

CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA

Faculdade de Tecnologia de Sorocaba – “José Crespo Gonzales”

Curso Superior de Tecnologia em Eletrônica Automotiva

VITOR GABRIEL ARTMANN

**USO DO MICROCONTROLADOR STM32 COM AMBIENTE DE
DESENVOLVIMENTO INTEGRADO ARDUÍNO**

SOROCABA

2023

VITOR GABRIEL ARTMANN

USO DO MICROCONTROLADOR STM32 COM AMBIENTE DE DESENVOLVIMENTO INTEGRADO ARDUÍNO

Trabalho apresentado à Faculdade de Tecnologia de Sorocaba - “José Crespo Gonzales” como requisito para aprovação na disciplina Projeto de Trabalho de Graduação I, sob a orientação do Professor Dr. José Luiz Antunes de Almeida

Sorocaba

2023

RESUMO

ARTMANN, Vitor Gabriel. **Uso do microcontrolador STM32 com Ambiente de Desenvolvimento Integrado Arduíno**. XX fls. Projeto de Trabalho de Graduação I – Curso Superior de Tecnologia em Eletrônica Automotiva. Faculdade de Tecnologia de Sorocaba - “José Crespo Gonzales”. Centro Estadual de Educação Tecnológica Paula Souza. Sorocaba. 2023.

Este trabalho visa apresentar um guia de utilização básica do microcontrolador STM32 com a IDE do Arduíno, com a implementação de projetos elementares que ajudam a entender as principais diferenças entre o Arduíno UNO clássico, que utiliza um microcontrolador AVR ATmega 328P de 8 bits e o avançado STM32 de 32 bits. Serão apresentados três projetos que ajudam a ilustrar essas diferenças e entender o funcionamento do poderoso microcontrolador STM32.

Palavras-chave: microcontrolador; STM32; Arduíno;.

LISTA DE ILUSTRAÇÕES

Figura 1	Placa de Desenvolvimento Blue Pill.....	07
Figura 2	Diagrama de pinos Blue Pill.....	08
Figura 3	Placas STM32.....	09
Figura 4	Esquema de ligação.....	11
Figura 5	Modos de inicialização do MCU.....	08
Figura 6	Seleção da placa.....	12
Figura 7	Código - blink.....	13
Figura 8	Controle de brilho de led com potenciômetro.....	14
Figura 9	Controle de brilho de led com potenciômetro - esquema de ligação prática.....	14
Figura 10	Código - Controle de brilho de led com potenciômetro	15
Figura 11	Ligação entre dispositivos CAN sem uso de transceiver.....	17
Figura 12	Esquema de ligação – Comunicação com rede CAN.....	18
Figura 13	Esquema de ligação prática – Comunicação com rede CAN.....	18
Figura 14	Dados recebidos por rede CAN.....	19

SUMÁRIO

1	INTRODUÇÃO.....	06
2	CONHECENDO O HARDWARE.....	06
3	CARACTERÍSTICAS GERAIS.....	07
4	PROGRAMANDO.....	09
4.1	Primeiros passos.....	09
4.2.	Carregando o código.....	10
4.3	Blink.....	13
4.4	Controle de brilho de led com potenciômetro.....	13
4.5	Comunicação com rede CAN.....	17
5.	CONSIDERAÇÕES FINAIS.....	20
6.	REFERÊNCIAS.....	21

1. INTRODUÇÃO

O STM32 é um microcontrolador extremamente poderoso. O núcleo ARM Cortex-M3, presente na família de microcontroladores STM32 da *ST Microelectronics*, representa um avanço baseado na arquitetura ARM de 32 bits. O chip STM32 possui, além do core ARM Cortex-M3, uma quantidade de periféricos internos suficiente para dispensar a ligação externa da maioria dos itens comumente utilizados nos microcontroladores de 8 bits mais difundidos do mercado por um custo baixíssimo de, em média, três dólares a unidade.

Na última década, boa parte dos fabricantes adotaram o ARM7 e o ARM9 como a CPU de suas linhas de microcontroladores, embarcando-os em uma diversa gama de equipamentos, que vão desde simples relógios, passando por complexas máquinas de automação, e chegando até mesmo a serem utilizados em sofisticados equipamentos de uso médico e hospitalar.

Este trabalho visa apresentar um guia de utilização básica do microcontrolador STM32 com a IDE do Arduíno, com a implementação de projetos elementares que ajudam a entender as principais diferenças entre o Arduíno UNO clássico, que utiliza um microcontrolador AVR ATmega 328P de 8 bits e o avançado STM32 de 32 bits.

2. CONHECENDO O HARDWARE

Neste trabalho, o hardware usado será a placa chamada Blue Pill, uma plataforma de desenvolvimento que utiliza o microcontrolador STM32. Assemelhando-se em tamanho ao Arduíno Nano, a Blue Pill utiliza em uma de suas versões o STM32F103C8T6. Esta será a versão da Blue Pill utilizada neste trabalho.

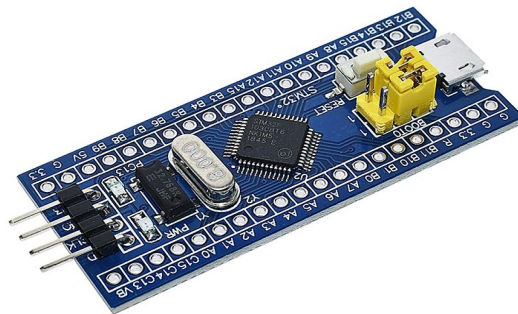
O nome do microcontrolador, na verdade, é uma referência às suas características, conforme pode-se ver a seguir:

- STM: refere-se à fabricante, STMicroelectronics
- 32: arquitetura de 32 bits

- F103: arquitetura ARM Cortex M3
- C: 48 pinos
- 8: memória flash de 64kb
- T: refere-se ao encapsulamento do CI, o LQFP de 48 pinos
- 6: refere-se à temperatura de operação

Uma rápida busca pela internet pelo termo “placa Blue Pill”, retorna vários resultados, incluindo alguns sites de vendas, onde ela pode ser achada pelo preço de, em média, aproximadamente 25 a 35 reais. A seguir, uma imagem do aspecto geral da placa Blue Pill.

Figura 1 - Placa de Desenvolvimento Blue Pill



Fonte: <https://www.botnroll.com/pt/arduino-controladores/3577-placa-de-desenvolvimento-stm32f103-blue-pill.html>

3. CARACTERÍSTICAS GERAIS

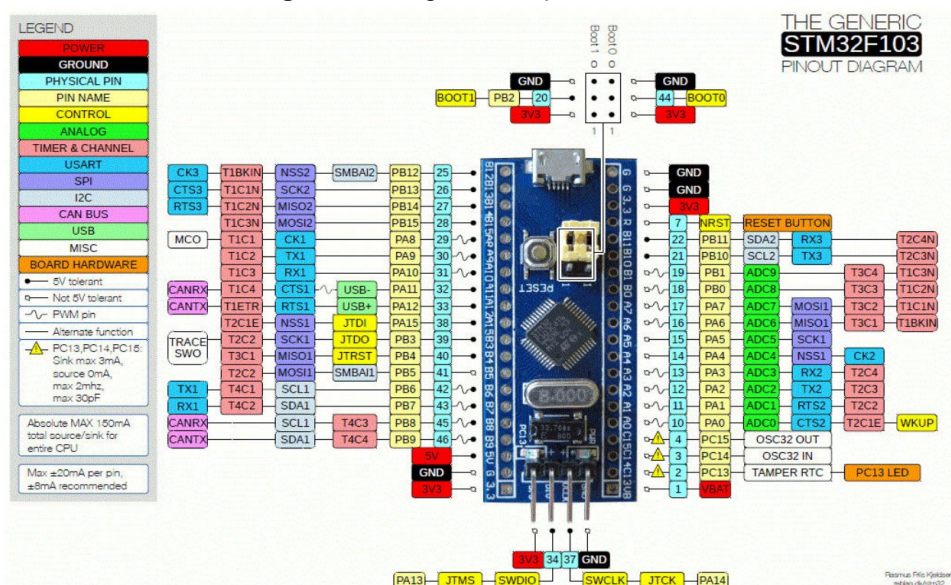
Apesar do STM32 funcionar com 3,3V, a Blue Pill possui um regulador de tensão de 5V, o que possibilita que a placa seja alimentada com 5V diretamente da entrada USB do computador, por exemplo. Além disso, mesmo que o MCU opere a 3,3V, grande parte de seus pinos GPIO são tolerantes a 5V, o que significa que são capazes de operar como entrada com nível lógico de 5V. Existem também dois LEDs integrados, um usado para indicação de alimentação e o outro conectado ao pino

GPIO PC13. A placa também possui 4 pinos que são usados para alternar o modo de inicialização do MCU entre o modo de programação e o modo de operação, selecionados através de jumper. Esse funcionamento será abordado mais detalhadamente posteriormente. A Blue Pill ainda conta com um cristal oscilador de 32 KHz, usado pelo RTC (Real Time Clock) interno para aplicações envolvendo o uso de relógio. A seguir, os principais periféricos do STM32F103:

- 4 portas GPIO de 16 bits (a maioria é tolerante a 5 Volts);
- 3 USART (Receptor e Transmissor Síncrono e Assíncrono);
- 2 controladores I2C;
- 2 controladores SPI;
- 2 ADC (conversor analógico para digital);
- 2 DMA (controladores de acesso direto à memória);
- 4 temporizadores;
- Temporizadores Watchdog;
- 1 controlador USB;
- 1 controlador CAN;
- 1 gerador CRC;
- RAM estática de 20K;
- Memória Flash 64K (ou 128K);
- CPU ARM Cortex M3, com clock máximo de 72 MHz.

Na figura 02, temos o diagrama de pinos da Blue Pill, muito útil ao se utilizar a placa:

Figura 2 - Diagrama de pinos Blue Pill



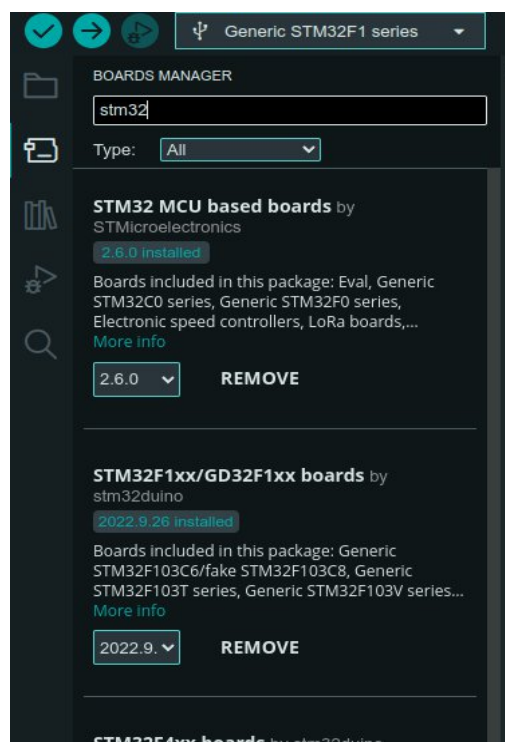
Conforme pode-se observar pelo diagrama acima, vale destacar que a corrente máxima que pode circular pelos GPIOs é de 20mA, sendo que o recomendado é de 8mA, o que é cerca de metade do que suporta o Arduino Uno, portanto deve-se ficar atento à isso sob risco de danificar a placa.

4. PROGRAMANDO

4.1 PRIMEIROS PASSOS

O primeiro passo para programar a Blue Pill usando a IDE do Arduino é fazer a instalação dos drivers para as placas STM32. Na versão mais recente da IDE (2.2.1), basta abrir o gerenciador de placas e digitar “STM32”. Para rodar os sketches que serão apresentados aqui, será necessário instalar os dois drivers mostrados a seguir:

Figura 3 - Placas STM32



Fonte: Autor

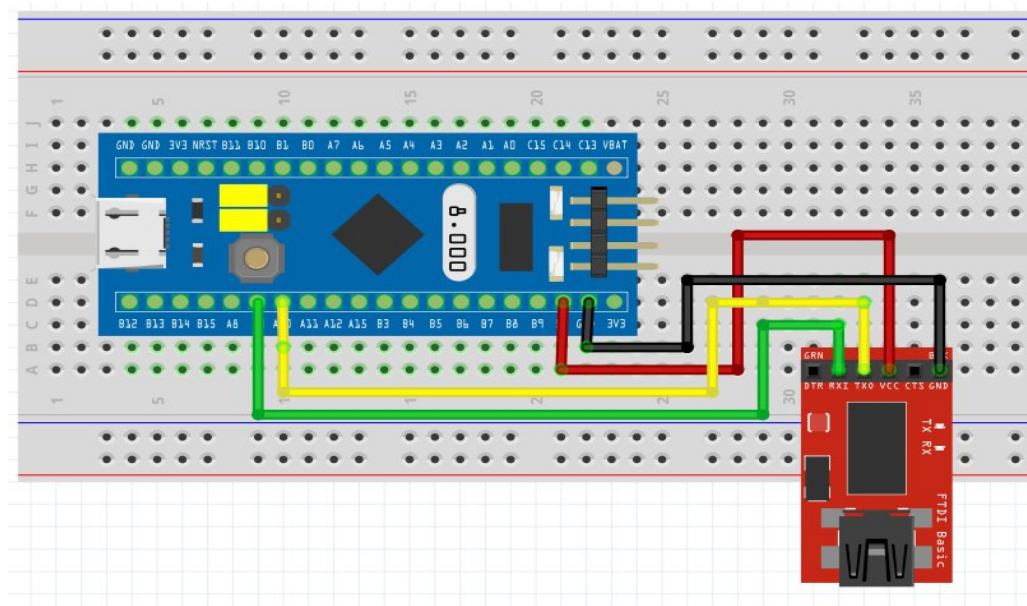
Adicionalmente, para rodar o sketch “Comunicação com Rede CAN” será necessário instalar o STM32CubeProgrammer, disponível em: <https://www.st.com/en/development-tools/stm32cubeprog.html#get-software>.

4.2. CARREGANDO O CÓDIGO

Para fazer o upload dos códigos há diversas maneiras. Pode-se utilizar um carregador ST-LINK, específico para microcontroladores da família STM8 e STM32, um conversor USB-Serial ou ainda o próprio conector USB da placa, sendo necessário nesse caso, carregar um bootloader. Neste trabalho, utilizaremos um módulo FTDI conversor USB-Serial. O uso desse tipo de conversor, além de muito simples, é interessante pois sua aplicação não se limita apenas aos microcontroladores da ST, uma vez que também pode ser utilizado com algumas placas Arduino. Existem vários modelos desses conversores disponíveis no mercado, sendo um dos mais comuns, e o que será utilizado nesse trabalho, aquele baseado no chip CH340, cuja utilização depende da instalação do driver apropriado, disponível em: <https://sparks.gogo.co.nz/ch340.html>.

Um módulo conversor USB-Serial apresenta, via de regra, os seguintes pinos: VCC, GND, Rx e Tx. Esses serão os pinos utilizados para programar o STM32. O pino Rx do conversor deverá ser conectado ao pino A9 da Blue Pill e o pino Tx ao pino A10. Além disso, a placa poderá ser alimentada tanto pelo pino de 3v3 quanto pelo pino 5V, observando-se a tensão de alimentação que o conversor fornece. A seguir, um exemplo do esquema de ligação para programação:

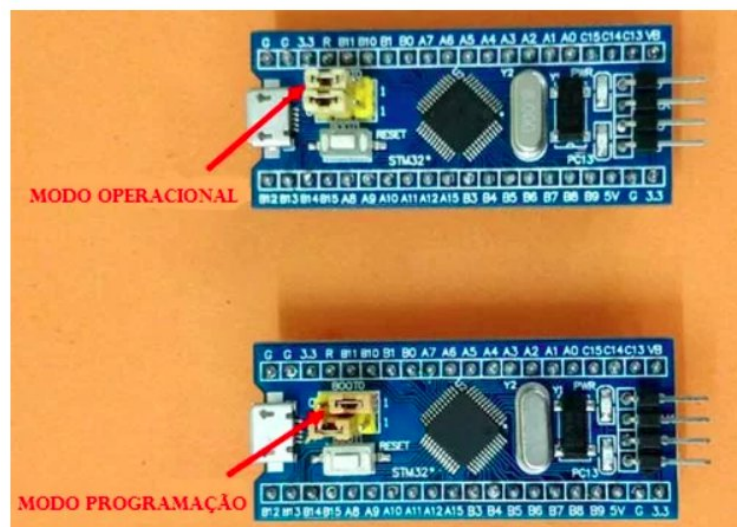
Figura 4 - Esquema de ligação



Fonte: Autor

A Blue Pill apresenta 4 pinos que são usados para alternar o modo de inicialização do MCU entre o modo de programação e o modo operacional, selecionados através de jumper e localizados próximos ao conector USB da placa. A figura abaixo ilustra como devem ser posicionados os jumpers para cada modo. Para carregar um código na placa, o jumper deve estar posicionado em modo programação.

Figura 5 - Modos de inicialização do MCU

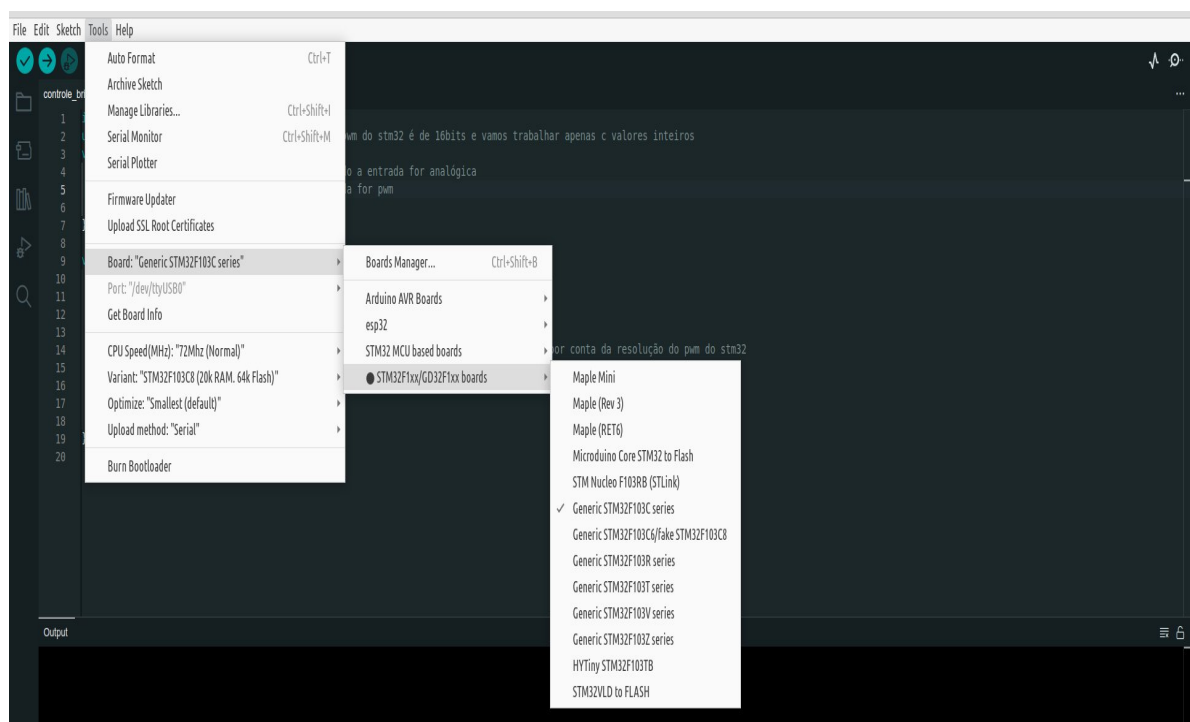


Fonte: <https://capsistema.com.br/index.php/2020/12/12/introducao-ao-stm32-circuito-azul-usando-arduino-ide-led-piscando/>

Ao posicionar o jumper no modo programação, assim que se realiza o upload do código na placa o programa irá começar a rodar. No entanto, uma vez que o programa já esteja rodando, caso seja necessário carregar novamente algum código, deve-se pressionar o botão de reset da placa, caso contrário, a tentativa de upload resultará em erro. Ao pressionar o botão de reset, o programa gravado na memória deixará de ser executado e a placa ficará novamente apta a receber um novo código. Com o jumper no modo de inicialização “operacional”, ao ser alimentada, a placa rodará o programa salvo na memória e caso seja resetada, reiniciará o programa normalmente. Com o jumper no modo de inicialização “operacional”, qualquer tentativa de carregar código também resultará em erro.

Para carregar um código, deve-se selecionar um dos drivers STM32 instalados e selecionar a placa correspondente, no caso da Blue Pill, a STM32F103C8T6. Em ambos os drivers, o método de upload deverá ser o *serial*. A figura a seguir ilustra essa configuração:

Figura 6 - Seleção da placa



Fonte: Autor

4.3 BLINK

O primeiro código a ser carregado na placa é o clássico pisca-led. Este código encontra-se na biblioteca de exemplos do STM32 e é muito parecido com o exemplo existente para a placa Arduino UNO, com a diferença de nomeação do pino GPIO, que na Blue Pill denomina-se PC13.

Este código irá piscar o led da placa em intervalos regulares que podem ser modificados no *delay()*. Na figura abaixo observa-se o código na íntegra:

Figura 7 - Código - blink

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin PC13 as an output.
  pinMode(PC13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(PC13, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(PC13, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

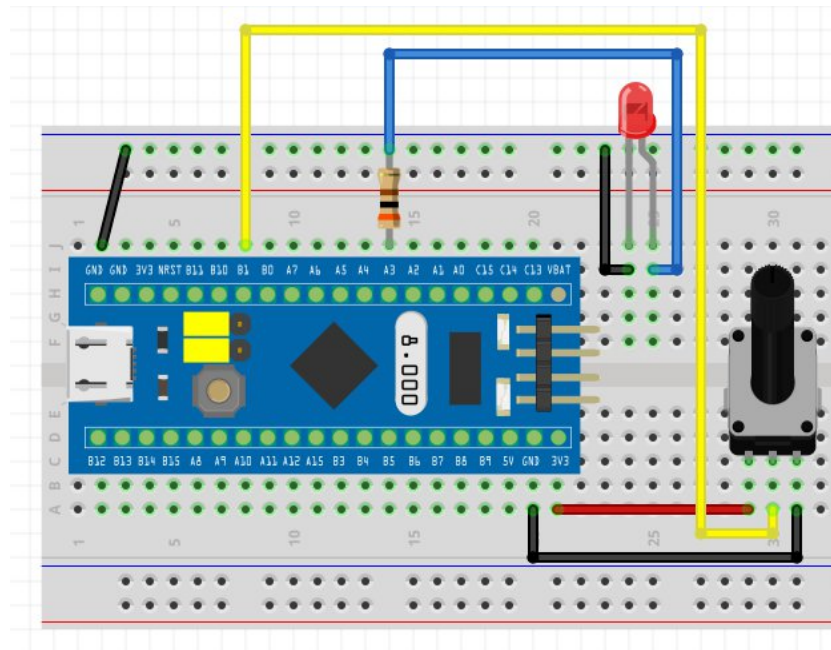
Fonte: Autor

4.4 CONTROLE DE BRILHO DE LED COM POTENCIÔMETRO

Este código controla o brilho de um led com o uso de um potenciômetro. Para executar este código, o led da placa não poderá ser usado, uma vez que está conectado ao GPIO PC13, que não suporta PWM. Portanto, será necessário utilizar um led externo e um resistor. Também será necessário utilizar um potenciômetro, que no caso deste exemplo, será um potenciômetro linear de 10K ohms.

O esquema de ligação para este exemplo está ilustrado na imagem a seguir:

Figura 8 - Controle de brilho de led com potenciômetro

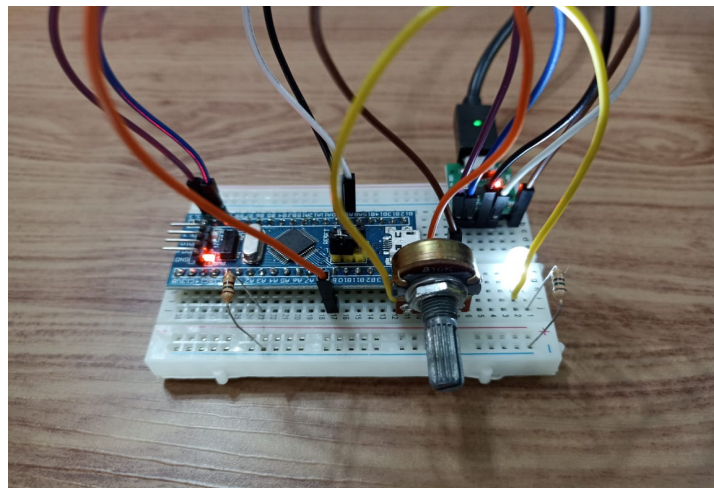


Fonte: Autor

Vale destacar que o resistor usado deve ser de aproximadamente 300 ohms, a fim de limitar a corrente que circula pelo pino no valor recomendado de cerca de 10mA. Também é necessário observar que o potenciômetro deve ser alimentado com 3,3V, uma vez que os pinos de entrada analógica do STM32 não são tolerantes a 5V, conforme pode-se observar no diagrama de pinos (figura 2).

A figura a seguir demonstra o esquema de ligação na prática:

Figura 9 - Controle de brilho de led com potenciômetro - esquema de ligação prática



Fonte: Autor

Na figura abaixo observa-se o código na íntegra:

Figura 10 – Código - Controle de brilho de led com potenciômetro

```
1
2 int x;
3 unsigned int y; //y precisa ser unsigned, pois o pwm do stm32 é de 16bits e vamos trabalhar apenas c valores inteiros
4 void setup() {
5     pinMode(PB1, INPUT_ANALOG); //deve ser usado quando a entrada for analógica
6     pinMode(PA3, PWM); //deve ser usado quando a saída for pwm
7     Serial.begin(9600);
8 }
9
10 void loop() {
11     x=analogRead(PB1); //conversor ad do stm32 é de 12 bits
12     Serial.print("pot:");
13     Serial.print(x);
14     y=map(x, 0, 4095, 0, 65535);
15     pwmWrite(PA3, y); //equivalente ao analogWrite. Deve ser usado ao invés do analogwrite por conta da resolução do pwm do stm32
16     Serial.print(" led:");
17     Serial.println(y);
18     //delay(500);
19 }
20
21
```

Output Serial Monitor x

Message (Enter to send message to 'Generic STM32F103C series' on '/dev/ttyUSB0')

```
pot:2553 led:40857
pot:2564 led:41033
pot:2558 led:40937
pot:2557 led:40921
pot:2552 led:40841
pot:2555 led:40889
pot:2560 led:40969
pot:2552 led:40841
pot:2542 led:40681
pot:2556 led:40905
pot:2559 led:40953
pot:2562 led:41001
pot:2557 led:40921
pot:2558 led:40937
pot:2558 led:4093
```

Fonte: Autor

O código acima lê o valor de tensão de entrada no pino RB1, controlado pelo potenciômetro, e gera um sinal PWM com duty cycle correspondente, no pino PA3. Além disso, ainda imprime no monitor serial os valores correspondentes à leitura do potenciômetro e duty cycle do PWM. Este código serve para demonstrar algumas das principais diferenças entre o Arduino Uno e o STM32 quando se trata do conversor analógico-digital e dos timers que geram o PWM.

O conversor A/D do Arduino Uno é um conversor de 10 bits. Isso significa que a tensão de entrada no pino analógico será representada em valores binários de 0 (0000000000) a 1023 (1111111111). Assumindo uma tensão de entrada variável de 0V a 5V, onde 0V equivalerá ao valor digital 0 e 5V equivalerá ao valor digital 1023, teremos uma resolução de 4,88mV por bit.

O conversor A/D do STM32, por sua vez, é um conversor de 12 bits. Teremos portanto, a tensão de entrada no pino analógico representada em valores de 0 (000000000000) a 4095 (111111111111). Tendo em vista que a tensão máxima que os pinos de entrada analógica suportam é de 3,3V, teremos uma resolução de 0,8mV por bit, ou seja, cerca de 6 vezes melhor do que aquela do Arduino Uno.

Outro ponto de grande diferença entre o Arduino Uno e o STM32 é o PWM. No Arduino, o timer que controla o PWM através da função *analogWrite()* é de 8 bits, ou seja, o duty cycle do PWM é representado através de valores binários que variam de 0 a 255, onde 0 representaria um duty cycle de 0% e 255 um duty cycle de 100%, seguindo basicamente a mesma lógica de representação de valores do conversor A/D.

Já no STM32, o PWM é implementado através de timers de 16 bits, por meio da função *pwmWrite()*, ou seja, seu duty cycle é representado por valores que variam de 0 a 65535. Isso representa uma resolução cerca de 257 vezes melhor.

Vale destacar também que, por conta dessa diferença, as funções usadas no código são diferentes. No STM32, para implementar um sinal PWM deve-se utilizar a função *pwmWrite()*, ao invés da *analogWrite()* do Arduino. Caso seja utilizada a *analogWrite()* no STM32, ela também irá gerar um sinal PWM, porém com sua resolução limitada a 8 bits como no Arduino Uno. No *void setup()*, a função *pinMode()* também apresenta diferenças. No STM32, deve-se declarar um pino de entrada analógica como *pinMode(pino, INPUT_ANALOG)* e uma saída PWM como *pinMode(pino, PWM)*. Vale destacar também que para que o código possa ser corretamente compilado, deve-se utilizar o driver de placa “STM32F1xx/GD32F1xx boards”. Caso seja utilizado o driver “STM32 MCU based boards”, a compilação poderá resultar em erro.

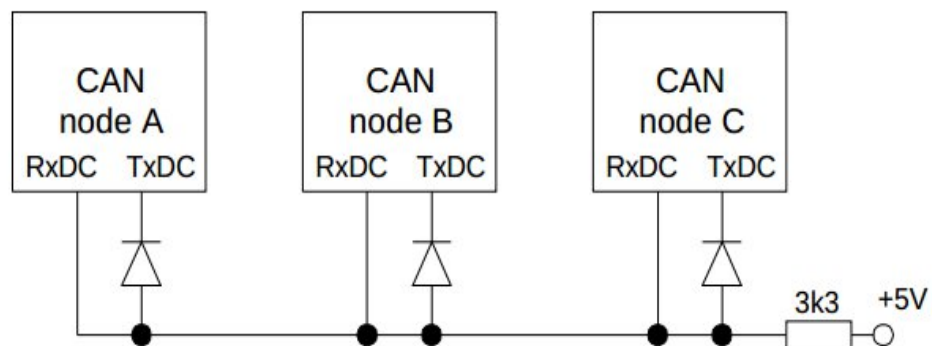
Um índice com as principais funções usadas na programação de MCUs STM32 usando linguagem Arduino, pode ser encontrado em: <http://docs.leaflabs.com/static.leaflabs.com/pub/leaflabs/maple-docs/0.0.12/language-index.html>.

4.5 COMUNICAÇÃO COM REDE CAN

O próximo código estabelece comunicação através de rede CAN entre duas placas Blue Pill e imprime no monitor serial as mensagens recebidas e enviadas por cada placa. Para rodar o programa é necessário instalar a biblioteca “STM32_CAN”, disponível em <https://github.com/pazi88/STM32_CAN>, além de possuir instalado em sua máquina o STM32CubeProgrammer.

A ligação entre as placas Blue Pill será feita diretamente, sem o uso de um CAN transceiver. Para tanto, usaremos o esquema de ligação apresentado pela Application Note “AP2921” da Siemens, que trata justamente de comunicação on-board entre dispositivos CAN sem uso de transceiver. A figura 2 da AP2921 ilustra o seguinte esquema de ligação:

Figura 11 - Ligação entre dispositivos CAN sem uso de transceiver

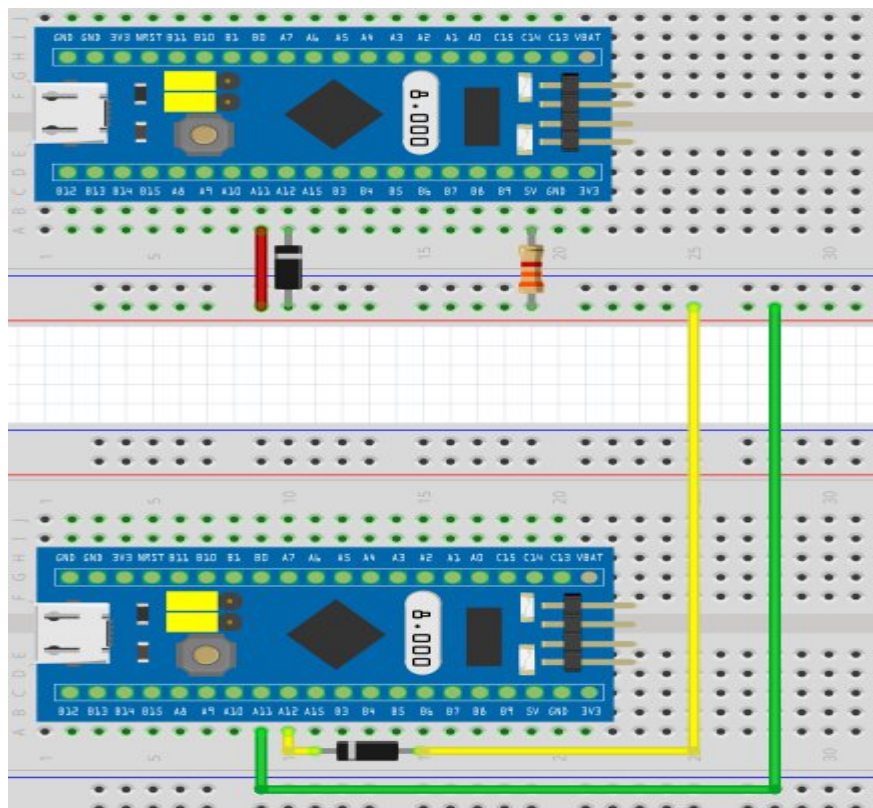


Fonte: AP2921 - Siemens

A Application Note na íntegra está disponível em: <https://www.mikrocontroller.net/attachment/28831/siemens_AP2921.pdf>.

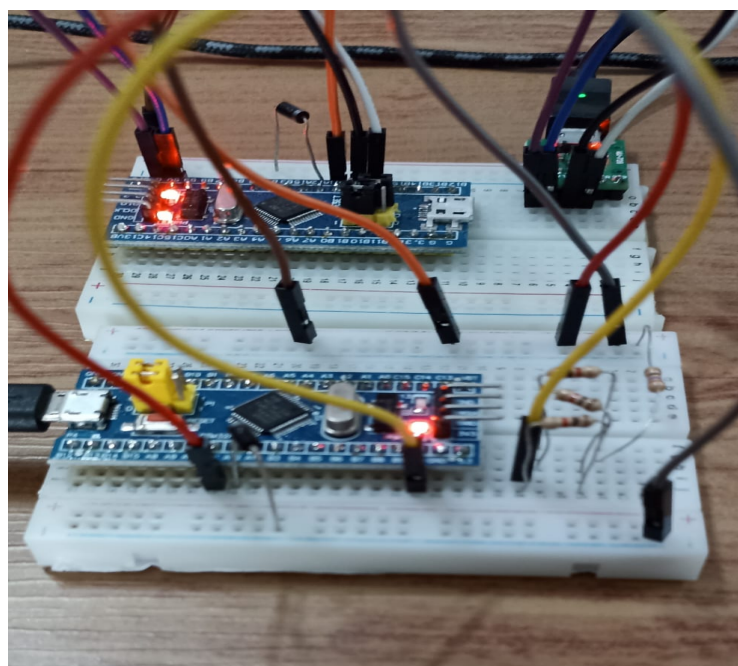
As figuras abaixo ilustram o esquema de ligação entre as placas Blue Pill:

Figura 12 - Esquema de ligação – Comunicação com rede CAN



Fonte: Autor

Figura 13 - Esquema de ligação prática – Comunicação com rede CAN



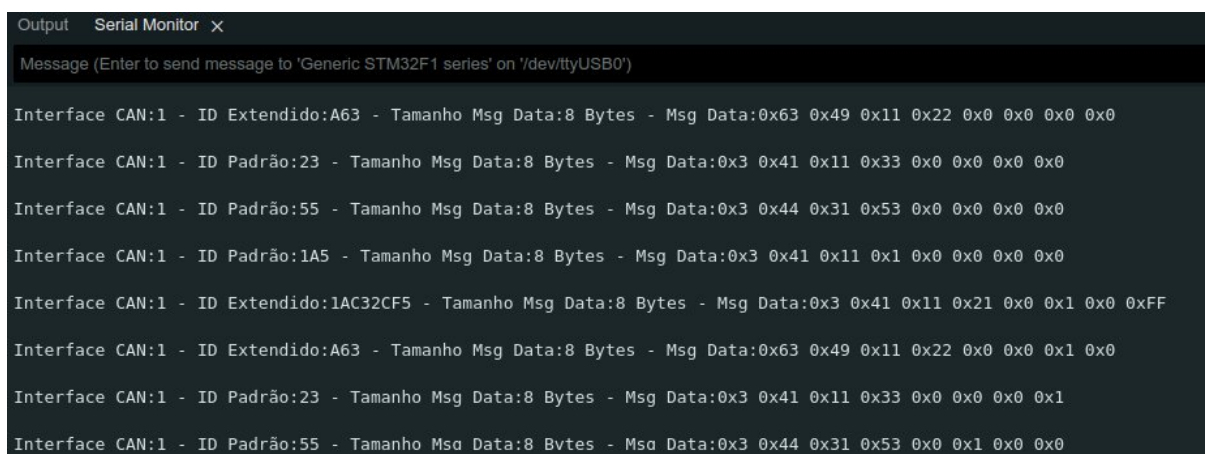
Fonte: Autor

Cada placa será carregada com um código diferente, uma para enviar os dados da mensagem e outra para recebê-los. Em ambos os casos, o ID das mensagens e o campo de dados serão impressos no monitor serial, tanto para a placa que envia as mensagens quanto para a placa que os recebe.

O ID e os dados de cada mensagem enviada são aleatórios e gerados pela placa transmissora. A cada envio de mensagem a placa piscará o led embutido. A placa receptora, por sua vez, piscará seu led toda vez que o ID e os dados da mensagem recebida corresponderem ao determinado na programação. O ID e os dados que farão a placa receptora piscar o led podem ser livremente alterados, observando-se aqueles gerados pela placa transmissora. Vale destacar que para compilar esses códigos corretamente, o driver de placas a ser utilizado deverá ser o “STM32 MCU based boards”, pois ele oferece suporte à biblioteca CAN utilizada.

Os sketches a serem carregados nas placas, bem como os outros utilizados neste trabalho estão disponíveis no repositório GitHub do autor em: <https://github.com/vitorgabriel2008>. Abaixo, uma imagem do monitor serial da placa receptora, revelando os dados recebidos através da rede CAN:

Figura 14 - Dados recebidos por rede CAN



```
Output  Serial Monitor x
Message (Enter to send message to 'Generic STM32F1 series' on '/dev/ttyUSB0')

Interface CAN:1 - ID Extendido:A63 - Tamanho Msg Data:8 Bytes - Msg Data:0x63 0x49 0x11 0x22 0x0 0x0 0x0 0x0
Interface CAN:1 - ID Padrão:23 - Tamanho Msg Data:8 Bytes - Msg Data:0x3 0x41 0x11 0x33 0x0 0x0 0x0 0x0
Interface CAN:1 - ID Padrão:55 - Tamanho Msg Data:8 Bytes - Msg Data:0x3 0x44 0x31 0x53 0x0 0x0 0x0 0x0
Interface CAN:1 - ID Padrão:1A5 - Tamanho Msg Data:8 Bytes - Msg Data:0x3 0x41 0x11 0x1 0x0 0x0 0x0 0x0
Interface CAN:1 - ID Extendido:1AC32CF5 - Tamanho Msg Data:8 Bytes - Msg Data:0x3 0x41 0x11 0x21 0x0 0x1 0x0 0xFF
Interface CAN:1 - ID Extendido:A63 - Tamanho Msg Data:8 Bytes - Msg Data:0x63 0x49 0x11 0x22 0x0 0x0 0x1 0x0
Interface CAN:1 - ID Padrão:23 - Tamanho Msg Data:8 Bytes - Msg Data:0x3 0x41 0x11 0x33 0x0 0x0 0x0 0x1
Interface CAN:1 - ID Padrão:55 - Tamanho Msg Data:8 Bytes - Msg Data:0x3 0x44 0x31 0x53 0x0 0x1 0x0 0x0
```

Fonte: Autor

5. CONSIDERAÇÕES FINAIS

Este trabalho apresentou uma visão geral, bem como, as principais diretrizes para o uso do microcontrolador STM32 com o Ambiente de Desenvolvimento Arduíno. Apresentou também, exemplos de códigos que podem ser usados para entender a diferença de hardware e principalmente de programação existente entre o Arduíno Uno e o STM32, servindo como verdadeiro guia de uso deste poderoso microcontrolador no ambiente Arduíno.

6. REFERÊNCIAS

<https://capsistema.com.br/index.php/2020/12/12/introducao-ao-stm32-circuito-azul-usando-arduino-ide-led-piscando/>

https://www.mikrocontroller.net/attachment/28831/siemens_AP2921.pdf