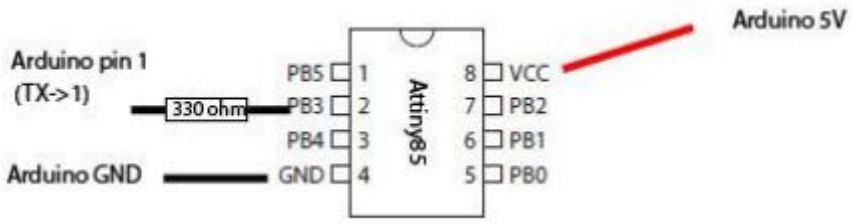

Figura 7 - Esquema de montagem para uso da biblioteca SoftwareSerial

O pino PB3 (pino físico 2), estabelecerá a comunicação TX para o Arduíno em seu pino 1 (pino TX). Além disso, é necessária a conexão, no Arduíno, do reset com seu ground e dos grounds no ATtiny e no Arduíno.
Um exemplo de código pode ser:

```
#include <SoftwareSerial.h>

SoftwareSerial mySerial = SoftwareSerial(4, 3); //(rx, tx)

// Put this in setup()

mySerial.begin( 19200 );

// Put this in loop()

mySerial.print("Sent :");

mySerial.println(buffer);
```

EEPROM

O attiny85 não tem suporte para cartão de memória, logo usaremos sua memória EEPROM para gravar os dados que serão obtidos do sensor. O attiny85 possui 512 bytes para memória EEPROM(armazena dados). Para escrever, ler e apagar dados da EEPROM do attiny85 usamos exatamente os mesmos procedimentos do arduino, logo já temos exemplos do arduino para isso que podem ser usados como o auxílio da biblioteca EEPROM. Consultando o datasheet do attiny85 temos que podemos realizar o procedimento de apagar/escrever nessa memória 100000 vezes.

Minimum Wait Delay Before Writing the Next Flash or EEPROM Location	
Symbol	Minimum Wait Delay
tWD_EEPROM	4ms

Procedimentos para montagem

Passo 1

A IDE do arduino pode ser baixada em https://www.arduino.cc/en/Main/Software
Na versão mais recente (a partir da 1.6) é possível procurar bibliotecas na própria IDE e instalar, no caso do attiny85, as modificações necessárias para que apareça como opção de placa. Para que as placas apareçam é preciso ir em arduino>arquivo>preferências em URLs adicionais cole o link
https://raw.githubusercontent.com/damellis/attiny/ide-1.6.x-boards-manager/package_damellis_attiny_index.json

No caso de estarmos usando outra versão segue o procedimento:
1. No link faça o download da versão apropriada para sua IDE : https://code.google.com/archive/p/arduino-tiny/
2. Vá na pasta onde o arduino foi instalado e dentro da pasta hardware cole a pasta do arquivo baixado.
3. O attiny85 e suas variações devem aparecer na ide em ferramentas > placas > attiny85.

Passo 2

Para usar o arduino como um intermediário com o attiny85 precisamos seguir alguns passo:
1. Execute o exemplo ArduinoISP.
   - Este esboço transforma o Arduino em um AVRISP.
   - Por padrão, os pinos de hardware SPI pinos MISO, MOSI e SCK são usados.
2. Faça a seguinte montagem:
   - Desconecte o arduino da entrada usb.
   - 5v do arduino para o pino de alimentação do attiny85(pino físico 8).
   - GND do arduino para GND do attiny85(pino físico 4).
   - pino 10 do arduino no pino de reset do attiny85 (pino físico 1).
   - pinos 11, 12 e 13 do arduino nos pinos físicos 5, 6 e 7 respectivamente do attiny85.
   - um capacitor de 10uF entre o reset do arduino(polo positivo - perna maior) e ground do arduino(polo negativo - geralmente marcado). O capacitor impede que o arduino reinicie no início do upload de código.
   - Na IDE escolha a placa como attiny85 8 MHz e o programador como Arduino as ISP(configuração que permite o arduíno operar como intermediário).
   - Conecte o arduino na entrada usb e não esqueça de marcar a porta.
   - Vá em ferramentas > burn bootloader.
   - Agora já podemos usar o arduíno para gravar um código no attiny85.
   - Usando a mesma configuração física abra o exemplo blink e execute. (fique atento a alterações nos pinos que estão no código blink do arduíno - substitua o pino 13 por pino 0)

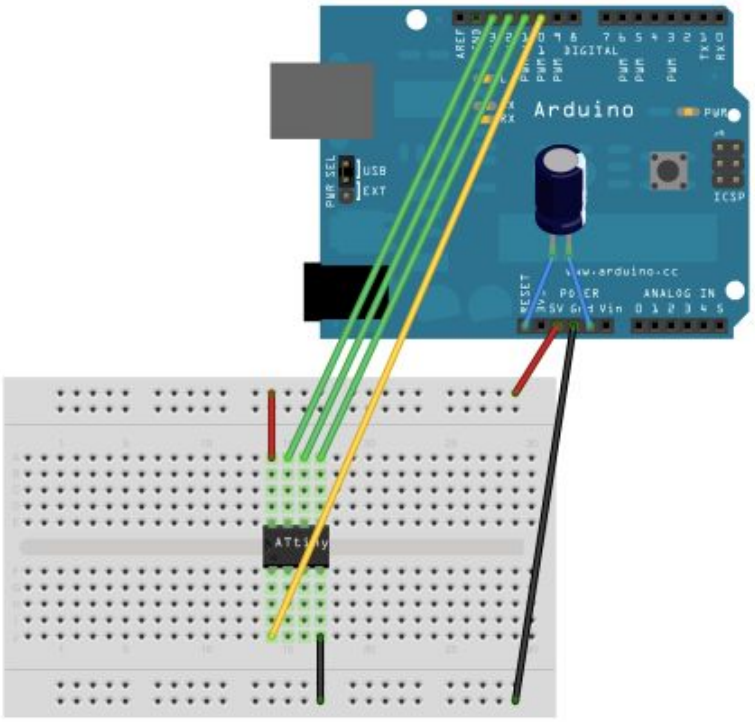

Figura 8 - esquema da montagem para gravação de dados no attiny85

Passo 3

Toda vez que “queimamos bootloader” para trabalhar com attiny85 definimos fuses padrão para esse dispositivo (como o clock), um padrão é ter o fuse EESAVE(1), esse fuse é responsável por agapar a memória EEPROM do attiny85 toda vez que um novo esboço é carregado, logo foi necessário altera o fuse EESAVE para 0, para isso foram seguidos alguns passos:

1. Instalação de WinAVR (para permitir a configuração dos fuses através do prompt de comando)
2. Calculadora de fuses http://www.engbedded.com/fusecalc (a calculadora já gera o hexadecimal necessário para o código)
3. Carregar o esboço ArduinoISP para o arduino, fazer a conexão entre arduino e attiny85 já conhecida para gravação e passar um esboço blink, mantenha a conexão física e siga para o próximo passo.
4. Para checar o dispositivo abrimos o prompt e inserimos o código ”avrdude -c stk500v1 -p attiny85 -P COM3 -b 19200” (-c = programador onde stk500v1 equivale ao arduino como ISP, -p = parte no caso o microcontrolador que vamos programar, -P porta em que o arduino está conectado, -b 19200 é o baud (taxa de transmissão) do arduinoISP), depois de executado rode de novo a linha ”avrdude -c stk500v1 -p attiny85 -P COM3 -b 19200”.
5. Note que o baud foi definido em 19200, vale lembrar que até então usávamos tinydebugserial.h que não funciona nessa faixa de transmissão,

logo vamos optar por usar softwareserial.h que trabalha nessa faixa.

6. Uma última consideração sobre softwareserial é que ele só trabalha com attiny85 em 8MHz, logo precisamos definir o fuse para isso, note que para fuse padrão esses 8MHz são divididos em 8 (resultando em 1MHz) , logo na calculadora de fuses vamos desmarcar essa opção de dividir o clock. Estava em CKOUT=0 e vai para CKOUT=1.
7. Configurando os fuses, segue a linha que deve ser inserida no prompt ”avrdude -c stk500v1 -p attiny85 -P COM3 -b 19200 -U lfuse:w:0xe2:m -U hfuse:w:0xd7:m -U efuse:w:0xff:m”, configuração terminada.

Passo 4

O bmp 180 é o sensor de temperatura e pressão que será usado no projeto. Para usá-lo com o arduino é necessário adicionar sua biblioteca, que pode ser incluída diretamente a partir da ide 1.6 ou incluindo bibliotecas através de arquivo baixado.
Link : https://github.com/adafruit/Adafruit-BMP085-Library
Possui apenas 4 entradas: alimentação, terra, SCL (clock) e SDA (transferência de dados).
Possui um regulador de tensão, logo não há problema em alimentá-lo em 5V.
Na biblioteca baixada existem exemplos que indicam como fazer a ligação nos pinos do arduino.
Para o attiny pino 5 - SDA, pino 7 - SCL.

Figura 9 - Imagem do bmp180

Passo 4

Primeiro problema: A biblioteca do bmp180 é usada juntamente com outra biblioteca geral de sensores do arduino. Para usar o bmp180 com o attiny85 não podemos usar essas bibliotecas pelo espaço em memória ocupados por elas, por isso toda a configuração do

bmp180 terá que estar no código principal de funcionamento do microaltímetro. No datasheet do bmp180 são encontradas as informações necessárias para fazer isso. Algumas informações são relevantes a respeito da comunicação do sensor com o arduino. A comunicação é feita por I2C que trabalha no modelo master-slave. A função do master é coordenar a comunicação, sendo que é ele quem envia informações a determinado slave ou consulta informações. Para a comunicação entre bmp180 e o arduino existe a biblioteca Wire, para trabalhar como attiny temos que utilizar a biblioteca TinyWireM.

Logo no código final teremos que utilizar a biblioteca TineWireM e utilizamos ela para receber os dados do sensor de pressão. Já existe uma biblioteca própria para trabalhar com o attiny85 e bmp180 observando todas essas informações, ela se chama tinyBMP085 e vamos utilizá-la.

Passo 5

Como temos apenas 512 bytes disponíveis para gravar nossos dados de vôo e tendo em vista que não conseguimos gravar dados do tipo float na EEPROM, escrevemos um código que se encontra no drive CODIGO05052016arduinobmpeeprom. Esse código retrata essas condições já usando o bmp180 (porém com auxílio das bibliotecas pois o teste foi no arduino). Condições do código:

- Como não conseguimos gravar dados do tipo float e o bmp180 retorna dados com duas casas decimais, realizando algumas considerações, optamos por salvar os dados dois a dois, logo iremos pegar um valor lido pelo bmp180 e o quebraremos em dois valores e salvaremos em endereços de memória sequenciais. Logo cada byte guardará um valor, e cada 2 bytes de memória teremos o valor completo do dado. Como cada byte consegue gravar um número de até 2^(8) = 256, logo esse será o número de maior valor que será gravado, essa foi uma das considerações feitas. Como não esperamos chegar a altitudes de 2560,00 m, o código multiplica o valor lido por 10, pegando assim 5 dígitos, depois o divide por 100 e retorna a parte inteira e salva, em seguida salva a parte fracionada. No caso de 2560,00m , seguindo os passos teríamos: 2560,00 x 10 = 25600,0; 25600(descartando a parte depois da vírgula); 25600/100= 256(parte inteira salva em um endereço de memória n); 00(resto da divisão salvo em um endereço n+1). Logo nosso valor salvo em n não ultrapassará 256, e nosso valor salvo em n+1 não ultrapassará 99.

Passo 6

Para testar o código de vôo agora com o attiny85 precisaríamos de obter uma resposta para saber se o código funcionará e também para conseguir ler os dados posteriormente. Poderíamos usar leds para testar o funcionamento do código do bmp180(passo 4). Para fins mais práticos, com a intenção de ler os dados do attiny85, para realizar o passo 5 com o attiny85 precisamos mostrar os dados através de um monitor serial, para isso usaremos SoftwareSerial. O código a seguir:

```
#include <tinyBMP085.h>
#include <TinyWireM.h>
#include <SoftwareSerial.h>

tinyBMP085 bmp;
SoftwareSerial SSerial = SoftwareSerial (4, 3);;


void setup()
{
  bmp.begin();
  SSerial.begin(19200);
}

void loop()
{
    int pres = bmp.readPressure();
    int temp = bmp.readTemperature10C();
    int alt = bmp.readAltitude();
    int altMM = bmp.readAltitudemm();
    int altSTDdm = bmp.readAltitudeSTDdm();

  SSerial.println("Pressao");
  SSerial.println(pres);
  SSerial.println("Temperatura");
  SSerial.println(temp);
  SSerial.println("Atitude");
  SSerial.println(alt);
  SSerial.println("Atidude em mm");
  SSerial.println(altMM);
  SSerial.println("Atidude certa em dm");
  SSerial.println(altSTDdm);
delay(500);

}
```

Passo 7

Por fim, considerando todas as observações listadas até agora, chegamos em um código de escrita e leitura usando todas as bibliotecas necessárias:

ESCREVENDO EEPROM:

```
#include <EEPROM.h>
#include <tinyBMP085.h>
#include <TinyWireM.h>
#define BMP085_ULTRAHIGHRES  3  // modo de operação mais demorado porem mais preciso

tinyBMP085 bmp;

int addr = 0;


void setup()
{
  pinMode(4, OUTPUT);
  digitalWrite(4, HIGH);
```

```
  delay(10000);
  for (int i = 0 ; i < EEPROM.length() ; i++) {    //limpando a eeprom
    EEPROM.write(i, 0);
  }
  bmp.begin();
  digitalWrite(4, LOW);    // turn the LED off by making the voltage LOW

}

void loop()
{
  digitalWrite(4, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(4, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
  uint8_t altSTDdm = bmp.readAltitudeSTDdm()/100;
  uint8_t altSTDdm1 = bmp.readAltitudeSTDdm()%100;
  EEPROM.write(addr, altSTDdm);
  EEPROM.write(addr+1, altSTDdm1);

  delay(100);

  addr = addr + 2;
  if (addr == 100){
    while(1);
  }


}
```

LENDO EEPROM:

```
#include <EEPROM.h>
#include <tinyBMP085.h>
#include <TinyWireM.h>
#include <SoftwareSerial.h>

tinyBMP085 bmp;
SoftwareSerial SSerial = SoftwareSerial(4,3);

int addr = 0;

void setup()
{
  bmp.begin();
  SSerial.begin(19200);
}

void loop()
{

  uint8_t value = EEPROM.read(addr);
  uint8_t value1 = EEPROM.read(addr+1);

  SSerial.print(addr);
  SSerial.print(" ");
  SSerial.print(value, DEC);
  SSerial.print(value1, DEC);
  SSerial.println();
```

Referências

https://sites.google.com/site/ronaldoecaetano/microcontrolador/atmega328

http://br-arduino.org/2015/01/programar-o-attiny85-com-arduino-como-funciona.html