Esse artigo foi escrito pelo engenheiro Ismael Lopes da Silva, exclusivamente para o site "www.embarcados.com.br". O link para o artigo é "https://www.embarcados.com.br/blue-pill-entradas-e-saidas-digitais-gpios/".

No sexto artigo da série "A Blue Pill", daremos continuidade no estudo. A dica é acompanhar a série desde o primeiro artigo porque é um caminho estruturado, onde a sequência traz benefícios no aprendizado (linha de raciocínio). Nesse artigo iniciaremos sobre as entradas e saídas de propósito geral GPIO, e vamos testar nosso primeiro programa.

**Entrada e Saída para Proposito Geral (GPIO)**

Nesee artigo usaremos a biblioteca libopencm3 para criar o código fonte de outro programa que também pisca o LED PC13. Este programa demonstrará a configuração e uso da GPIO. Esse programa foi levemente modificado, e é chamado miniblink. Foi modificado para ter o tempo do pisca diferente para distinguir do programa pisca LED original que vem na PCB Blue Pill. Após criar e executar este pequeno programa, veremos a API GPIO (Interface de programação de aplicativos) fornecida pela biblioteca libopencm3.

**O programa miniblink**

Mude para o subdiretório miniblink, conforme mostrado a seguir:

```
~$ cd stm32f103c8t6/miniblink/
```

Para ter certeza que o projeto miniblink será construido (build), então, execute o seguir comando para forçar-nos a construir (buid) o miniblink.

```
~/stm32f103c8t6/miniblink$ make clobber

rm -f *.o *.d generated.* miniblink.o miniblink.d
rm -f *.elf *.bin *.hex *.srec *.list *.map
```

Agora vamos dar um build no miniblink, conforme mostrado a seguir:

```
~/stm32f103c8t6/miniblink$ make

arm-none-eabi-gcc -Os -g -std=c99 -mthumb -mcpu=cortex-m3 -msoft-float -mfix-cortex-m3-ldrd -Wextra -Wshadow -Wimplicit-function-declaration -Wredundant-decls -Wmissing-prototypes -Wstrict-prototypes -fno-common -ffunction-sections -fdata-sections -I/home/ismael/stm32f103c8t6/libopencm3/include -I/home/ismael/stm32f103c8t6//rtos/libwwg/include  -MD -Wall -Wundef -DSTM32F1 -I/home/ismael/stm32f103c8t6/libopencm3/include -I/home/ismael/stm32f103c8t6//rtos/libwwg/include  -o miniblink.o -c miniblink.c
arm-none-eabi-gcc --static -nostartfiles -T/home/ismael/stm32f103c8t6//stm32f103c8t6.ld -mthumb -mcpu=cortex-m3 -msoft-float -mfix-cortex-m3-ldrd -Wl,-Map=miniblink.map -Wl,--gc-sections  miniblink.o -specs=nosys.specs -Wl,--start-group -lc -lgcc -lnosys -Wl,--end-group -L/home/ismael/stm32f103c8t6//rtos/libwwg -lwwg -L/home/ismael/stm32f103c8t6/libopencm3/lib -lopencm3_stm32f1 -o miniblink.elf
arm-none-eabi-size miniblink.elf
```

```
   text    data     bss     dec     hex filename
    700       0       0     700     2bc miniblink.elf
arm-none-eabi-objcopy -Obinary miniblink.elf miniblink.bin
```

Ao fazer isso, você verá algumas linhas de comando executadas para construir (build) e linkar (link) seu executável chamado miniblink.elf. Para usar o utilitário flash, escrevendo a nova aplicação na memória do MCU, precisamos de um arquivo de imagem. A última etapa do processo de construção (build) mostra como o utilitário específico ARM objcopy  é usado para converter miniblink.elf num arquivo de imagem miniblink.bin.

Antes da última etapa, você pode ver que o comando ARM size mostrou os tamanhos das seções de dados e texto do seu programa. Nosso miniblink consiste apenas em 700 bytes de flash (texto da seção) e não aloca SRAM (dados da seção). Embora o programa miniblink não use a seção de dados (SRAM), há SRAM sendo usada para as pilhas de chamadas.

```
~/stm32f103c8t6/miniblink$ ls -la

total 152
drwxr-xr-x  2   ismael ismael   4096 fev 23 17:24 .
drwxr-xr-x 12  ismael ismael   4096 out 20 18:45 ..
-rw-r--r--  1    ismael ismael    244 out 20 18:45 Makefile
-rwxr-xr-x  1    ismael ismael    700 fev 23 17:24 miniblink.bin
-rw-r--r--  1    ismael ismael   1388 out 20 18:45 miniblink.c
-rw-r--r--  1    ismael ismael   1256 fev 23 17:24 miniblink.d
-rwxr-xr-x  1    ismael ismael 146340 fev 23 17:24 miniblink.elf
-rw-r--r--  1    ismael ismael  36762 fev 23 17:24 miniblink.map
-rw-r--r--  1    ismael ismael   5760 fev 23 17:24 miniblink.o
```

**Escrevendo na memória flash o programa miniblink**

Usando novamente o framework make, agora escreveremos o miniblink na memório do MCU usando o arquivo de imagem. Conecte seu programador ST-Link V2 no Desktop/Notebook e verifique os jumpers na Blue Pill. Antes de escrever o novo programa, vamos apagar o programa atual do MCU, portanto:

```
~/stm32f103c8t6/miniblink$ st-flash erase

st-flash 1.5.1-45-g393e942
2020-02-23T19:37:48 INFO usb.c: -- exit_dfu_mode
2020-02-23T19:37:48 INFO common.c: Loading device parameters....
2020-02-23T19:37:48 INFO common.c: Device connected is: F1 Medium-density device, id
0x20036410
2020-02-23T19:37:48 INFO common.c: SRAM size: 0x5000 bytes (20 KiB), Flash: 0x10000 bytes (64
KiB) in pages of 1024 bytes
Mass erasing
```

Observe que o LED apagou. Agora vamos escrever o programa miniblink no MCU.

```
~/stm32f103c8t6/miniblink$ make flash
```

```
/usr/local/bin/st-flash  write miniblink.bin 0x8000000
st-flash 1.5.1-45-g393e942
2020-02-23T19:38:07 INFO common.c: Loading device parameters....
2020-02-23T19:38:07 INFO common.c: Device connected is: F1 Medium-density device, id
0x20036410
2020-02-23T19:38:07 INFO common.c: SRAM size: 0x5000 bytes (20 KiB), Flash: 0x10000 bytes (64
KiB) in pages of 1024 bytes
2020-02-23T19:38:07 INFO common.c: Attempting to write 700 (0x2bc) bytes to stm32 address:
134217728 (0x8000000)
Flash page at addr: 0x08000000 erased
2020-02-23T19:38:07 INFO common.c: Finished erasing 1 pages of 1024 (0x400) bytes
2020-02-23T19:38:07 INFO common.c: Starting Flash write for VL/F0/F3/F1_XL core id
2020-02-23T19:38:07 INFO flash_loader.c: Successfully loaded flash loader in sram
  1/1 pages written
2020-02-23T19:38:07 INFO common.c: Starting verification of write complete
2020-02-23T19:38:07 INFO common.c: Flash written and verified! jolly good!
```

Feito isso, sua Blue Pill deve resetar automaticamente e iniciar o programa miniblink piscando o LED PC13. Com as constantes de tempo modificadas, o LED dever piscar com um ciclo de trabalho 70/30, portanto, 70% aceso e 30% apagado. Esse programa não usa o clock da CPU controlado por um cristal oscilador. Ele usa o clock RC interno. Por esse motivo, um Blue Pill poderá piscar um pouco mais rápido ou mais lento que outro.

**O código fonte do programa miniblink**

Vamos agora examinar o código fonte do programa miniblink que você acabou de executar. Se ainda não estiver em o subdiretório miniblink, mude para lá agora:

```
~$ cd stm32f103c8t6/miniblink
```

Nesse subdiretório, você deve encontrar o arquivo-fonte do programa miniblink.c.

```
#include <libopencm3/stm32/rcc.h>
#include <libopencm3/stm32/gpio.h>
static void
gpio_setup(void)
{
        /* Enable GPIOC clock. */
        rcc_periph_clock_enable(RCC_GPIOC);
        /* Set GPIO8 (in GPIO port C) to 'output push-pull'. */
        gpio_set_mode(GPIOC, GPIO_MODE_OUTPUT_2_MHZ,
         GPIO_CNF_OUTPUT_PUSHPULL, GPIO13);
}

int
main(void)
{
        int i;
```

```
        gpio_setup();
        for (;;)
        {
                gpio_clear(GPIOC, GPIO13);              /*LED on */
                for (i = 0; i < 1500000; i++)   /*Wait a bit. */
                __asm__("nop");
                gpio_set(GPIOC, GPIO13);                /*LED off */
                for (i = 0; i < 500000; i++)            /*Wait a bit. */
                __asm__("nop");
        }
        return 0;
}
```

A estrutura do programa é bastante simples. Consiste no seguinte:

- ✔ Uma função principal do programa 'main'. Note que não há argumentos argc ou argv para a função main;
- ✔ Dentro da função main, a função gpio_setup() é chamada para executar alguma inicialização;
- ✔ Também dentro da função main um loop infinito é formado, onde o LED é aceso e apagado. Observe que a instrução 'return 0' nunca é executada e é fornecida apenas para impedir que o compilador obtenha um erro de compilação.

Mesmo com esse programa simples, há muito o que aprender. Esse programa de exemplo é executado na frequência padrão da CPU, pois nada foi configurado. Veremos essas configurações mais adiante.

A figura 1.1 ilustra como o LED PC13, que pisca, está conectado ao MCU na Blue Pill. Podemos ver que para acender o LED a saída do MCU deve ter nível baixo, porque o LED é alimentado diretamente  da fonte de + 3,3 Volts através do resistor R1, limitador de corrente. É por isso que o comentário nessa linha diz que o LED acende, mesmo que a chamada de função seja gpio_clear(). A função gpio_set() para apagar o LED. Essa lógica invertida é usada simplesmente devido essa parte do projeto da PCB Blue Pill.

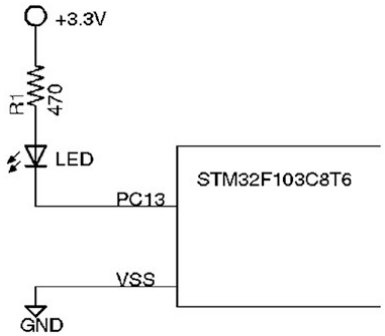


**Figura 1.1** – Porta de saída (GPIO)

Observe novamente essas duas chamadas de funções clear e set:

```
gpio_clear(GPIOC, GPIO13);    /* LED on */
```

.

.
.
```
gpio_set(GPIOC, GPIO13);      /* LED off */
```

Note que essas chamadas usam dois argumentos, que são:

- ✔ O nome da porta GPIO;
- ✔ O número do pino da porta GPIO.

Através da biblioteca libopencm3, você especifica se quer resetar ou setar um bit de uma determinada porta, basta usar as funções gpio_clear() ou gpio_set(). Você também pode alternar um bit de uma determinada porta, usando a função gpio_toggle(). Também, é possível ler e escrever todos os pinos de uma porta, usando as funções gpio_port_read() e gpio_port_write().

Conforme a proposta inicial estamos evoluindo no aprendizado, portanto, aqui concluo o sexto artigo da série. Somente antes de encerrar quero destacar a forma de trabalho, que detalhamos nos primeiros artigos dessa série. Não usamos uma IDE, com suas ferramentas de compilação embutidas. Estamos fazendo as tarefas explicitamente, onde escrevemos o programa em um editor de texto qualquer (gedit ou vim), compilamos esse arquivo-fonte (usando o utilitário make que invoca o compilador), criamos um arquivo de imagem, e escrevemos para a memória flash da Blue Pill. Basicamente estamos mostrando as coisas por trás dos bastidores, aquilo que uma IDE faz automaticamente. É a melhor forma de desenvolvimento? A resposta é “depende do gosto de quem está programando”. Tem softwares e ferramentas que dão muito mais produtividade, porém, com alto nível de abstração. Você escolhe seu caminho, mas, nessa série a proposta está bem definida.