DISCIPLINA EA701

Introdução aos Sistemas Embarcados

ROTEIRO 1: Fundamentos dos Sistemas Embarcados

INTRODUÇÃO AO MICROCONTROLADOR STM32H7A3, À PLACA NUCLEO-H7A3ZI-Q, AO *CORE* CORTEX-M7 E AO AMBIENTE DE DESENVOLVIMENTO STM32CubeIDE

Profs. Antonio A. F. Quevedo e Wu Shin-Ting

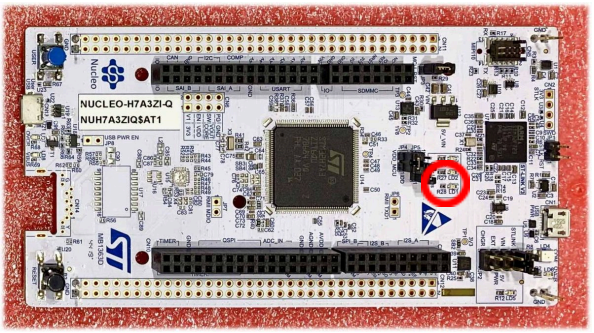
FEEC / UNICAMP

Revisado em janeiro de 2025 por Ting com auxílio do Chatgpt
Revisado em julho de 2024

INTRODUÇÃO

Os sistemas embarcados estão presentes em diversos aspectos da nossa vida cotidiana, desde dispositivos simples como relógios digitais até complexos sistemas automotivos. Para entender esses sistemas, é essencial conhecer os componentes básicos que os compõem e as ferramentas utilizadas para seu desenvolvimento. Esses componentes incluem o núcleo (STM32H7A3ZIT-Q), o microcontrolador (STM32H7A3), a placa de desenvolvimento (NUCLEO-H7A3ZI-Q) e o ambiente de desenvolvimento integrado (STM32CubeIDE).

Antes de explorarmos os detalhes desses componentes, vamos considerar **como fazer o LED verde integrado na placa NUCLEO-H7A3ZI-Q piscar**.



Neste roteiro, apresentaremos **duas abordagens para a personalização de microcontroladores**, cada uma delas operando em um nível de abstração diferente: **a manipulação direta dos *bits* dos registradores e a programação de instruções em *assembly* que controlam esses *bits***. Essas

abordagens destacam a diferença entre a execução manual repetitiva e a automatização de processos, permitindo que o processador execute tarefas. Em seguida, forneceremos uma visão geral do papel de cada componente mencionado na implementação dessas duas abordagens.

PROJETOS-EXEMPLO

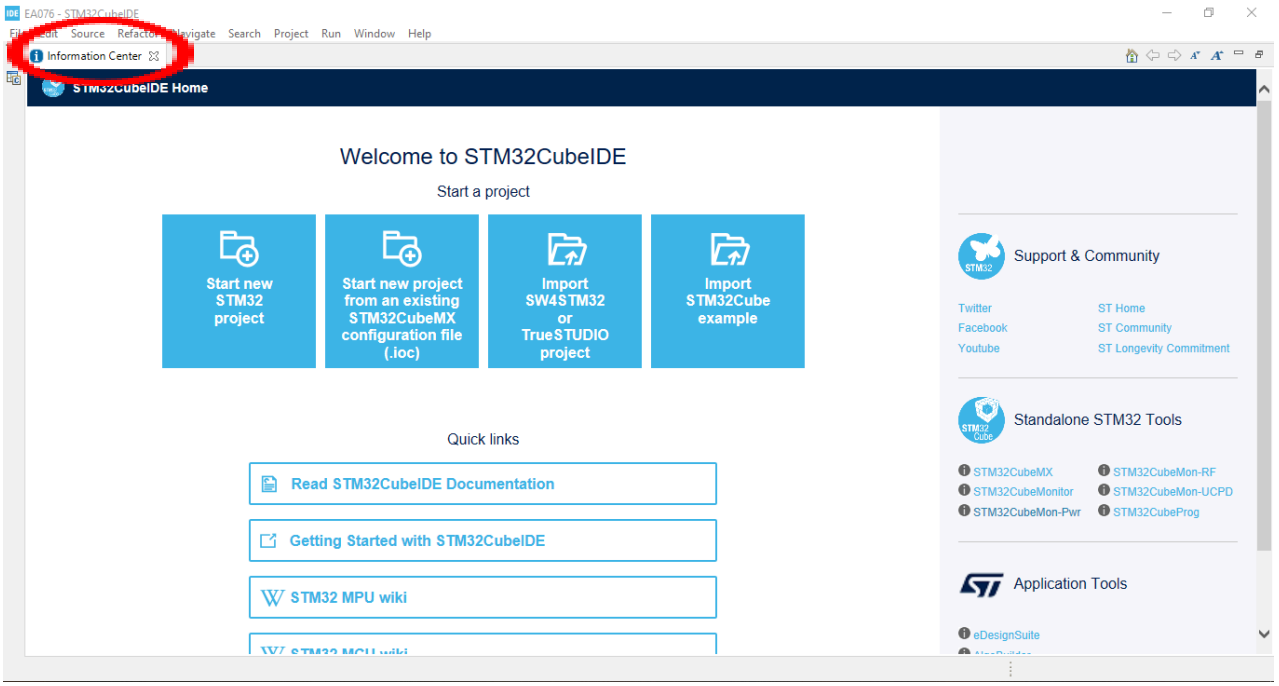
A personalização de microcontroladores para tarefas específicas é uma prática fundamental no desenvolvimento de sistemas embarcados. Essa abordagem permite adaptar o *hardware* às necessidades de uma aplicação, configurando diretamente os registradores que controlam periféricos, modos de operação e fluxos de dados. O processo pode ser realizado em dois níveis principais de abstração: manual e programático. No **nível manual**, a configuração é feita diretamente por meio de um depurador, envolvendo a manipulação explícita dos *bits* nos registradores de controle, configuração e dados. Utilizando ferramentas como interfaces JTAG ou SWD, é possível acessar os registradores do microcontrolador em tempo real e ajustar manualmente os valores de *bits*, definindo o comportamento do sistema de forma interativa. Já no **nível programático**, a personalização é realizada por meio de código, utilizando instruções em linguagem de máquina (*assembly*) ou linguagens de alto nível, como C, para modificar os valores dos registradores. Essa abordagem permite automatizar a configuração, integrando-a ao fluxo de execução do *software* e garantindo maior reprodutibilidade e eficiência no desenvolvimento.

Criando um novo projeto

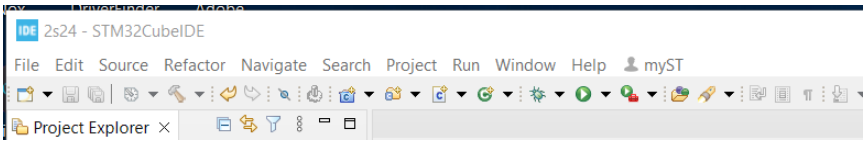
Você já imaginou personalizar um microcontrolador para executar uma tarefa específica, controlando diretamente seus registradores? Como será que conseguimos acessar esses registradores, manipular seus bits e observar os efeitos em um dispositivo físico? Hoje, contamos com ambientes de desenvolvimento poderosos, equipados com ferramentas de depuração integradas, que permitem explorar o *hardware* e o *software* de forma interativa. Que tal seguirmos juntos, passo-a-passo, na criação do seu primeiro projeto no ambiente STM32CubeIDE e descobrir como controlar o estado do LED verde da placa de desenvolvimento NUCLEO-H7A3ZI-Q manualmente?

1. Após iniciar a execução do aplicativo STM32CubeIDE instalado nos computadores do LE-30, será perguntado o diretório para o *workspace* que deseja utilizar. Há ainda a opção de usar sempre este *workspace* sem precisar perguntar a cada inicialização. Cada aluno deve ter seu próprio *workspace*, preferencialmente identificado com o número de RA.

2. Na primeira vez em que se abre um *workspace* novo, a janela que abre é dominada por uma aba intitulada “Information Center”. Basta clicar no “x” ao lado do título para fechar essa aba e ter o ambiente de trabalho pronto para uso.



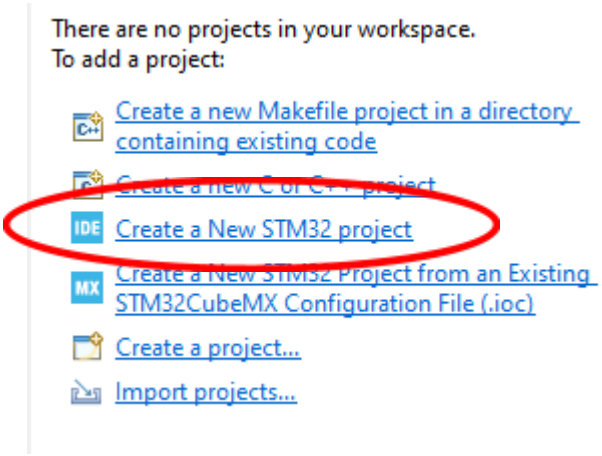
3. O IDE vai abrir na **perspectiva de programação**. Inicialmente no painel à esquerda aparece o painel do "Project Explorer", com várias opções de uso frequente.



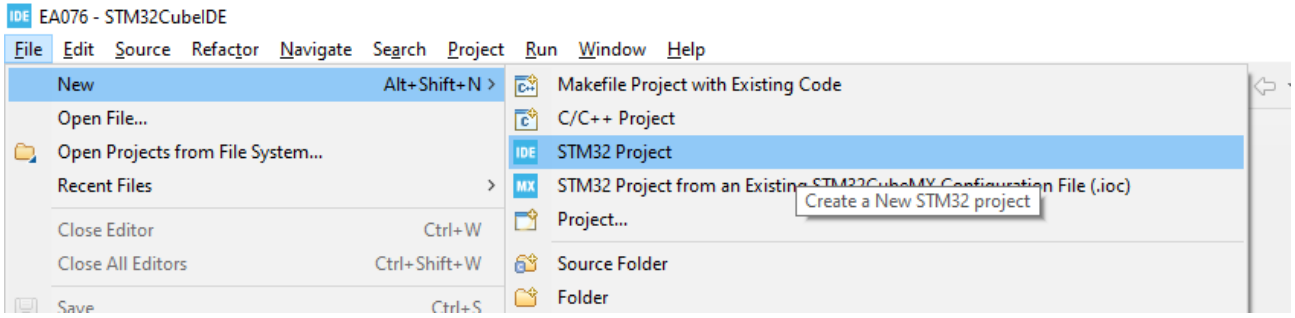
ATENÇÃO: É fundamental que o IDE esteja logado em uma conta MyST antes das próximas etapas. Caso se tente baixar os pacotes de suporte ao microcontrolador antes do login, o IDE não será capaz de realizar o *download*, e não será mais possível logar na conta, sendo necessário reinstalar o IDE.

Para se saber se o IDE está logado, basta ver a opção mais à direita da barra de ferramentas. Se ela for "MyST", o login não foi realizado. Se ela for "Hello" seguida de algum nome, o login está feito. Existe uma conta FEEC para ser usada nos computadores do LE-30. Caso o IDE não esteja logado, fale com o professor.

4. Vamos criar o primeiro projeto. Para isto, podemos usar a opção no "Project Explorer" chamada "Create a New STM32 Project".

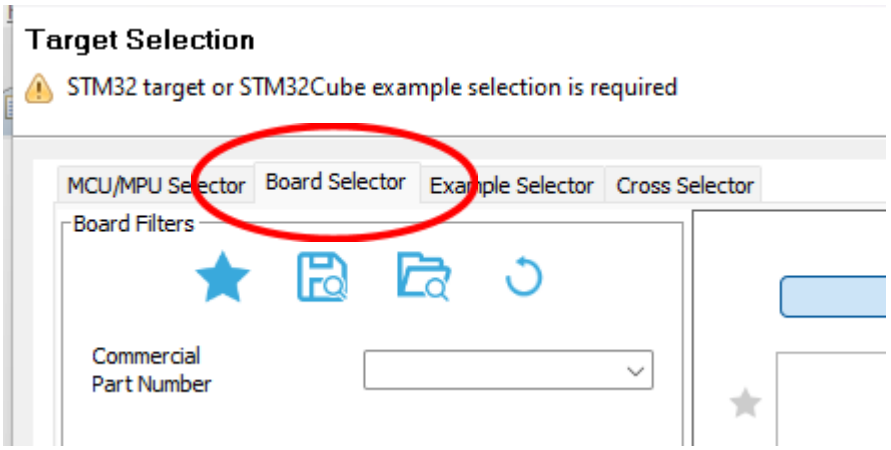


Alternativamente, no menu superior selecione *File – New – STM32 Project*.



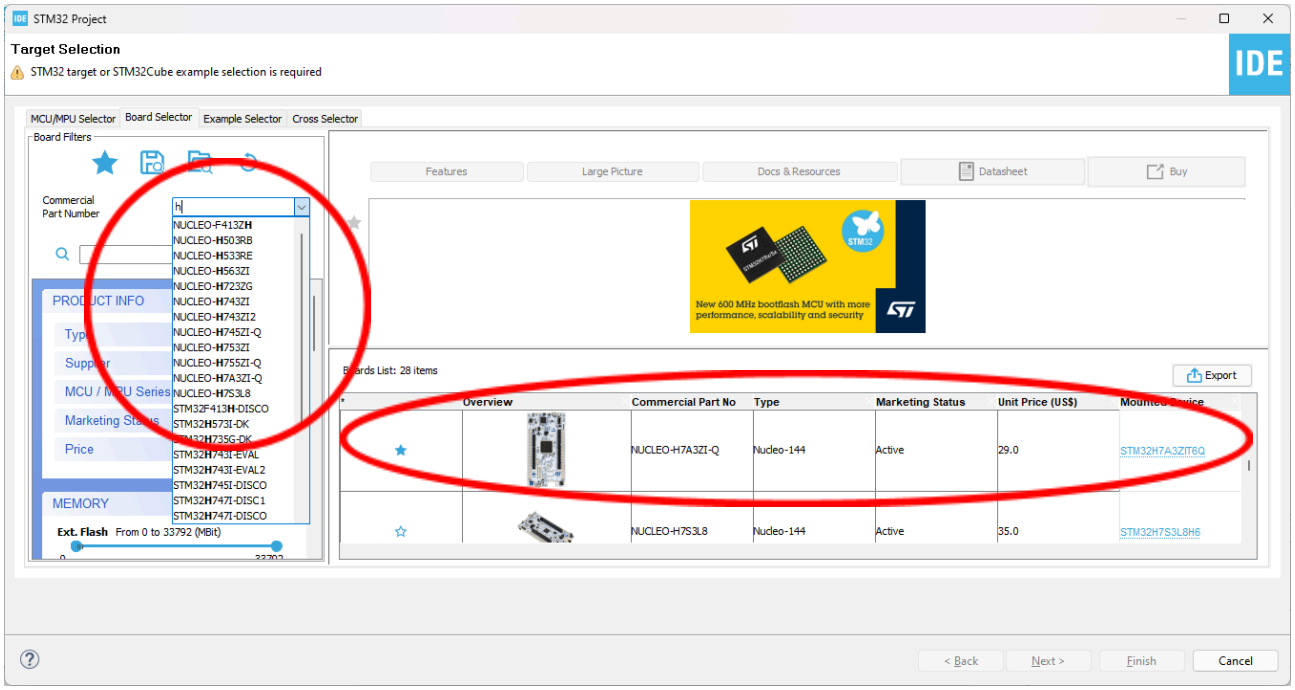
5. Após alguns segundos, abre uma nova janela chamada "*Target Selection*". Nela vamos definir o modelo de microcontrolador que vamos utilizar. À esquerda temos uma série de filtros que podemos usar, para seleção do núcleo de processador ARM, série, linha, tipo de encapsulamento, etc. Entretanto, a forma mais fácil de selecionar o dispositivo é buscando por texto, no campo "*Commercial Part Number*".

Nesta disciplina estamos usando uma placa de desenvolvimento desenvolvida pela ST (fabricante do microcontrolador e desenvolvedora do IDE) denominada NUCLEO-H7A3ZI-Q. Ao invés de selecionar o microcontrolador-alvo ("*MCU/MPU Selector*"), vamos selecionar a placa de desenvolvimento ("*Board Selector*"), que já possui alguns periféricos. No canto superior esquerdo, clique na aba "*Board Selector*".

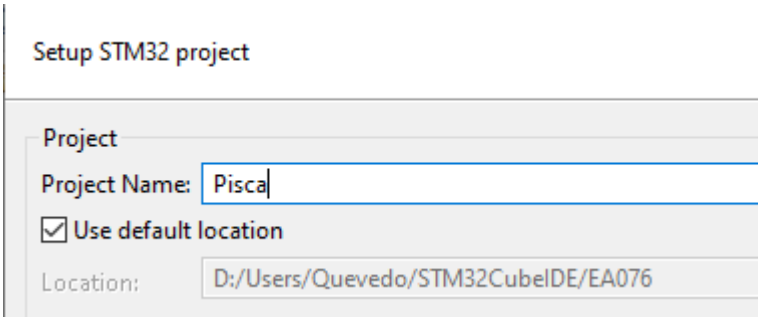


No campo “*Commercial Part Number*”, digite “h”, e verá as opções de placa à direita. Na tabela à direita, ou na *drop-down*, que aparece no campo “*Commercial Part Number*”. Selecione a placa NUCLEO-H7A3ZI-Q.

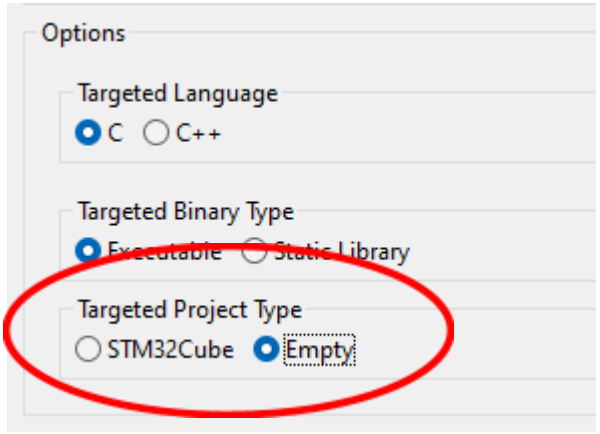
À esquerda do nome, há uma estrela com o interior branco. Ao clicar nela, você inclui esta placa de desenvolvimento na sua lista de favoritos (a estrela fica azul). Isso permite que você vá diretamente à lista de favoritos nos projetos futuros, clicando na estrela acima do campo “*Commercial Part Number*” e na linha da placa. O botão “*Next*” é então ativado.



6. Ao clicar no botão “*Next*”, aparece a janela na qual se dá nome ao projeto. Digite “Pisca” no campo de “*Project Name*”

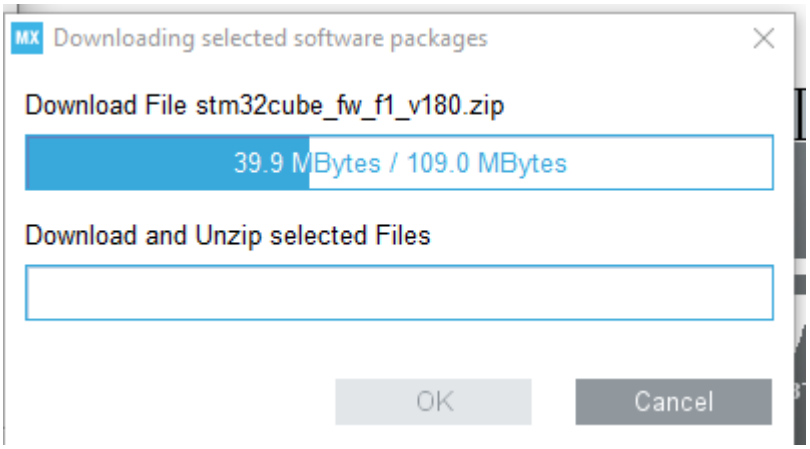


Na mesma janela, no campo “Targeted Project Type”, mude a opção para “Empty”. Com isso, o projeto será criado sem a adição de bibliotecas de suporte.

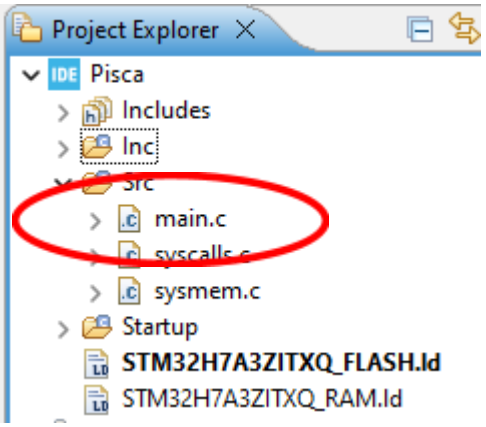


Por fim, clique no botão "*Finish*" para que o projeto seja criado.

Na primeira vez que uma determinada família de microcontroladores é usada, o IDE baixa os pacotes de suporte àquela família. Isso pode durar algumas dezenas de minutos. Nos computadores da sala de aula, o suporte à família H7 já deve ter sido instalado.



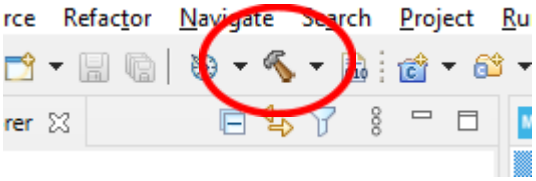
7. Após a instalação dos pacotes de suporte, o IDE entra na perspectiva de Programação. À esquerda, temos o painel "Project Explorer", no qual podemos ver a estrutura de arquivos do projeto. Clique na marca ao lado da pasta "Src" para expandir a pasta e ver os arquivos de código-fonte em C. Na expansão da pasta, pode-se ver o arquivo "main.c". Dê um duplo-clique nele para abri-lo no editor de texto interno do IDE.



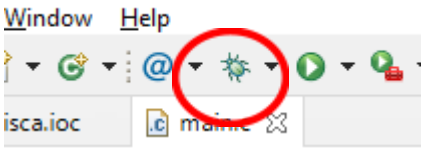
8. Agora vamos escrever a função “main()” em C com uma instrução, como declara/aloca uma variável “counter” na memória e incrementa esta variável em 1 no laço de iteração infinita. Usamos este programa apenas para acessarmos os registradores do microcontrolador na perspectiva de depuração e configurá-los.

```c
#include <stdint.h>

#if !defined(__SOFT_FP__) && defined(__ARM_FP)
  #warning "FPU is not initialized, but the project is compiling for an FPU. Please initialize the FPU before use."
#endif

int main(void)
{
    uint32_t counter=0;

    /* Loop forever */
    for(;;) {
        /* apenas para ter acesso aos registradores pela perspectiva
         * de depuracao
         */
        counter += 1;
    }
}
```

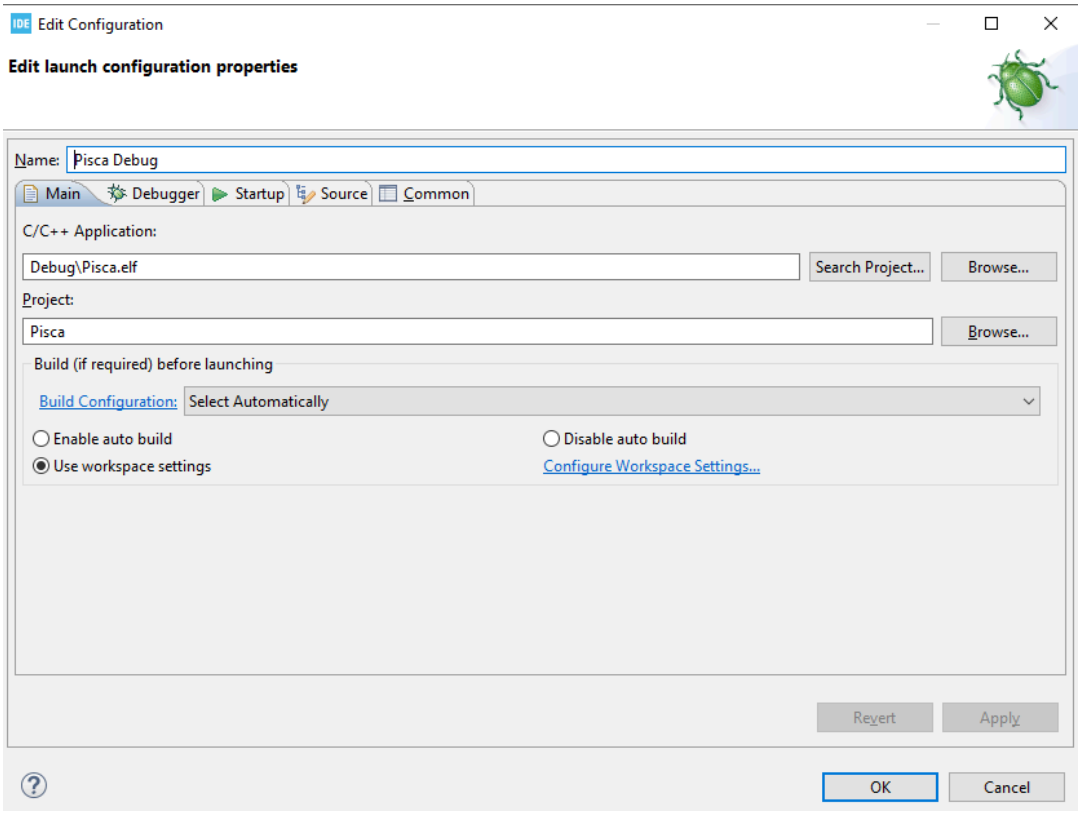
9. Não se esqueça de salvar as modificações do arquivo (Ctrl-S ou o ícone de salvar). Depois use o ícone de “martelo” para fazer o “*Build*”:



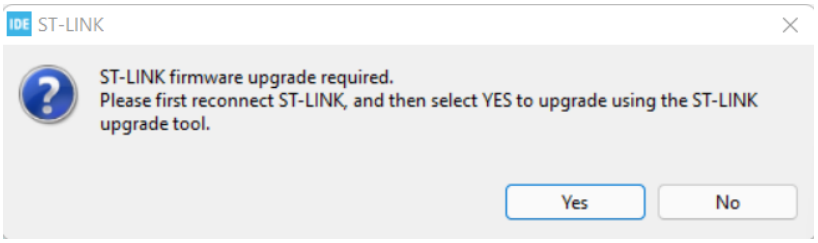
10. Após o “*Build*”, estamos prontos para carregar o programa na placa. Clique no botão com o ícone de um besouro para carregar (antes disso, conecte a placa a uma porta USB do computador) e ao fim da transferência é chaveado para a perspectiva de Depuração (*Debug*) automaticamente.



Na primeira vez que o “*Debug*” é chamado em um projeto novo, abre-se a janela “*Edit launch configuration properties*” com uma série de configurações para a depuração. Vamos manter todas as opções padrão, então basta clicar no botão “OK”. Isto só é necessário na primeira transferência/carga de cada projeto para execução no modo “*Debug*”. Depois disso, o IDE solicita a mudança de perspectiva. Clique no botão “*Switch*” (também com a opção de lembrar a decisão).

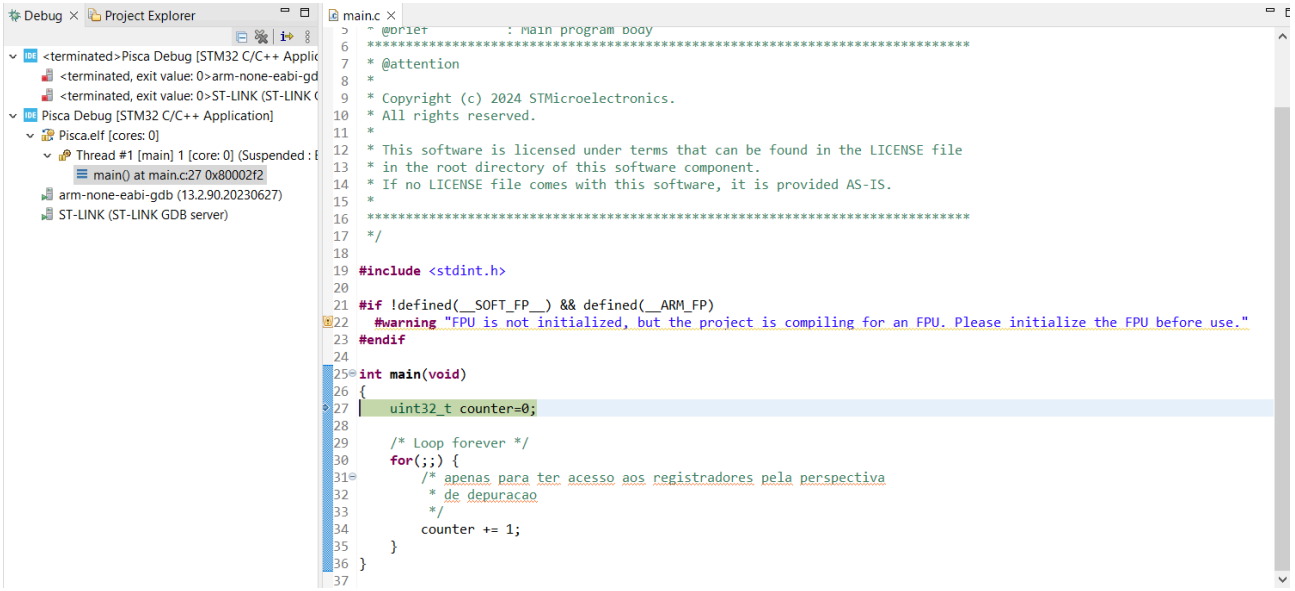


OBS: Quando há uma atualização do *firmware* do ST-LINK, o IDE não fará a carga do programa. Em vez disso, aparecerá a seguinte janela, pedindo para atualizar o *firmware*:

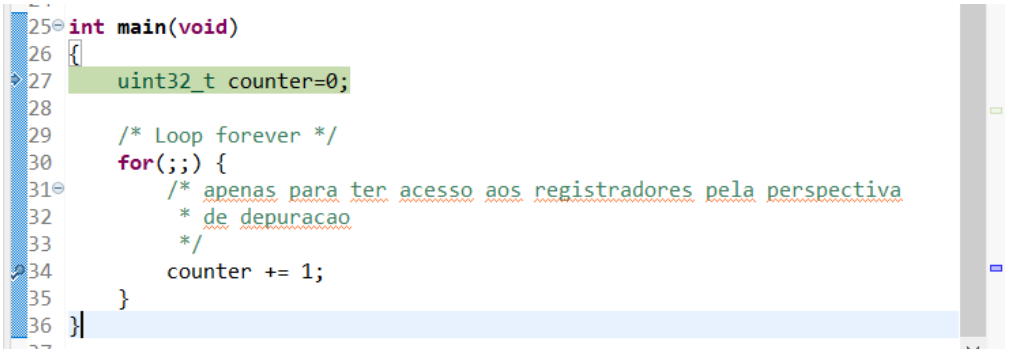


Neste caso, execute os procedimentos da seção "**ATUALIZANDO O *FIRMWARE* DO ST-LINK**", ao final desta mesma apostila.

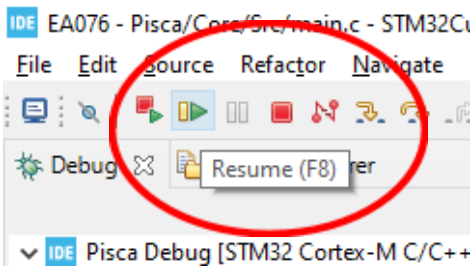
11. Agora vemos na janela esquerda ("*Debug*"), que a MCU está parada no início da função "main()" (destacado na janela). Ao lado desta janela, aparece o código-fonte do arquivo "main.c".



12. Adicione um *breakpoint* no código, para que o fluxo de controle seja interrompido toda vez que sua execução chegar a este ponto. Nessas paradas podemos modificar o conteúdo dos registradores.



13. Nesta mesma janela, clique no triângulo verde (Resume), ou pressione F8, para iniciar a execução.

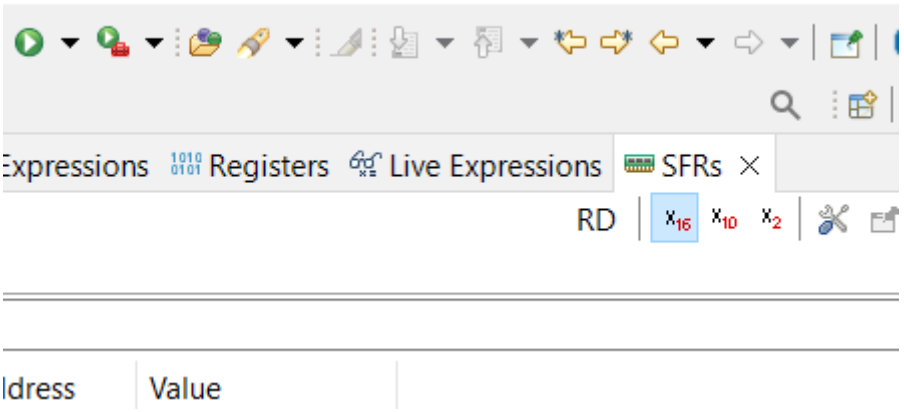


Junto ao botão de "*Resume*", há ainda os botões "*Terminaate and Relaunch*" (quadrado vermelho e triângulo verde), "*Suspend*" (símbolo de pausa), "*Terminate*" (quadrado vermelho) e "*Disconnect*" (uma linha em forma de "N"). Estes botões permitem, respectivamente, terminar e reiniciar a execução, pausar a execução (aguardando o "Resume"), interromper definitivamente a execução, até que se faça um "Reset", e continuar executando o programa sem a conexão ao IDE.

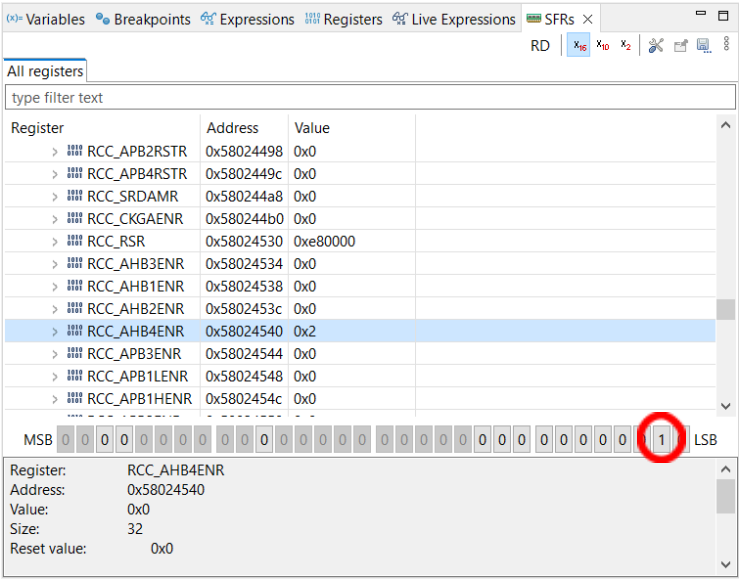
14. No STM32CubeIDE, o acesso completo aos componentes mapeados no espaço de endereçamento do processador está disponível na perspectiva de depuração. Vamos usar essa facilidade para modificar os *bits* dos registradores manualmente e ver os efeitos sobre o LED verde. Ao lado direito da perspectiva de depuração, há um conjunto de abas que permite a visualização de

variáveis e regiões de memória, bem como os registradores da CPU e registradores dos periféricos. Ao parar no ponto de parada, abra a aba SFRs (*Special Function Registers*) dos periféricos, procure o módulo RCC e expanda o seu conteúdo.

Obs: Caso a aba SFRs não esteja disponível, ative-a através do menu *Windows* > *Show View* > *SFRs*.

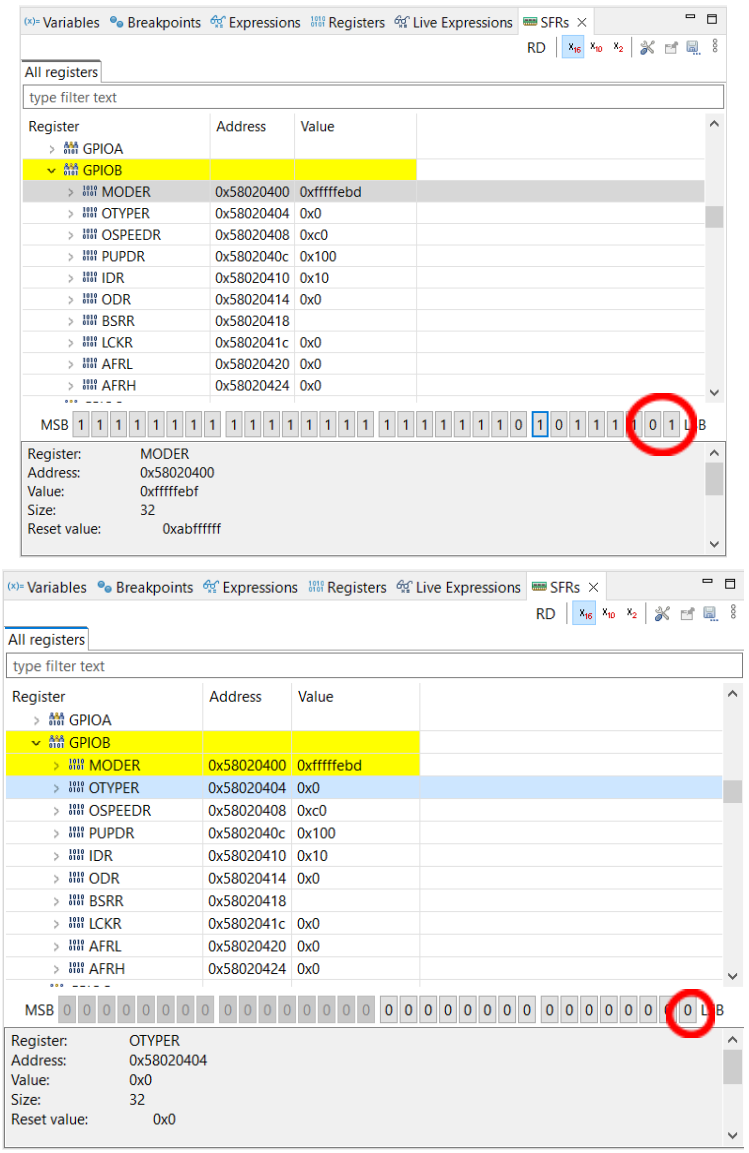


Localize o registrador RCC_AHB4ENR. Coloque o *bit* "1" desse registrador em "1" clicando o *bit* na linha de quadradinhos. Alternativamente, pode-se modificar o conteúdo do registrador escrevendo diretamente "0x2" no campo "Value" da linha destacada em azul e dando um "ENTER".

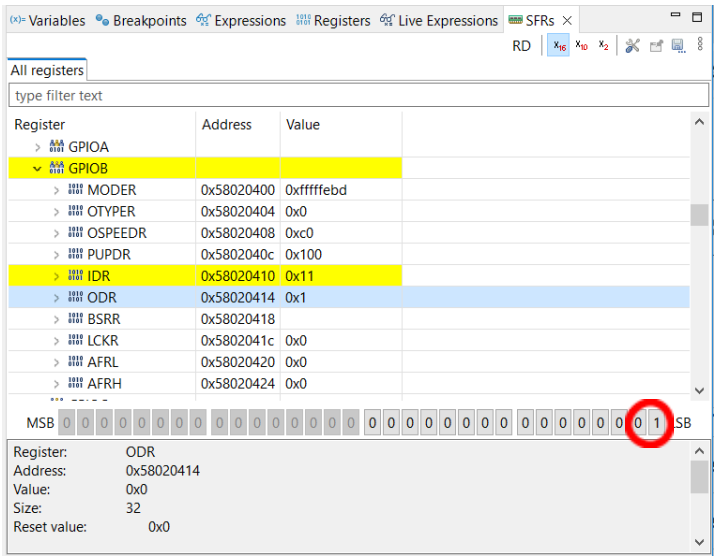


Observe que cada registrador RCC_* no módulo RCC possui um endereço de memória específico, conforme listado na coluna "Address". Todos os endereços são múltiplos de 4.

15. Procure o módulo GPIOB e expanda o seu conteúdo. De forma análoga, cada registrador GPIOB_* no módulo GPIOB tem um endereço específico. Ajuste os *bits* dos registradores GPIOB_MODER e GPIOB_OTYPER conforme as ilustrações abaixo.



16. Com os registradores configurados para controlar o pino PTB0 como saída digital, o LED verde pode ser controlado manualmente, alternando o nível lógico do *bit* ‘0’ do registrador GPIOB_ODR entre ‘0’ e ‘1’.



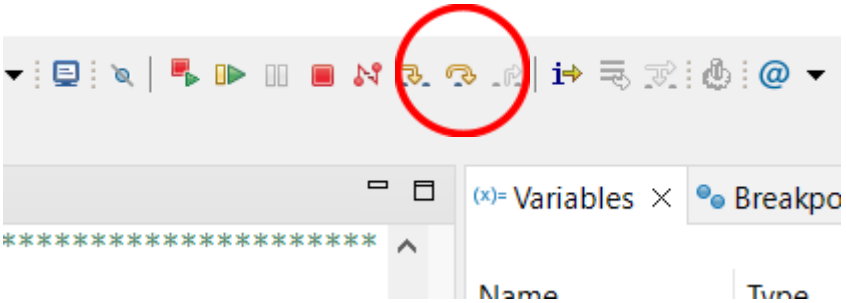
O que acontecerá se alterarmos manualmente o *bit* 0 ou o *bit* 16 do registrador GPIOB_BSRR? Por quê? Neste tópico, vamos entender a função dos campos de *bits* de uma série de registradores integrados em um microcontolador, como os *bits* do registrador GPIOB_BSRR,

Dissecando um projeto

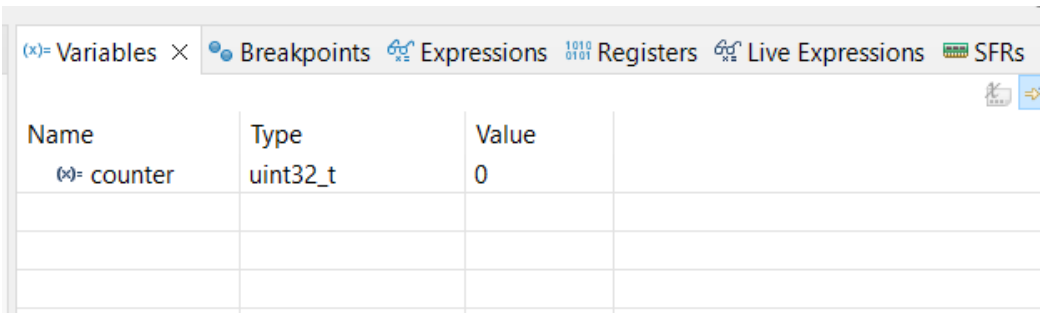
Agora que temos nosso primeiro projeto em mãos, que tal explorarmos uma das ferramentas mais poderosas do desenvolvimento embarcado: a depuração (em inglês, *debugging*)? No STM32CubeIDE, os recursos de depuração nos permitem tomar controle total do microcontrolador, analisando o código passo-a-passo, inspecionando variáveis e identificando erros que podem impedir o funcionamento correto do programa. Pense nas ferramentas de depuração do IDE como um raio X do *hardware* e *software*, revelando o que acontece dentro do microcontrolador em tempo real. Assim como médicos diagnosticam doenças para restaurar o funcionamento do corpo, nós podemos diagnosticar falhas no código e garantir que o sistema funcione como esperado. Vamos juntos desvendar esse processo e aprender como depurar um projeto de forma eficiente?

1. Agora reinicie o programa. Para isso, pode-se sair da perspectiva de *Debug* (com o botão *"Terminate"* (Ícone quadrado vermelho)) e reiniciar um *debug*, ou simplesmente com o botão "*Terminate and Relaunch"* (Ícone quadrado vermelho e seta verde).

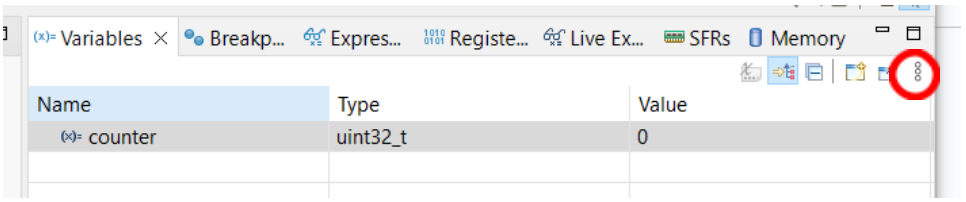
2. Use a tecla F6 ou o botão *Step Over* para avançar linha a linha (a linha a ser executada apresenta uma seta azul ao seu lado esquerdo),



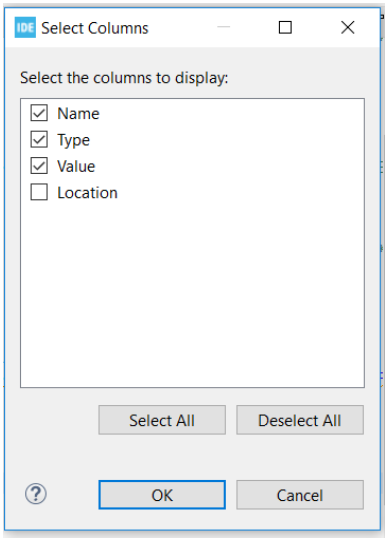
e veja na aba "Variables" como o valor da variável "counter" vai sendo modificado a cada iteração.

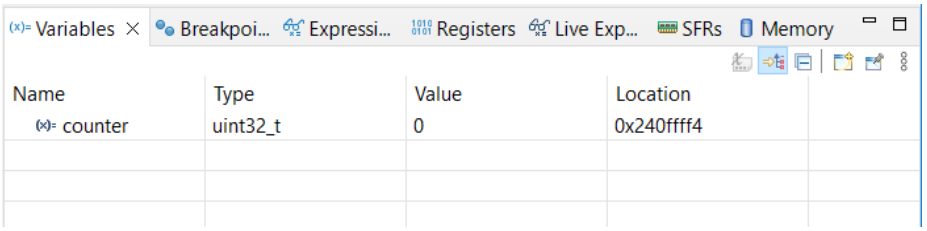


É possível ver o endereço em que uma variável é alocada. Para isso, clique nos 3 pontinhos no canto superior direito e aparecerão duas opções: “Number Format” e “Layout”. O primeiro é para configurar o formato de renderização das variáveis e o segundo, para configurar o *layout* da aba “Variables”, incluindo “Select Columns” para especificar os dados a serem mostrados na tabela.



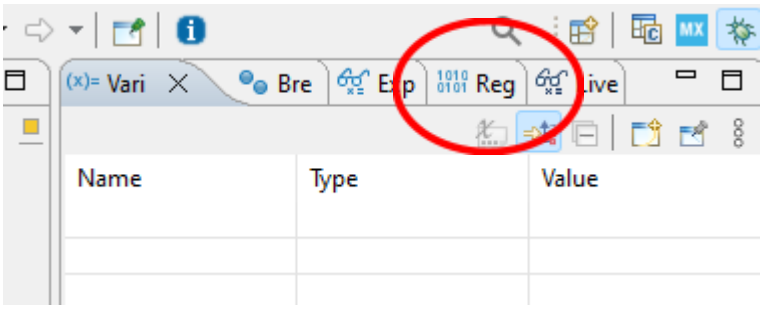
Ative a coluna “Location” para ver os endereços de memória das variáveis.



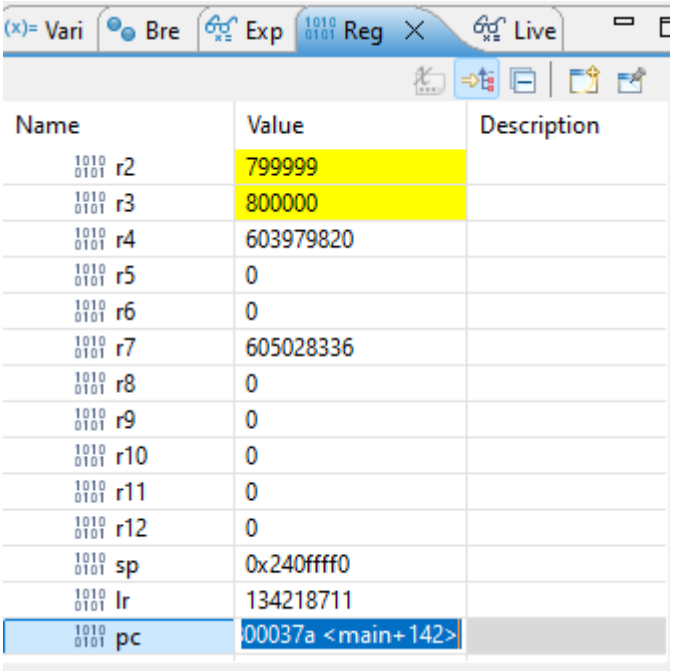


Consultando a Tabela do Manual de Referência, você saberia dizer em qual unidade de memória é armazenado o valor da variável “counter”? Neste tópico, vamos aprender o mapeamento dos componentes de memória no espaço de endereçamento do microcontrolador integrado na placa de desenvolvimento que usaremos. Assim, você será capaz de identificar o componente de memória específico onde cada variável é armazenada.

3. Veja na aba “Registers” de registradores da CPU. Clique na aba “Registers” para ver os registradores do núcleo.



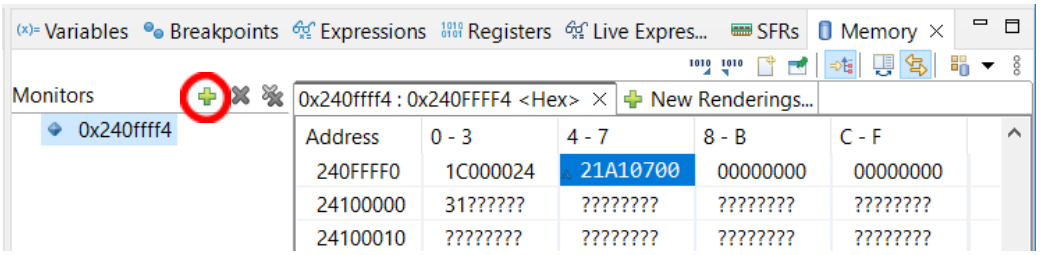
Desça a lista até que possa visualizar os registradores SP (do inglês *Stack Pointer*), LR (do inglês *Link Register*) e PC (do inglês *Program Counter*). Use o botão "Resume" novamente para executar o programa até cada *breakpoint* e veja como alguns dos registradores mudam, especialmente o PC. As mudanças (exceto no PC) aparecem com fundo amarelo.



Reinicie o programa e veja o valor do SP.

4. Dê um "Resume" e na primeira pausa, vá para a aba "Variables" e altere o valor da variável "counter" na coluna "Value" para 500000, dê um "Enter" seguido de "Resume".

5. Vá até a aba "Memory" e adicione o endereço de "counter" mostrado na coluna "Location" da aba "Variables". Para fazer isso, clique no botão "+" destacado e insira o endereço no campo disponível na janela *pop-up* que será exibido.



Obs: Caso a aba Memory Monitor não esteja disponível, ative-a através do menu *Windows > Show View > Memory Monitor.*

Um novo projeto de programação (em *assembly*)

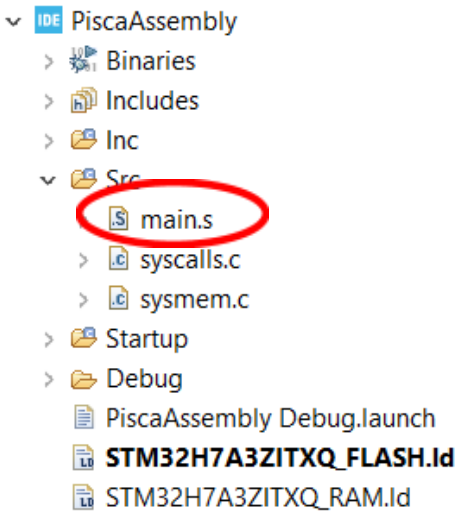
Se a ideia de manipular manualmente os *bits* dos milhares de registradores do microcontrolador STM32H7A3 ao longo do semestre parece desafiadora e tediosa, você não está sozinh@! Algumas pessoas podem se perguntar: "Existe uma maneira mais eficiente de configurar o

microcontrolador?" A resposta é sim! Vamos dar o primeiro passo rumo à automação desse processo, programando a configuração dos registradores diretamente em linguagem *assembly*.

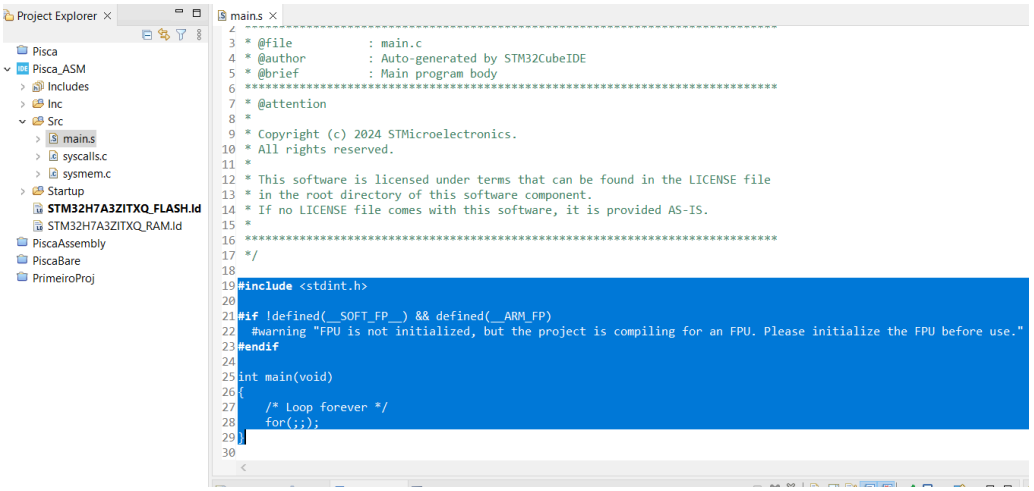
Para isso, é fundamental compreender os fundamentos dos sistemas embarcados, a arquitetura e organização do microcontrolador e, claro, saber se comunicar com o processador na linguagem que ele entende. Este curso tem como objetivo iniciar sua capacitação para o desenvolvimento profissional de sistemas embarcados, permitindo que você domine cada etapa, desde a manipulação de *bits* até o controle avançado do hardware.

Vamos juntos explorar essa abordagem e dar mais um passo rumo ao domínio dos microcontroladores?

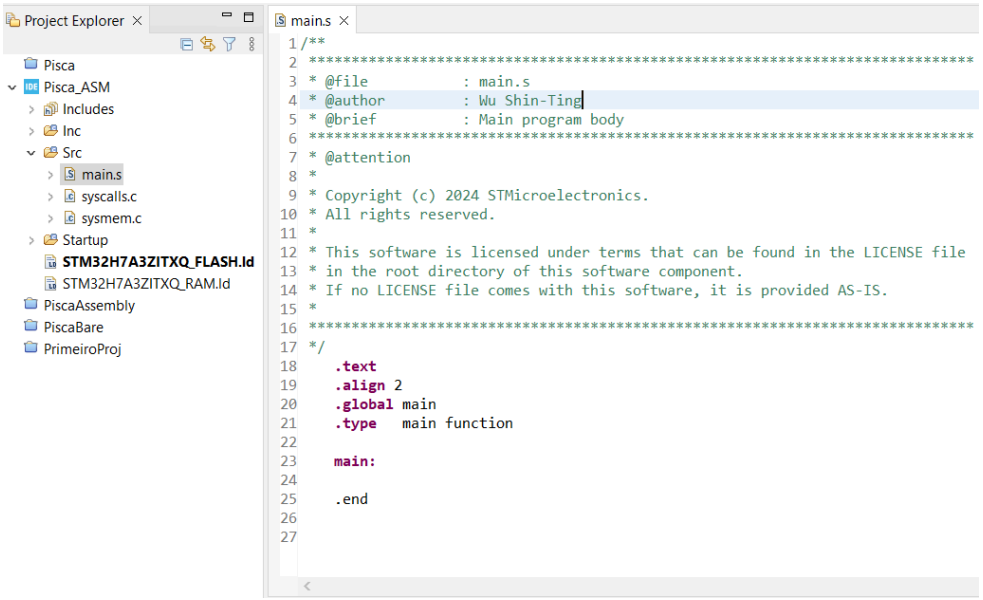
1. Vamos criar um programa em *assembly*, também conhecida como linguagem de montagem, que seta os *bits* dos registradores de configuração e alterna o *bit* "0" do registrador GPIOB_ODR automaticamente. Para isso é necessário consultarmos o repertório de instruções do Cortex-M7 no Manual de Programação e usá-las no programa.
2. Vamos criar um novo projeto seguindo os passos 1 – 6 do projeto anterior, dando o nome de "Pisca_ASM" para este projeto.
3. Renomeie arquivo"main.c" da pasta Src para "main.s".



4. Substitua em main.s a função "main ()" de C



pela função "main:" de *assembly*.



5. Para podermos alterar o conteúdo dos registradores RCC_AHB4ENR, GPIOB_MODER, GPIOB_OTYPER e GPIOB_ODR, usamos a diretiva ".word" para reservar espaços de memória rotulados, alinhados aos endereços múltiplos de 4 (diretiva ".align"), no segmento de instruções (.text) e inicializá-lo com os endereços dos registradores.

```
18    .text
19    .align 2
20    .global main
21    .type   main function
22
23    main:
24
25    .align 4
26RCC_AHB4ENR:
27    .word   0x58024540
28GPIOB_MODER:
29    .word   0x58020400
30GPIOB_OTYPER:
31    .word   0x58020404
32GPIOB_ODR:
33    .word   0x58020414
34
35    .end
```

6. Configuramos agora os três registradores RCC_AHB4ENR, GPIOB_MODER e GPIOB_OTYPER, usando somente dois registradores R2 e R3, lembrando que a arquitetura de Cortex-M7 é *load-store*, ou seja todas as operações envolvendo dados são entre os registradores. Carregamos primeiro o conteúdo de um registrador no R2, atualizamos o valor com uma operação de *bit*, e sobrescrevemos com o valor atualizado o conteúdo do registrador, cujo endereço é carregado no R3. Note o uso de pseudo-instruções "=RCC_AHB4ENR", "=GPIOB_MODER" e "=GPIOB_OTYPER". No contexto da linguagem Assembly para processadores ARM, a expressão "ldr r3, =X" é uma forma de carregar no R3 um valor imediato se X é um valor, ou o endereço de um valor se X é um rótulo.

```
# Configurar RCC_AHB4ENR
    ldr     r3, =RCC_AHB4ENR
    ldr     r3, [r3, #0]
    ldr     r2, [r3, #0]
    mov     r3, #2
    orr     r2, r3
    ldr     r3, =RCC_AHB4ENR
    ldr     r3, [r3, #0]
    str     r2, [r3, #0]
```

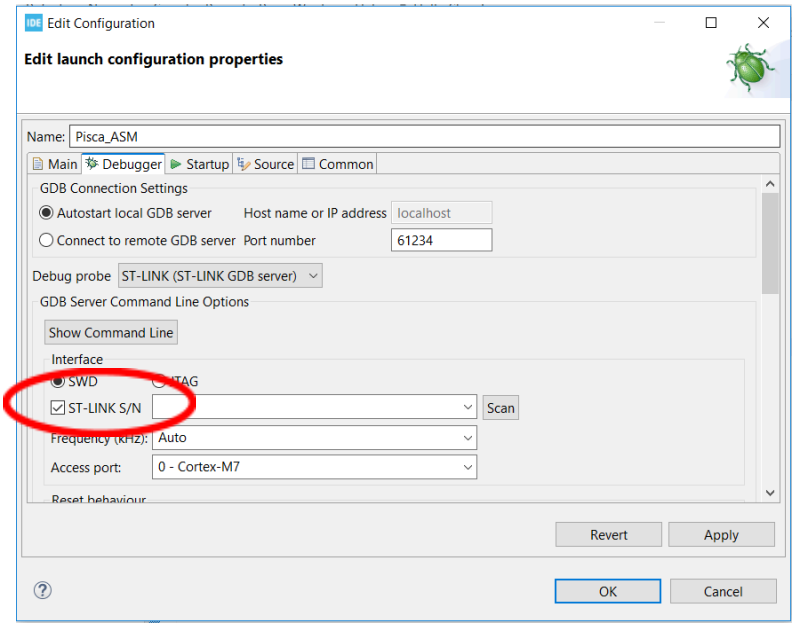
```
# Configurar GPIOB_MODER
    ldr     r3, =GPIOB_MODER
    ldr     r3, [r3, #0]
    ldr     r2, [r3, #0]
    mov     r3, #2
    bic     r2, r3
    mov     r3, #1
    orr     r2, r3
    ldr     r3, =GPIOB_MODER
    ldr     r3, [r3, #0]
    str     r2, [r3, #0]
```

```
# Configurar GPIOB_OTYPER
    ldr     r3, =GPIOB_OTYPER
    ldr     r3, [r3, #0]
    ldr     r2, [r3, #0]
    mov     r3, #1
    bic     r2, r3
    ldr     r3, =GPIOB_OTYPER
    ldr     r3, [r3, #0]
    str     r2, [r3, #0]

```

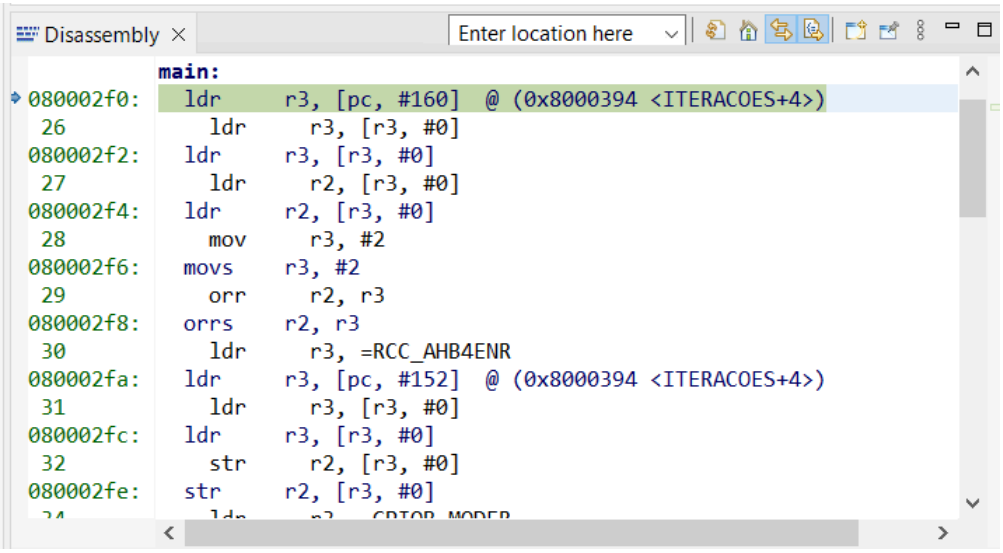
7. Configurados esses registradores, só precisamos alternar o *bit* '0' do registrador GPIOB_ODR para acender o LED (*bit* '0' em '1') e apagar o LED (*bit* '0' em '0').

```
#
# Executar a tarefa
#
LacoInfinito:
# Resetar a saida PTB0 (= 0)
    ldr     r3, =GPIOB_ODR
    ldr     r3, [r3, #0]
    ldr     r2, [r3, #0]
    mov     r3, #1
    bic     r2, r3
    ldr     r3, =GPIOB_ODR
    ldr     r3, [r3, #0]
    str     r2, [r3, #0]
# Setar a saida PTB0 (= 1)
    ldr     r3, =GPIOB_ODR
    ldr     r3, [r3, #0]
    ldr     r2, [r3, #0]
    mov     r3, #1
    orr     r2, r3
    ldr     r3, =GPIOB_ODR
    ldr     r3, [r3, #0]
    str     r2, [r3, #0]
    b   LacoInfinito

```

8. Construa o projeto e transfira-o para o microcontrolador. Abrirá a janela de configuração da transferência. Vá para a aba "Debugger". Como o arquivo gerado a partir do código em *assembly* não é compatível com o processo de linkagem automático utilizado pelo IDE para C, deve-se ativar "ST-LINK S/N" explicitamente. Dê "OK" para transferir o código executável para o microcontrolador. O ST-LINK é uma interface de programação e depuração que conecta o microcontrolador ao IDE..

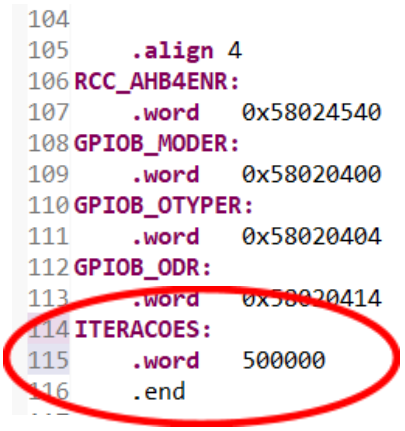


9. Abra a aba “Disassembly” na Perspectiva de Depuração (“Windows” > “Show View” > “Disassembly”) e veja através dos endereços, em verde, mostrados na primeira coluna a unidade de memória em que as instruções são armazenadas.



10. Execute o programa. O que você observou? O LED ficou alternando entre os dois estados? Quando a frequência de alternância dos estados de um LED (ou qualquer fonte de luz) é muito alta, nossa percepção visual pode não acompanhar individualmente cada ciclo de ligado e desligado. A percepção visual da alternância de um LED é influenciada pela persistência retiniana, que é o tempo que uma imagem persiste na retina após sua fonte ter sido removida. Esse fenômeno pode fundir os estados ligado e desligado do LED em uma única percepção visual, especialmente em altas frequências de alternância. A maior frequência para que seja perceptível a alternância de um LED depende da capacidade do sistema visual humano de detectar mudanças rápidas na luz. Frequências acima de aproximadamente 60 a 70 Hz são tipicamente necessárias para que a intermitência não seja percebida. Portanto, para que as piscadas sejam perceptíveis, precisamos reduzir a frequência de alternância, inserindo um “laço de espera” de 500.000 iterações.

10. Insira após a linha 65 e 74, respectivamente, os seguintes blocos de instruções de “laço de espera”, depois de reservar um espaço de memória rotulado com ITERACOES e inicializado com 500.000 (em decimal).



```
66 # Laco de espera com <ITERACOES> (PTB0 em 0)
67     movs    r3, #0
68     str     r3, [r7, #4]
69     b       ini1
70 laco1:
71     ldr     r3, [r7, #4]
72     add     r3, #1
73     str     r3, [r7, #4]
74 ini1:
75     ldr     r3, [r7, #4]
76     ldr     r2, =ITERACOES
77     ldr     r2, [r2, #0]
78     cmp     r3, r2
79     bls     laco1
```

```
89 # Laco de espera com <ITERACOES> (PTB0 em 1)
90     movs    r3, #0
91     str     r3, [r7, #4]
92     b       ini2
93 laco2:
94     ldr     r3, [r7, #4]
95     add     r3, #1
96     str     r3, [r7, #4]
97 ini2:
98     ldr     r3, [r7, #4]
99     ldr     r2, =ITERACOES
100    ldr     r2, [r2, #0]
101    cmp     r3, r2
102    bls     laco2
```

'b' é a instrução de desvio incondicional. As possíveis condições de desvio são listadas no Manual de Programação. Por exemplo, bls = b + ls indica que a condição de desvio é "menor ou igual" (*less or same*).

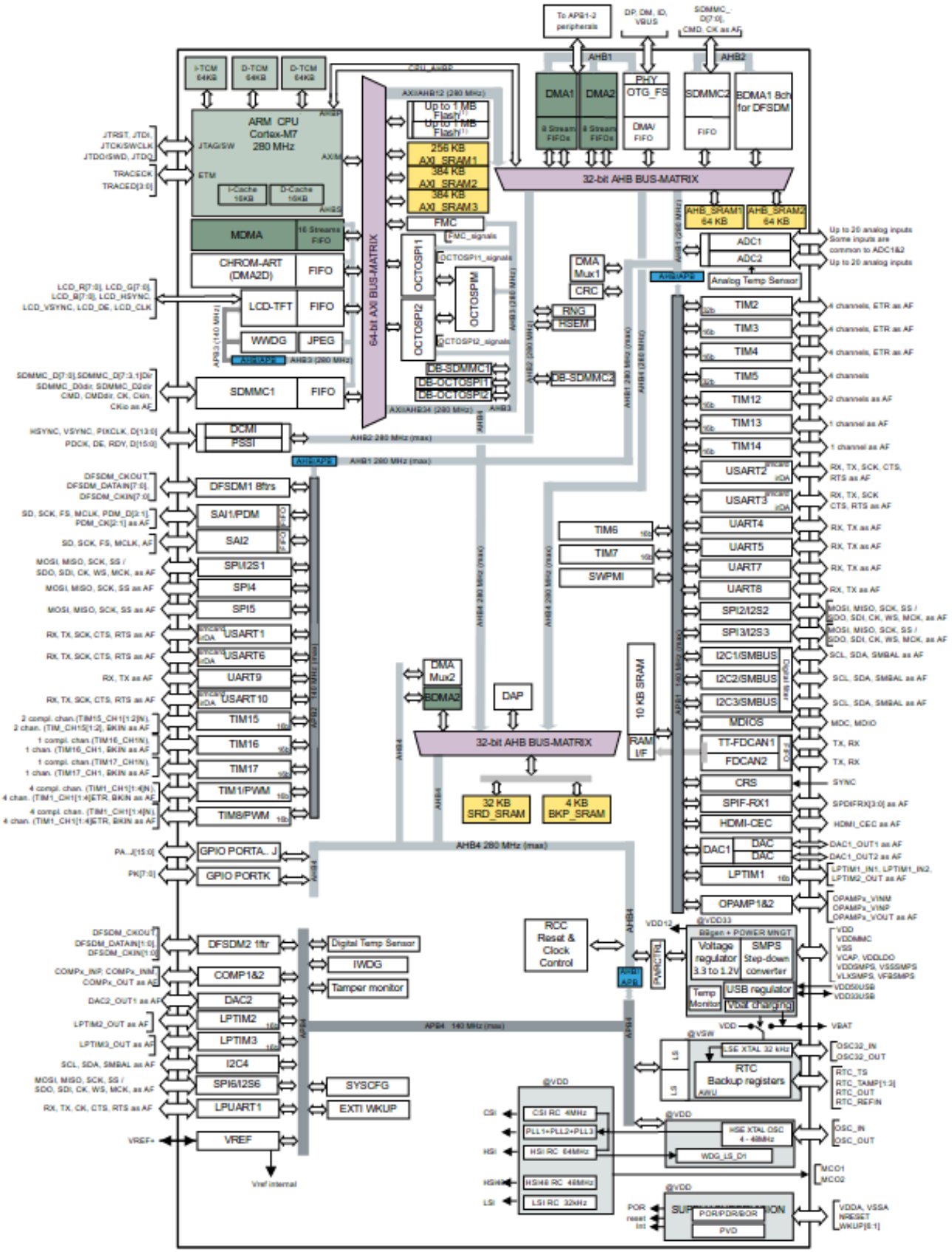
11. Construa o projeto, transfira-o para o microcontrolador e execute o projeto. Qual é o resultado observado? Neste tópico, você irá constatar a vantagem de um projeto baseado em microcontroladores, compreendendo a flexibilidade que eles oferecem na personalização do *hardware* através da programação (*software)*.

FUNDAMENTOS TEÓRICOS

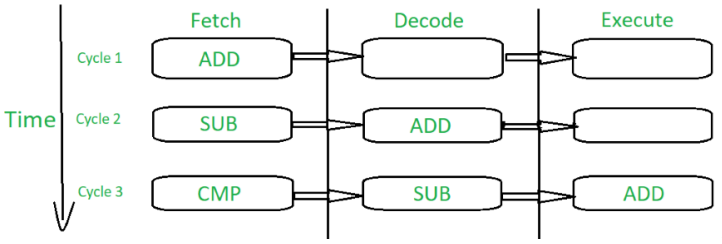
Munidos da experiência concreta de um cenário prático de alternar o estado de um LED verde usando um microcontrolador e as duas soluções através dos projetos-exemplo, vamos compreender os princípios por trás das soluções, com foco nos fundamentos teóricos que sustentam a organização do microcontrolador. Esses fundamentos são essenciais para escrever os programas, rastrear a execução de um programa, identificar e corrigir erros nos códigos, escolher o microcontrolador adequado para uma tarefa específica e projetar sistemas embarcados.

MICROCONTROLADOR STM32H7A3

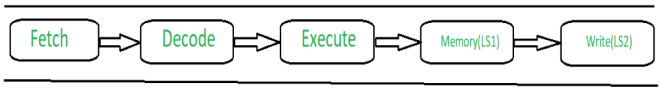
O microcontrolador STM32H7A3, parte da família STM32 de alto desempenho da STMicroelectronics, é projetado para aplicações que requerem desempenho elevado e funcionalidades avançadas. O diagrama de blocos apresentado na Figura 1 do *Datasheet* proporciona uma visão geral dos seus componentes. Ele possui um **núcleo ARM Cortex-M7**. O núcleo Cortex-M7 pertence à família ARM Cortex-M de núcleos RISC de 32 *bits*, implementando a arquitetura ARMv7E-M (*Harvard*) e operando até 280 MHz.


ARM CPU
Cortex-M7
280 MHz
I-TCM 64KB
D-TCM 64KB
I-Cache 16KB
D-Cache 16KB
MDMA
16 Streams FIFO
CHROM-ART (DMA2D)
FIFO
LCD-TFT
WWDG
JPEG
SDMMC1
DCMI PSSI
64-bit AXI BUS-MATRIX
Up to 1 MB Flash
256 KB AXI SRAM1
384 KB AXI SRAM2
384 KB AXI SRAM3
FMC
OCTOSPI1
OCTOSPI2
OCTOSPIM
DB-SDMMC1
DB-OCTOSPI1
DB-OCTOSPI2
DB-SDMMC2
DMA1
DMA2
8 Stream FIFOs
PHY OTG_FS
DMA FIFO
SDMMC2
FIFO
BDMA1 8ch for DFSDM
32-bit AHB BUS-MATRIX
AHB_SRAM1 64 KB
AHB_SRAM2 64 KB
DMA Mux1
CRC
RNG
HSEM
ADC1
ADC2
Analog Temp Sensor
TIM2
TIM3
TIM4
TIM5
TIM12
TIM13
TIM14
USART2
USART3
UART4
UART5
UART7
UART8
SPI2/I2S2
SPI3/I2S3
I2C1/SMBUS
I2C2/SMBUS
I2C3/SMBUS
MDIOS
TT-FDCAN1
FDCAN2
CRS
SPIF-RX1
HDMI-CEC
DAC1
LPTIM1
OPAMP1&2
TIM6
TIM7
SWPMI
10 KB SRAM
RAM I/F
DMA Mux2
BDMA2
DAP
32 KB SRD_SRAM
4 KB BKP_SRAM
DFSDM1 8flts
SAI1/PDM
SAI2
SPI/I2S1
SPI4
SPI5
USART1
USART6
UART9
USART10
TIM15
TIM16
TIM17
TIM1/PWM
TIM8/PWM
GPIO PORTA..J
GPIO PORTK
DFSDM2 1flt
COMP1&2
DAC2
LPTIM2
LPTIM3
I2C4
SPI6/I2S6
LPUART1
VREF
Digital Temp Sensor
IWDG
Tamper monitor
SYSCFG
EXTI WKUP
RCC Reset & Clock Control
Voltage regulator 3.3 to 1.2V
SMPS Step-down converter
USB regulator
Temp Monitor
Vbat charging
LSE XTAL 32 kHz
RTC Backup registers
HSE XTAL OSC 4 - 48MHz
CSI RC 4MHz
PLL1+PLL2+PLL3
HSI RC 64MHz
HSI48 RC 48MHz
LSI RC 32kHz
WDG_LS_DI
PVD
AHB1
AHB2
AHB3
AHB4
APB1
APB2
APB3
APB4
AXI/AHB12 (280 MHz)
AXI/AHB34 (280 MHz)
AHB1 280 MHz (max)
AHB2 280 MHz (max)
AHB3 (280 MHz)
AHB4 280 MHz (max)
APB1 140 MHz (max)
APB2 140 MHz (max)
APB3 (140 MHz)
APB4 140 MHz (max)

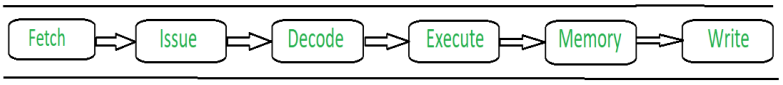
O *datapath* do núcleo Cortex-M7 é responsável pela manipulação dos dados durante o processamento das instruções. Este núcleo utiliza um *pipeline* de 6 estágios e é superescalar com emissão dupla em ordem, ou seja é capaz de executar múltiplas instruções por ciclo de relógio, utilizando um mecanismo de emissão dupla de instruções, enquanto mantém a ordem de execução das instruções conforme elas são originalmente especificadas no programa.



Pipeline de 3 estágios



Pipeline de 5 estágios



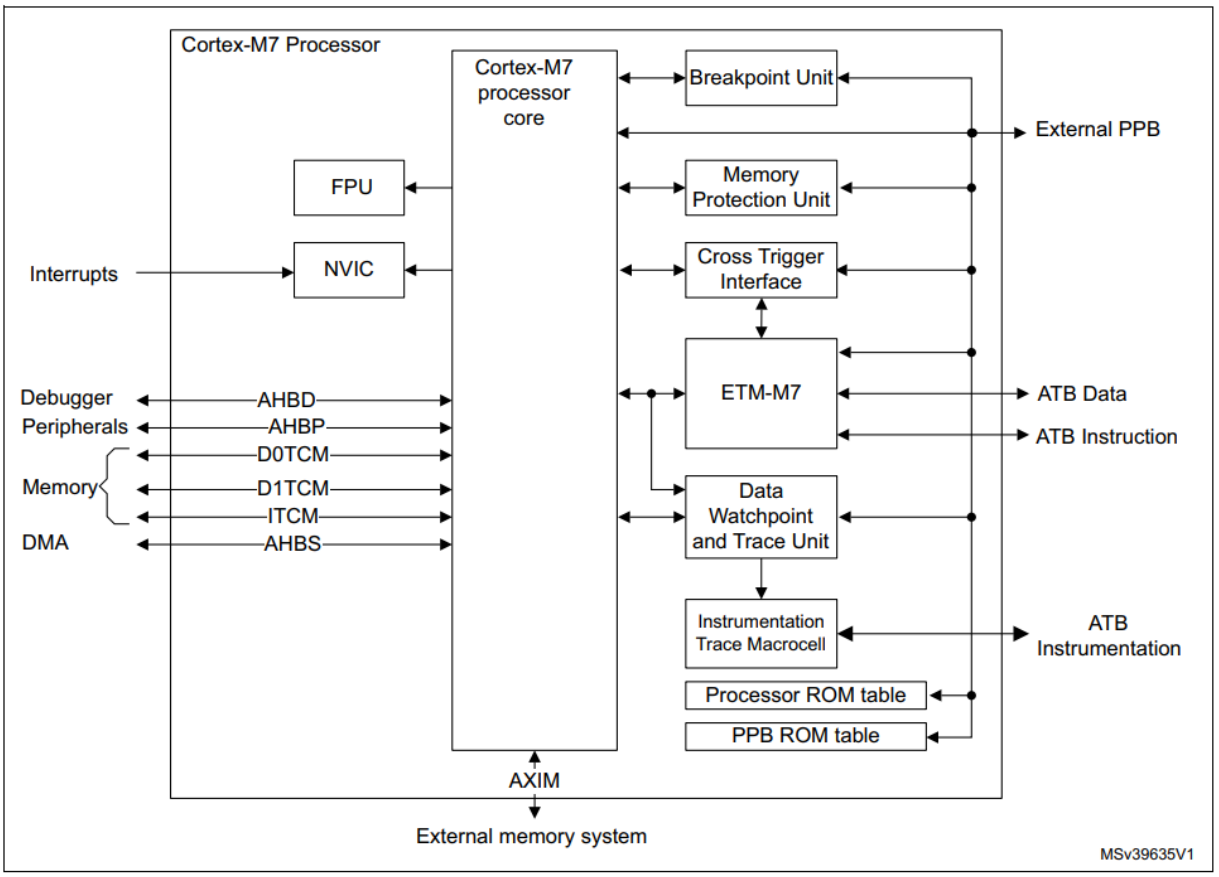
Pipeline de 6 estágios

A divisão do ***pipeline* em 6 estágios** pode variar entre diferentes arquiteturas, mas uma configuração comum inclui:

1. **Fetch (Busca de Instrução):** A instrução é buscada da memória e carregada no pipeline. O endereço da próxima instrução é recuperado.
2. **Issue (Emissão de Instrução):** A instrução é emitida para uma unidade de processamento apropriado.
3. **Decode (Decodificação de Instrução):** A instrução é decodificada para identificar qual operação deve ser realizada. Aqui, os operandos e os registradores necessários são identificados e preparados.
4. **Execution (Execução):** A operação aritmética ou lógica especificada pela instrução é realizada pela Unidade Lógica e Aritmética (ALU) ou outra unidade de execução.
5. **Memory Access (Acesso à Memória):** Se a instrução envolve uma operação de leitura ou escrita na memória (como em instruções de carga e armazenamento), o acesso à memória é realizado nesta etapa.
6. **Write Back (Gravação de Resultados):** O resultado da operação é escrito de volta nos registradores ou na memória, completando o ciclo de execução da instrução.

Possui suporte para unidade de ponto flutuante (FPU, do inglês *floating point unit*) de precisão simples e dupla, e instruções de processamento de sinais digitais (DSP, do inglês *digital signal processing*) para processamento de sinais, além de SIMD (do inglês *Single Instruction and Multiple Data*)[1]. O processador suporta também o controlador de **interrupções aninhadas** (NVIC do inglês *Nested Vectored Interrupt Controller*). A função do NVIC é gerenciar interrupções em microcontroladores baseados na arquitetura ARM Cortex-M. Ele permite que várias interrupções ocorram de forma hierárquica ou aninhada. Isso significa que, enquanto o processador está tratando uma interrupção de alta prioridade, ele pode ser temporariamente interrompido por uma interrupção de prioridade mais alta. Após a conclusão da interrupção de maior prioridade, o processador retoma a execução da interrupção anterior ou, se a interrupção mais alta for ainda mais prioritária, ela pode ser tratada imediatamente. O diagrama de blocos extraído do Manual de Programação mostra os principais componentes do processador Cortex-M7.

Figure 1. STM32 Cortex®-M7 implementation processor



Comparado ao Cortex-M4, o Cortex-M7 apresenta um desempenho mais próximo ao de um processador de sinal digital. Ele é capaz de executar operações de carga (*load*), armazenamento (*save*) e aritmética em paralelo, reduzindo a sobrecarga em laços. O Cortex-M7 interage

[1] SIMD é um tipo de arquitetura de processador paralelo que permite que uma única instrução opere em vários dados ao mesmo tempo. Isso contrasta com um **processador escalar,** que só pode operar em um único dado por vez.

diretamente com memórias TCM (do inglês *Tightly Coupled Memory*), o que resulta em baixa latência de interrupção e execução mais determinística. Dispõe também de macrocélulas de rastreamento para depuração e análise de *software*.

O STM32H7A3 é equipado com um **sistema de memória** robusto que suporta uma variedade de necessidades em sistemas embarcados. Ele inclui até 2 MB de memória *Flash*, utilizada para armazenar o código do programa de maneira não volátil. Para o armazenamento temporário de dados durante a execução do programa, o microcontrolador oferece cerca de 1,4 MB de RAM, que se divide em duas categorias principais. A primeira é a SRAM (do inglês *Static Random Access Memory*), conhecida por sua alta velocidade e eficiência energética. A segunda é a TCM RAM (do inglês *Tightly Coupled Memory* RAM), que também é baseada em tecnologia SRAM, mas é integrada diretamente ao caminho de dados do processador, proporcionando acesso ultra-rápido a dados críticos. A tabela extraída do Manual de Referência mostra o mapeamento dessas unidades de memória no espaço de endereçamento do núcleo.

Table 6. Memory map and default device memory area attributes (continued)

Region	Boundary address	Arm® Cortex®-M7	Type	Attributes	Execute never
RAM	0x3880 1000 - 0x3FFF FFFF	Reserved	Normal	Write-back, write allocate cache attribute	No
	0x3880 0000 - 0x3880 0FFF	Backup SRAM			
	0x3801 0000 - 0x387F FFFF	Reserved			
	0x3800 0000 - 0x3800 7FFF	SDR SRAM			
	0x3002 0000 - 0x37FF 7FFF	Reserved			
	0x3001 0000 - 0x3001 FFFF	AHB SRAM2			
	0x3000 0000 - 0x3000 FFFF	AHB SRAM1			
	0x2600 0000 - 0x2FFF FFFF	Reserved			
	0x2500 0000 - 0x25FF FFFF	GFXMMU			
	0x2410 0000 - 0x24FF FFFF	Reserved			
	0x240A 0000 - 0x240F FFFF	AXI SRAM3			
	0x2404 0000 - 0x2409 FFFF	AXI SRAM2			
	0x2400 0000 - 0x2403 FFFF	AXI SRAM1			
	0x2002 0000 - 0x23FF FFFF	Reserved			
	0x2000 0000 - 0x2001 FFFF	DTCM			
Code	0x1FF2 0000 - 0x1FFF FFFF	Reserved	Normal	Write-through cache attribute	No
	0x1FF0 0000 - 0x1FF1 FFFF	System Memory			
	0x08FF F400 - 0x1FEF FFFF	Reserved			
	0x08FF F000 - 0x08FF F3FF	OTP area	Normal	-	No
	0x0820 0000 - 0x08FF EFFF	Reserved	Normal	Write-through cache attribute	No
	0x0810 0000 - 0x081F FFFF	Flash memory bank 2			
	0x0800 0000 - 0x080F FFFF	Flash memory bank 1			
	0x0001 0000 - 0x07FF FFFF	Reserved			
	0x0000 0000 - 0x0000 FFFF	ITCM RAM			

O STM32H7A3 vai além das suas memórias internas ao integrar um **controlador de memória flexível** (FMC, do inglês *Flexible Memory Controller*) e suportar a expansão de memória externa

através de diversas interfaces. Isso inclui Quad-SPI[2] para acesso rápido a memórias *Flash* de alta velocidade, SDRAM para manipulação eficiente de grandes volumes de dados temporários, e a capacidade de conexão de SRAM externa para expansão conforme necessário. Além disso, oferece opções de emulação de EEPROM usando memória *Flash* para o armazenamento seguro e não volátil de dados cruciais, mesmo durante falhas de energia. A tabela extraída do Manual de Referência mostra o mapeamento dessas memórias externas no espaço de endereçamento do núcleo.

Table 6. Memory map and default device memory area attributes

Region	Boundary address	Arm® Cortex®-M7	Type	Attributes	Execute never
External devices	0xD000 0000 - 0xDFFF FFFF	FMC SDRAM Bank2 (or reserved in case of FMC remap)	Device	-	Yes
	0xCC00 0000 - 0xCFFF FFFF	FMC SDRAM Bank1 (or remap of FMC NOR/PSRAM/SRAM 4 Bank1)			
	0xC800 0000 - 0xCBFF FFFF	FMC SDRAM Bank1 (or remap of FMC NOR/PSRAM/SRAM 3 Bank1)			
	0xC400 0000 - 0xC7FF FFFF	FMC SDRAM Bank1 (or remap of FMC NOR/PSRAM/SRAM 2 Bank1)			
	0xC000 0000 - 0xC3FF FFFF	FMC SDRAM Bank1 (or remap of FMC NOR/PSRAM/SRAM 1 Bank1)			
	0xA000 0000 - 0xBFFF FFFF	Reserved			
External memories	0x9000 0000 - 0x9FFF FFFF	OCTOSPI1	Normal	Write-through cache attribute	No
	0x8000 0000 - 0x8FFF FFFF	FMC NAND flash memory			
	0x7000 0000 - 0x7FFF FFFF	OCTOSPI2			
	0x6C00 0000 - 0x6FFF FFFF	FMC NOR/PSRAM/SRAM 4 Bank1 (or remap of FMC SDRAM Bank1)		Write-back, write allocate cache attribute	
	0x6800 0000 - 0x6BFF FFFF	FMC NOR/PSRAM/SRAM 3 Bank1 (or remap of FMC SDRAM Bank1)			
	0x6400 0000 - 0x67FF FFFF	FMC NOR/PSRAM/SRAM 2 Bank1 (or remap of FMC SDRAM Bank1)			
	0x6000 0000 - 0x63FF FFFF	FMC NOR/PSRAM/SRAM 1 Bank1 (or remap of FMC SDRAM Bank1)			
Peripherals	0x4000 0000 - 0x5FFF FFFF	Peripherals (refer to *Table 7: Register boundary addresses*)	Device	-	Yes

O STM32H7A3 inclui uma **variedade de periféricos**, como interfaces de comunicação (UART, SPI, I2C, CAN, USB, HDMI e SAI), vários temporizadores (incluindo temporizadores de propósito geral, avançados e de alta resolução), conversores analógico-digital (ADC) de alta precisão e conversores digital-analógico (DAC), além de pinos GPIO configuráveis. Controladores e interfaces adicionais incluem um controlador de acesso direto à memória (DMA) para transferências de dados eficientes, e um relógio em tempo real (RTC) para temporização precisa. Em termos de segurança, o microcontrolador possui aceleradores de criptografia para segurança de dados e regiões de memória

[2] Quad-SPI (do inglês *Serial Peripheral Interface Quad*) é uma evolução do padrão SPI (do inglês *Serial Peripheral Interface*) que oferece um desempenho superior, especialmente em aplicações que exigem alta velocidade de transferência de dados, ao usar quatro linhas paralelas de dados ao invés de uma única linha..

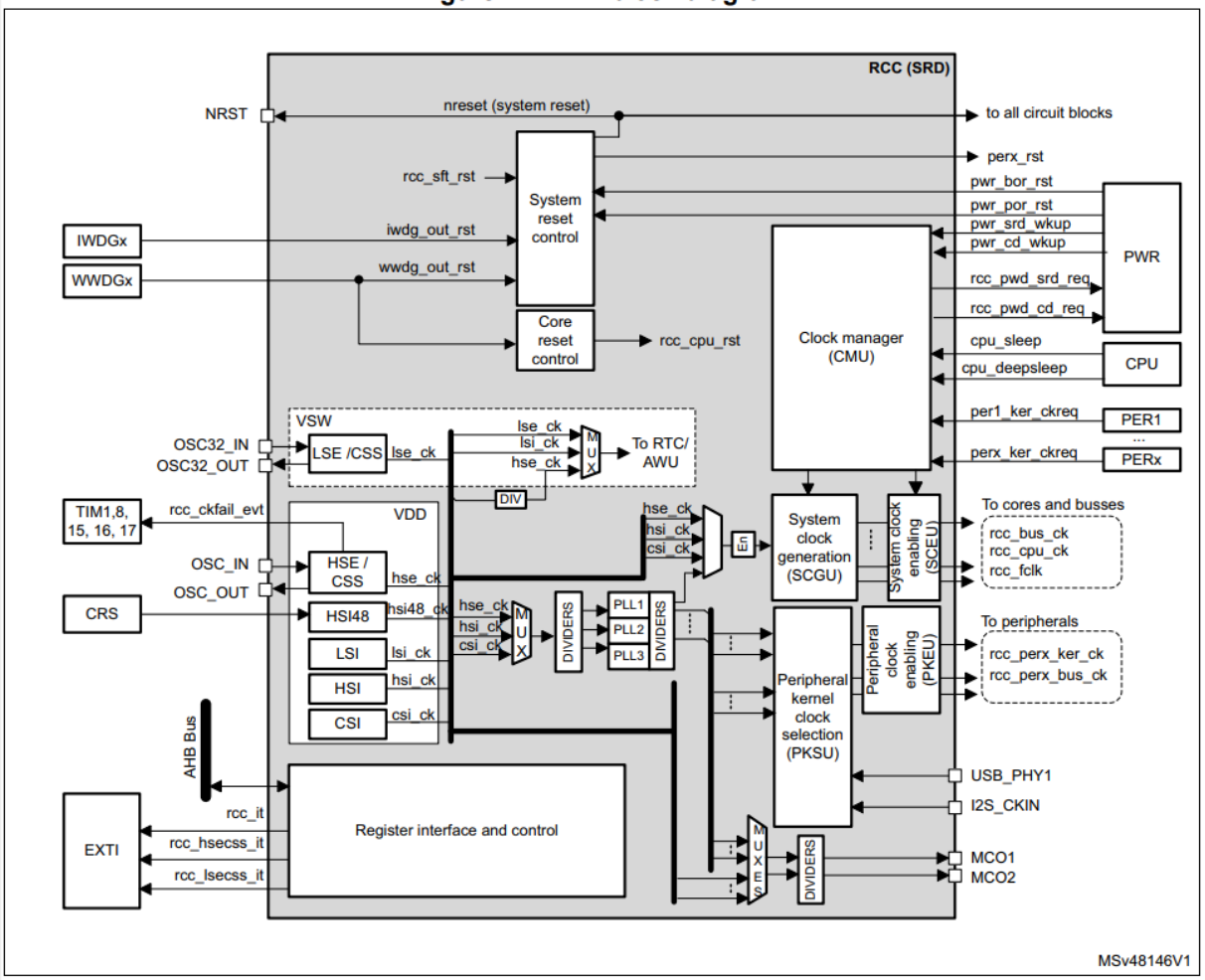
protegida através de MPU (do inglês *Memory Protection Unit*) para armazenamento seguro. Exploraremos uma parte desses periféricos ao longo dessa disciplina.

Para **programação e depuração dos códigos**, são integradas duas interfaces, **SWD** (do inglês *Serial Wire Debug*) e **JTAG** (do inglês *Joint Test Action Group*). JTAG é uma interface padrão de depuração e teste que utiliza múltiplos pinos (geralmente cinco, TDI, TDO, TCK, TMS e TRST) para comunicação. Ela permite não apenas a depuração de *software*, mas também testes estruturais de *hardware*, como verificação de conexões de solda e teste de integridade de circuitos. SWD, por sua vez, é uma interface de depuração serial projetada para oferecer uma alternativa de depuração de pinos reduzidos em comparação com a interface JTAG tradicional. SWD utiliza apenas dois pinos para comunicação (SWDIO e SWCLK), o que é vantajoso em termos de economia de pinos e simplicidade.

O **sistema de gerenciamento de sinais de relógio** do STM32H7A3 é sofisticado, contando com múltiplas fontes de relógio e multiplexadores para fornecer a flexibilidade necessária para diferentes modos de operação e diferentes domínios de relógio para periféricos distintos, permitindo uma gestão eficiente da energia e do desempenho. Ele inclui um oscilador de alta velocidade externo (HSE, do inglês *High Speed External*) e um oscilador de baixa velocidade externo (LSE, do inglês *Low Speed External*), bem como um oscilador interno de alta velocidade (HSI, do inglês *High Speed Internal*) e um oscilador de baixa velocidade interno (LSI, do inglês *Low Speed Internal*). Dispõe ainda de um oscilador de baixa velocidade CSI (do inglês *Clock Security System Internal*), que fornece sinais de relógio confiáveis e estáveis de baixa velocidade quando outras fontes não estão disponíveis. Há um sistema de segurança dos sinais de relógio (CSS, do inglês *Clock Security System*) que monitora a integridade dos sinais de relógio, particularmente o oscilador de alta velocidade externo (HSE).

Um *Phase-Locked Loop* (PLL) é usado para gerar frequências de relógio mais altas com alta precisão e estabilidade a partir de fontes como HSE ou HSI, fornecendo o sinal de *clock* necessário para o processador e seus periféricos. O PLL é um circuito de controle que ajusta dinamicamente a frequência de um oscilador para que ele se mantenha em fase com uma frequência de referência. Ele inclui um oscilador de referência, divisores de fase e de realimentação, um detector de fase e um oscilador controlado por tensão (VCO, do inglês *Voltage-Controlled Oscillator*). O funcionamento básico do PLL envolve a comparação da frequência dividida do oscilador de referência com a frequência dividida do sinal gerado pelo VCO. Com base nessa comparação, o detector de fase gera um sinal de erro que ajusta a tensão de controle do VCO, mantendo a saída do VCO sincronizada com a frequência de referência e efetivamente gerando um sinal que é múltiplo da frequência de referência. O diagrama de blocos extraído do Manual de Referência sintetiza a relação dos sinais de relógio presentes em STM32H7A3.

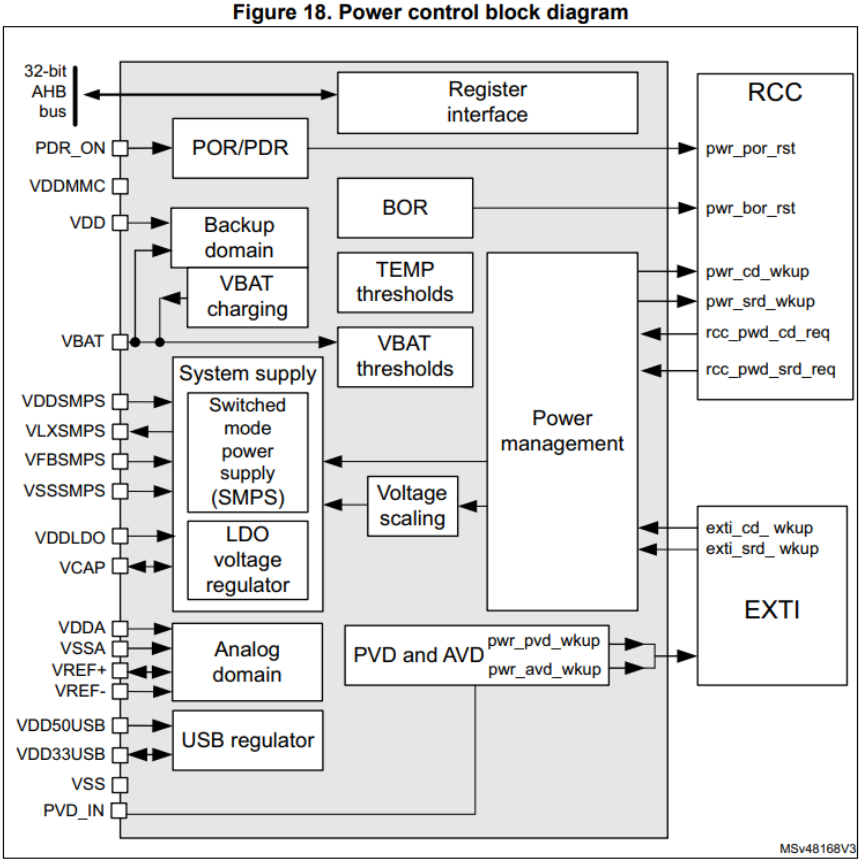
Figure 44. RCC block diagram



Em termos de **fonte de alimentação de energia**, o STM32H7A3 suporta uma faixa de tensão de 1,7V a 3,6V, com reguladores internos de tensão para fornecer a tensão necessária aos diferentes blocos internos. Ele inclui modos de baixa potência, como modo de espera e modo de suspensão, para reduzir o consumo de energia em aplicações que exigem alta eficiência energética. A fonte de alimentação pode ser gerida de forma dinâmica para balancear desempenho e eficiência de energia. É provido de um conjunto de circuitos que monitoram e garantem que a tensão de alimentação do microcontrolador esteja dentro de faixas operacionais seguras sob diferentes condições de alimentação e temperatura. O *Power-On Reset* (POR) garante que o microcontrolador reinicie corretamente quando a energia é aplicada, enquanto o *Power-Down Reset* (PDR) monitora e assegura que o microcontrolador seja corretamente reiniciado quando a tensão de alimentação cai abaixo de um certo limiar. O monitor BOR (do inglês *Brown-Out Reset Monitor*) supervisiona a tensão de alimentação para detectar quedas momentânea, conhecidas por *brown-outs*, e reseta o microcontrolador nestas condições. O PVD (do inglês *Programmable Voltage Detector*) permite ao microcontrolador monitorar a tensão de alimentação e tomar ações programáveis quando a tensão ultrapassa ou cai abaixo de um limiar configurável. Similar ao PVD, o AVD (do inglês *Analog Voltage Detector*) monitora tensões analógicas e pode gerar interrupções ou resetar o

microcontrolador se as tensões analógicas ultrapassarem limites pré-definidos. O monitor de VBAT (do inglês *Voltage Battery*) verifica a tensão da bateria de *backup* que alimenta o RTC e outros circuitos de *backup*. Ele garante que a tensão da bateria esteja adequada para manter esses circuitos operacionais durante a ausência da principal fonte de alimentação. E o monitor de temperatura garante que a temperatura do microcontrolador permaneça dentro de faixas operacionais seguras. Se a temperatura exceder ou cair abaixo dos limiares pré-definidos, o microcontrolador pode tomar ações corretivas, como reduzir a frequência de operação ou entrar em modo de baixa potência. O diagrama de blocos extraído do Manual de Referência sintetiza os circuitos presentes no controlador de energia em STM32H7A3.

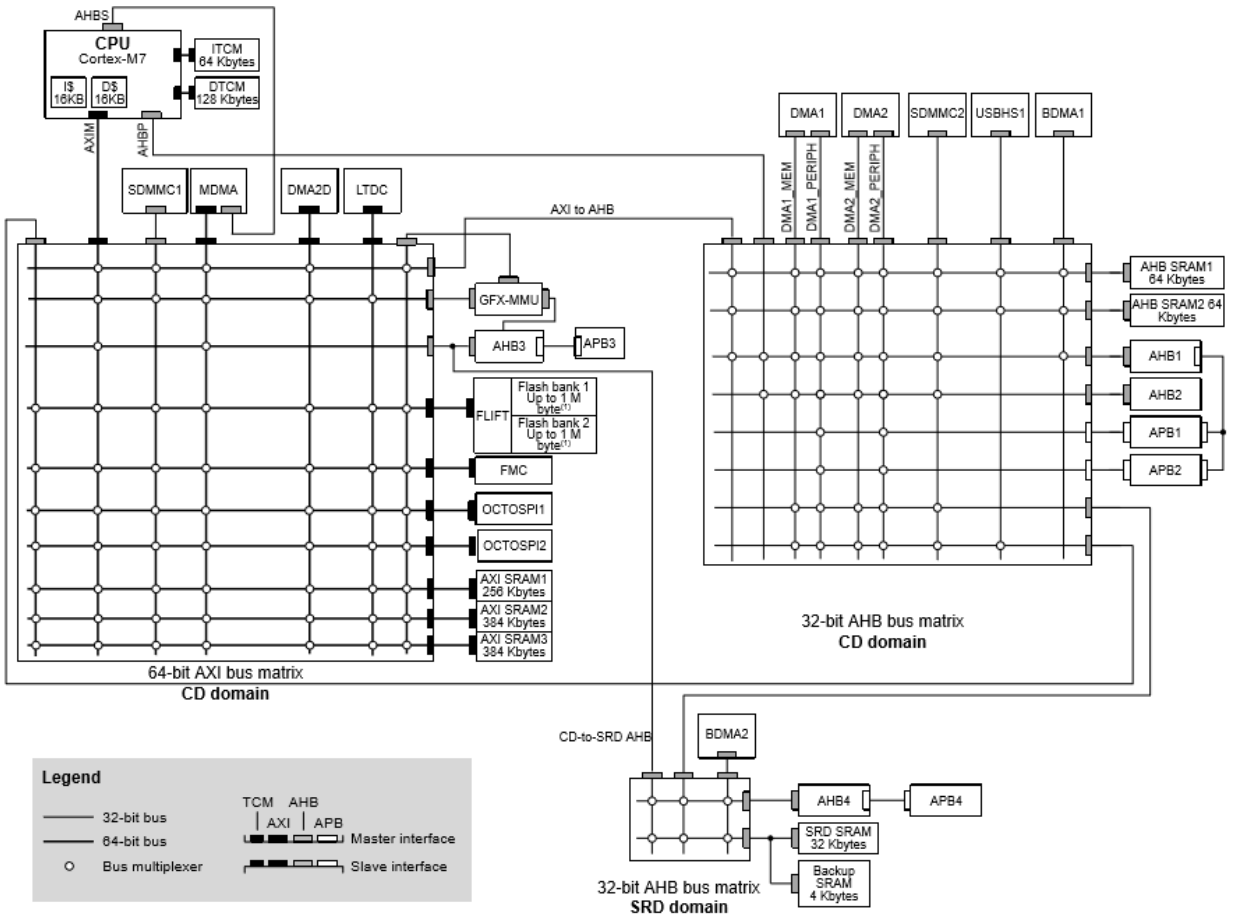
Figure 18. Power control block diagram



No STM32H7A3ZIT6-Q, a arquitetura de **barramentos** é projetada para otimizar o desempenho e a comunicação entre o processador e os periféricos, utilizando uma matriz de interconexão (*interconnect matrix*). O *Advanced High-performance Bus* (AHB) é o barramento de alto desempenho que conecta o núcleo do processador e a memória com periféricos de alta velocidade, como a memória *Flash* e SRAM. O *Advanced Peripheral Bus* (APB), por sua vez, conecta periféricos de menor velocidade e com requisitos de largura de banda menos críticos, como periféricos de comunicação. Para atender às necessidades variadas de comunicação e largura de banda entre o processador, a memória e os periféricos, esses barramentos são subdivididos em grupos como AHB1, AHB2, AHB3 e AHB4, e APB1, APB2, APB3 e APB4, cada um atendendo a

diferentes especificidades e requisitos dos dispositivos. A matriz de interconexão gerencia o tráfego entre esses barramentos e os periféricos, assegurando que as solicitações de acesso sejam atendidas com eficiência e mínima latência. Os controladores DMA (do inglês *Direct Memory Access)*, conectados na matriz de interconexão, permitem transferências de dados diretas entre a memória e os periféricos sem a intervenção do núcleo, utilizando canais dedicados e uma matriz de interconexão para garantir acesso eficiente e simultâneo a múltiplos recursos, melhorando o desempenho geral do sistema. O diagrama de blocos extraído do *Datasheet* ilustra a matriz de interconexão de barramentos e controladores DMA em STM32H7A3.

Figure 3. STM32H7A3xI/G bus matrix



STM32H7A3xI and STM32H7A3xG devices feature two banks of 1 Mbyte and 512 Kbytes each, respectively.

O STM32H7A3 é ideal para uma ampla gama de aplicações, incluindo automação industrial, eletrônica de consumo avançada, aplicações médicas, *Internet* das Coisas (IoT) e sistemas embarcados que exigem processamento intensivo e comunicação rápida. Combinando um processador/de núcleo de alto desempenho com uma rica seleção de periféricos e funcionalidades de segurança, suportado por um ambiente de desenvolvimento robusto e ferramentas abrangentes, o STM32H7A3A é uma excelente escolha para desenvolvedores de sistemas embarcados complexos e de alto desempenho.

NUCLEO-H7A3ZI-Q

A NUCLEO-H7A3ZI-Q é uma placa de desenvolvimento equipada com o microcontrolador STM32H7A3ZIT6, projetada para facilitar a prototipagem rápida e a criação de projetos baseados na família STM32H7A3ZIT6-Q. Esta placa oferece uma ampla gama de periféricos e circuitos adicionais que simplificam o desenvolvimento de aplicações avançadas. Por integrar o STM32H7A3ZIT6-Q, a placa vem equipada com uma variedade de interfaces e periféricos que facilitam a comunicação e a integração com outros dispositivos e sensores:

- **USB OTG FS**: Suporte para USB *On-The-Go* de *Full Speed*, permitindo a placa atuar como dispositivo ou *host* USB.
- **USART/UART**: Várias interfaces seriais para comunicação com periféricos seriais.
- **SPI, I2C, CAN, e SAI**: Interfaces de comunicação serial síncrona rápidas e eficientes para conectar diversos tipos de sensores e dispositivos.
- **ADC/DAC**: Conversores analógico-digital e digital-analógico para leitura e geração de sinais analógicos.

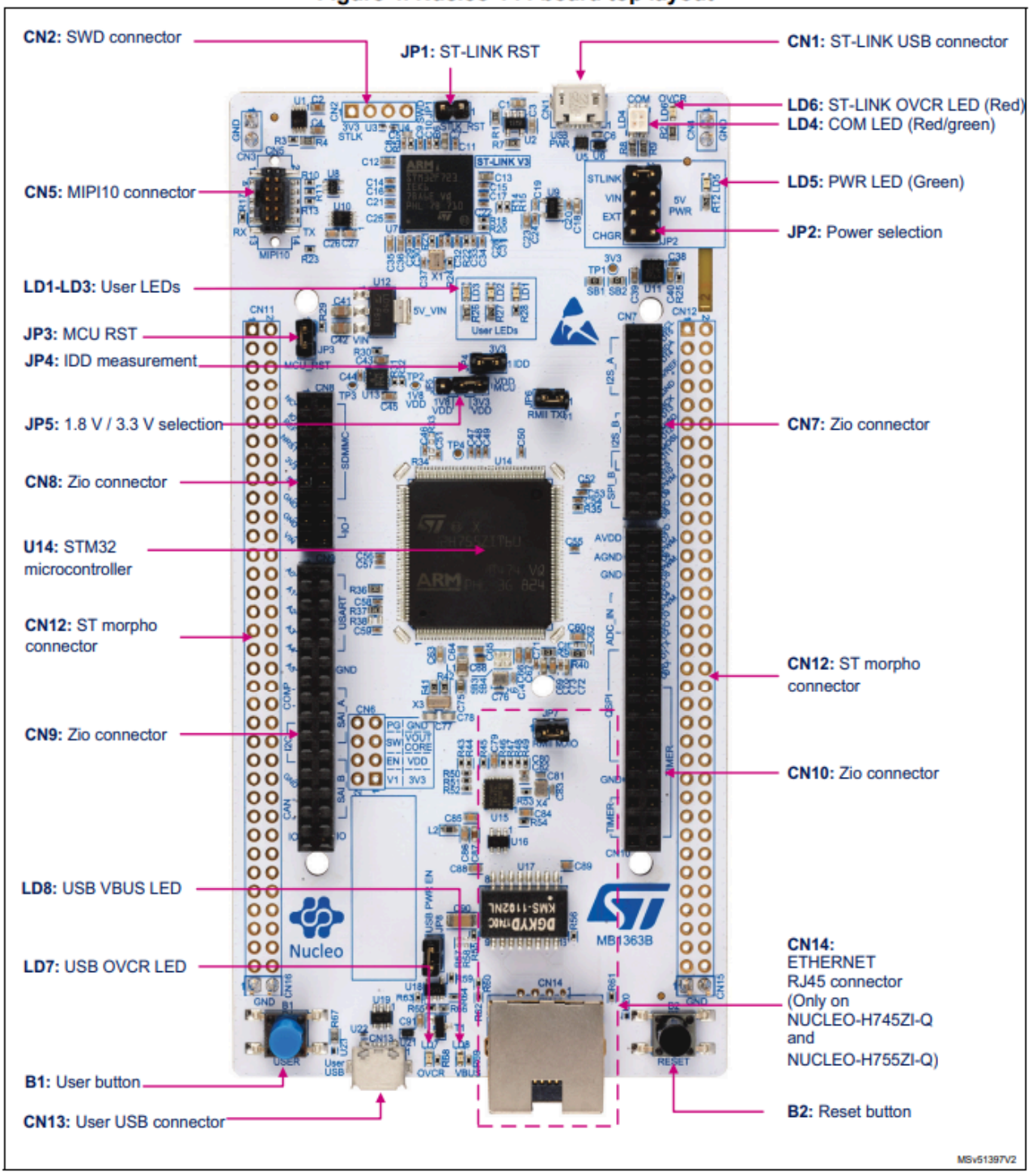
A NUCLEO-H7A3ZI-Q inclui vários circuitos adicionais que tornam a prototipagem e o desenvolvimento mais rápidos e eficientes:

- **ST-LINK/V2-1**: Depurador/programador *on-board* que facilita a programação e a depuração do código diretamente pela interface USB.
- **Conectores ST *Morpho* e Arduino**: Pinagens compatíveis com *shields* Arduino e pinos adicionais no estilo *Morpho*, fornecendo acesso a praticamente todos os pinos do STM32H7A3ZIT6-Q e permitindo a fácil expansão com módulos adicionais e *shields*.
- **Cristais de Relógio**: Inclui osciladores externos de alta precisão (HSE) e de baixa frequência (LSE) para temporização precisa e operação do RTC.
- **Botões e LEDs**: Botões de usuário e LEDs de estado (incluindo LED de alimentação e de usuário) para interações simples e indicações visuais rápidas.
- **Circuito de Fonte de Alimentação**: Suporte para múltiplas fontes de alimentação, incluindo alimentação via USB, conectores VIN e conector E5V, permitindo flexibilidade no fornecimento de energia.

A figura extraída do Manual de Usuário proporciona uma visão geral da placa. Essa placa é compatível com diversas ferramentas de desenvolvimento e ambientes de programação, incluindo o STM32CubeIDE, que facilita a configuração e o desenvolvimento de *firmware* para o

microcontrolador STM32H7A3ZIT6. Além disso, a biblioteca STM32CubeH7 oferece exemplos de código e *drivers* para uma rápida integração dos periféricos.

Figure 4. Nucleo-144 board top layout



PERSONALIZAÇÃO DE STM32H73A

Agora vamos ver como podemos particularizar as funções do STM32H3A para uma função específica de piscar o LED LD1 (verde) da Nucleo-H7A3ZI-Q. Para isso, devemos consultar os manuais e/ou folhas de dados do microcontrolador e da NUCLEO para identificar os registradores de configuração/controle, estado e dados envolvidos na operação. No caso do LED verde implementado na NUCLEO, destacamos o trecho de descrição dos LEDs disponíveis para usuários no Manual de Usuário.

O LED a ser controlado é ligado ao pino PB0, ou seja, o pino 0 da Porta B do microcontrolador.

6.6 LEDs

User LD1: a green user LED is connected to the STM32H7 I/O PB0 (SB65 OFF and SB54 ON) or PA5 (SB65 ON and SB54 OFF) corresponding to the ST Zio D13.

User LD2: a yellow user LED is connected to PE1.

User LD3: a red user LED is connected to PB14.

Caution: LD3 cannot be used if Zio D65 is used (In this case, remove R75).

These user LEDs are on when the I/O is HIGH value, and are off when the I/O is LOW.

LD4 COM: the tricolor LED LD4, green, orange, red, provides information about ST-LINK communication status. LD4 default color is red. LD4 turns to green to indicate that communication is in progress between the PC and the STLINK-V3E, with the following setup:

- Slow blinking red or OFF: At power-on before USB initialization
- Fast blinking red or OFF: After the first correct communication between PC and STLINK-V3E (Enumeration)
- Red LED ON: When the initialization between the PC and STLINK-V3E is complete
- Green LED ON: After a successful target communication initialization
- Blinking red/green: During communication with the target
- Green ON: Communication finished and successful
- Orange ON: Communication failure

LD5 PWR: the green LED indicates that the STM32H7 part is powered and +5 V power is available on CN8 pin 9 and CN11 pin 18.

LD6 USB power fault: LD5 indicates that the board power consumption on USB exceeds 500 mA, consequently the user must power the board using an external power supply.

LD7 and LD8 USB FS: refer to *Section 6.11: USB OTG FS or device*.

Para que o LED LD1 possa ser controlado, é necessário ativar a Porta B (que é desativada por padrão na inicialização do microcontrolador) pelo controlador “RCC” (*Reset and Clock Controller*) integrado no microcontrolador, e configurar o pino 0 desta porta para atuar como saída pelo módulo “GPIO” (*General-Purpose Input/Outputs*). Seguem-se os trechos de descrição desses dois componentes no *Datasheet* do microcontrolador STM32HA3x1/G.

3.7 Reset and clock controller (RCC)

The clock and reset controller is located in the SRD domain. The RCC manages the generation of all the clocks, as well as the clock gating and the control of the system and peripheral resets. It provides a high flexibility in the choice of clock sources and allows to apply clock ratios to improve the power consumption. In addition, on some communication peripherals that are capable to work with two different clock domains (either a bus interface clock or a kernel peripheral clock), the system frequency can be changed without modifying the baud rate.

3.7.1 Clock management

The devices embed four internal oscillators, two oscillators with external crystal or resonator, two internal oscillators with fast startup time and three PLLs.

The RCC receives the following clock source inputs:

- Internal oscillators:
  - 64 MHz HSI clock (1% accuracy)
  - 48 MHz RC oscillator
  - 4 MHz CSI clock
  - 32 kHz LSI clock
- External oscillators:
  - 4-50 MHz HSE clock
  - 32.768 kHz LSE clock

The RCC provides three PLLs: one for system clock, two for kernel clocks.

The system starts on the HSI clock. The user application can then select the clock configuration.

A high precision can be achieved for the 48 MHz clock by using the embedded clock recovery system (CRS). It uses the USB SOF signal, the LSE or an external signal (SYNC) to fine tune the oscillator frequency on-the- fly.

3.7.2 System reset sources

Power-on reset initializes all registers while system reset reinitializes the system except for the debug, part of the RCC and power controller status registers, as well as the backup power domain.

A system reset is generated in the following cases:

- Power-on reset (pwr_por_rst)
- Brownout reset
- Low level on NRST pin (external reset)
- Window watchdog
- Independent watchdog
- Software reset
- Low-power mode security reset
- Exit from Standby

3.8 General-purpose input/outputs (GPIOs)

Each of the GPIO pins can be configured by software as output (push-pull or open-drain, with or without pull-up or pull-down), as input (floating, with or without pull-up or pull-down) or as peripheral alternate function. Most of the GPIO pins are shared with digital or analog alternate functions. All GPIOs are high-current-capable and have speed selection to better manage internal noise, power consumption and electromagnetic emission.

After reset, all GPIOs are in Analog mode to reduce power consumption.

The I/O configuration can be locked if needed by following a specific sequence in order to avoid spurious writing to the I/Os registers.

To maximize the performance, the I/O high-speed feature, HSLV, must be activated at low device supply voltage. This is needed to achieve the performance required for peripherals such as the SDMMC, FMC and OCTOSPI. The GPIOs are divided into four groups which can be optimized separately (refer to the description of HSLVx bits of SYSCFG_CCCSR register in RM0455).

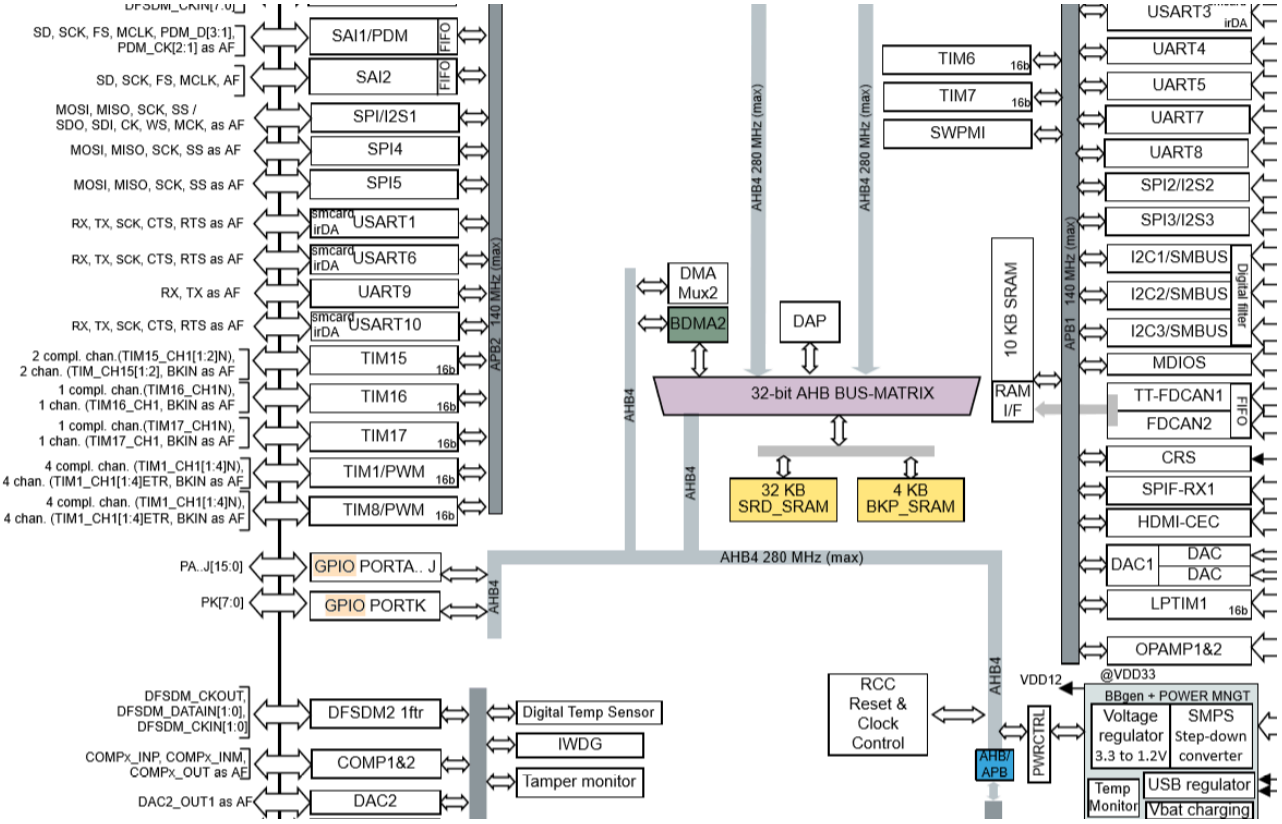
The I/O high-speed feature must be used only when V_{DD} is lower than 2.7 V, and both the HSLV user option bits (VDDIO_HSLV and VDDMMC_HSLV) and HSLVx bits must be set to enable it (refer to RM0455 for details).

Na arquitetura ARM, os periféricos internos do microcontrolador são mantidos inativos na inicialização através de um mecanismo denominado *clock gating,* com a finalidade de melhorar a eficiência energética. Cada periférico utilizado deve ser inicialmente ativado, caso contrário o acesso aos registradores correspondentes irá gerar uma exceção de *memory fault*. Para determinar os

endereços de memória em que os diversos registradores são mapeados e a função de cada um de seus *bits*, usamos o *Datasheet* e o Manual de Referência correspondentes ao microcontrolador utilizado.

Inicialmente, precisamos ativar o periférico PORTB, para que se possa utilizar qualquer pino da porta correspondente. Isto é feito colocando um determinado *bit* de um registrador de controle de *clock* em "1". Para determinar qual o registrador de controle devemos acessar, precisamos determinar em qual dos barramentos internos da arquitetura ARM o PORTB está conectado.

Na figura extraída do *Datasheet* do STM32H7A3, pode-se ver a figura das conexões com os barramentos. Ao lado esquerdo, pode-se ver um bloco denominado "GPIO PORTA..J", representando as portas digitais de propósito geral de A até J. Este conjunto está ligado ao barramento AHB4, portanto devemos procurar os *bits* de ativação destas portas nos registradores de controle do barramento AHB4, que pertencem ao controlador "RCC". As informações detalhadas dos registradores de um determinado módulo, como o módulo X, são encontradas na seção específica "X registers" dentro do respectivo capítulo no Manual de Referência do microcontrolador. Por exemplo, a seção "GPIO registers" dentro do capítulo GPIO contém descrições completas de todos os registradores associados ao módulo GPIO.



O RCC é descrito no capítulo 8 ("RCC"). A seção "RCC registers" descreve todos os registradores e a seção "RCC register map", os endereços de memória usados para acessar esses registradores. São ao todo 55 registradores de controle, Felizmente, os registradores são tipicamente nomeados de acordo com a função que desempenham para facilitar a busca e a compreensão do seu uso. Procurando pela função AHB4, reduzimos os 55 registradores para 2: "RCC AHB4 Clock Register" (RCC_AHB4ENR, seção 8.7.41) e "RCC AHB4 sleep clock register" (RCC_AHB4LPENR, seção 8.7.50). Uma leitura na descrição dos dois registradores nos leva a concluir que RCC_AHB4ENR é

responsável pela ativação de vários componentes, incluindo a Porta B pelo *bit* 1 denominado GPIOBEN (GPIO B ENable).

8.7.41 RCC AHB4 Clock Register (RCC_AHB4ENR)

Address offset: 0x140

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	SRDSRAMEN	BKPRAMEN	Res.	Res.	Res.	Res.	Res.	Res.	BDMA2EN	Res.	Res.	Res.	Res.	Res.
		rw	rw							rw					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	GPIOKEN	GPIOJEN	GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Para ativar a Porta B, só coloca o *bit* 1 em 1. Por isso, precisamos setar este *bit* sem alterar os demais. Ou seja, o conteúdo do registrador RCC_AHB4ENR deve ser (RCC_AHB4ENR *OR* 0x00000002).

Cada registrador inclui não apenas a descrição de cada *bit*, mas também "*Address offset*" e "*Reset value*". O "*Address offset*" nos permite calcular o endereço efetivo do registrador, somando-o ao endereço-base (*boundary address*) listado no Manual de Referência. Enquanto isso, o "*Reset value*" indica o valor inicial atribuído ao registrador quando o microcontrolador é inicializado. Sendo 0x58024400 o endereço-base do RCC, o endereço de RCC_AHB4ENR é 0x58024400 + 0x140 = 0x58024540.

Table 7. Register boundary addresses(1)

Boundary address	Peripheral	Bus	Register map
0x58025800 - 0x58025BFF	DMAMUX2	AHB4 (SRD)	*Section 17.6: DMAMUX registers*
0x58025400 - 0x580257FF	BDMA2		*Section 16.6: BDMA registers*
0x58024800 - 0x58024BFF	PWR		*Section 6.8: PWR registers*
0x58024400 - 0x580247FF	RCC		*Section 8.7: RCC registers*
0x58022800 - 0x58022BFF	GPIOK		*Section 11.4: GPIO registers*
0x58022400 - 0x580227FF	GPIOJ		*Section 11.4: GPIO registers*
0x58022000 - 0x580223FF	GPIOI		*Section 11.4: GPIO registers*
0x58021C00 - 0x58021FFF	GPIOH		*Section 11.4: GPIO registers*
0x58021800 - 0x58021BFF	GPIOG		*Section 11.4: GPIO registers*
0x58021400 - 0x580217FF	GPIOF		*Section 11.4: GPIO registers*
0x58021000 - 0x580213FF	GPIOE		*Section 11.4: GPIO registers*
0x58020C00 - 0x58020FFF	GPIOD		*Section 11.4: GPIO registers*
0x58020800 - 0x58020BFF	GPIOC		*Section 11.4: GPIO registers*
0x58020400 - 0x580207FF	GPIOB		*Section 11.4: GPIO registers*
0x58020000 - 0x580203FF	GPIOA		*Section 11.4: GPIO registers*
0x58006C00 - 0x580073FF	DFSDM2 (1 filter)		*Section 33.7: DFSDM channel y registers (y=0..7)* and *Section 33.8: DFSDM filter x module registers (x=0..7)*
0x58006800 - 0x58006BFF	DTS		*Section 28.6: DTS registers*

Vale mencionar que na seção "RCC register map" do Manual de Referência, há uma tabela que lista todos os registradores do módulo RCC e seus valores iniciais.

Table 63. RCC register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0B0	RCC_CKGAENR	JTAGCKG	EXTICKG	ECCRAMCKG	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	GFXMMUSCKG	AXIRAM3CKG	AXIRAM2CKG	AXIRAM1CKG	OCTOSPI1CKG	FMCCKG	OCTOSPI2CKG	FLIFTCKG	AHB34CKG	AHB12CKG	GFXMMUMCKG	LTDCCKG	DMA2DCKG	MDMACKG	SDMMCCKG	CPUCKG	AHBCKG	AXICKG
	Reset value	0	0	0												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0B4 to 0x12F	Reserved	Reserved																															
0x130	RCC_RSR	Res.	LPWRRSTF	Res.	WWDGRSTF	Res.	IWDGRSTF	Res.	SFTRSTF	PORRSTF	PINRSTF	BORRSTF	Res.	CDRSTF	Res.	Res.	RMVF	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
	Reset value		0		0		0		0	1	1	1		1			0																
0x134	RCC_AHB3ENR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	GFXMMUEN	OTFD2EN	OTFD1EN	OCTOSPIMEN	Res.	OCTOSPI2EN	Res.	Res.	SDMMC1EN	Res.	OCTOSPI1EN	Res.	FMCEN	Res.	Res.	Res.	Res.	Res.	Res.	JPGDECEN	DMA2DEN	Res.	Res.	Res.	MDMAEN
	Reset value								0	0	0	0		0			0		0		0							0	0				0
0x138	RCC_AHB1ENR	Res.	Res.	Res.	Res.	Res.	USB1ULPIEN	USB1OTGEN	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CRCEN	Res.	Res.	Res.	ADC12EN	Res.	Res.	Res.	DMA2EN	DMA1EN
	Reset value						0	0																0				0				0	0
0x13C	RCC_AHB2ENR	Res.	AHBSRAM2EN	AHBSRAM1EN	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	BDMA1EN	Res.	SDMMC2EN	Res.	Res.	RNGEN	HASHEN	CRYPTEN	Res.	HSEMEN	Res.	DCMI_PSSIEN
	Reset value		0	0																		0		0			0	0	0		0		0
0x140	RCC_AHB4ENR	Res.	Res.	SRDSRAMEN	BKPRAMEN	Res.	Res.	Res.	Res.	Res.	Res.	BDMA2EN	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	GPIOKEN	GPIOJEN	GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
	Reset value			0	0							0											0	0	0	0	0	0	0	0	0	0	0

Com a Porta B ativada, podemos configurar seu pino 0 (PB0) que é controlado pelo módulo GPIO. O capitulo “GPIO” descreve os módulos de GPIO e a seção “GPIO registers” apresenta os registradores. São ao todo 10 registradores. Pode-se dar uma rápida passada nas descrições desses registradores. Na seção 11.4.1 (pág. 516) encontramos o registrador *GPIO port mode register* (GPIOx_MODER), onde *x* corresponde à porta específica (A até K). O trecho de texto extraído do Manual de Referência mostra que um par de *bits* define o modo de operação de cada pino n, que varia de 0 a 15. Os dois *bits* (n/2+1,n/2) configuram o modo de operação do pino n. Os *bits* 1 e 0 correspondem ao pino 0.

11.4.1 GPIO port mode register (GPIOx_MODER) (x =A to K)

Address offset:0x00

Reset value: 0xABFF FFFF for port A

Reset value: 0xFFFF FEBF for port B

Reset value: 0xFFFF FFFF for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **MODER[15:0][1:0]:** Port x configuration I/O pin y (y = 15 to 0)
These bits are written by software to configure the I/O mode.
00: Input mode
01: General purpose output mode
10: Alternate function mode
11: Analog mode (reset state)

Vendo a descrição dos *bits*, percebe-se que para que o pino funcione como saída digital, é necessário que o valor do par de *bits* seja “01” (“*General purpose output mode*”). É necessário

garantir que o *bit* 0 seja “1” e que o *bit* 1 seja “0”. Pode-se atribuir ((GPIOB_MODER *OR* 0x00000001) *AND* 0xFFFFFFFD) a GPIOB_MODER. Além disso, observe que o código “10” configura o pino para uma função alternativa além de GPIO ou analógica que pode ser consultada no *Datasheet*; a própria função é especificada nos registradores GPIOx_AFRL e GPIOx_AFRH. O endereço de GPIOB_MODER pode ser determinado somando o endereço-base do módulo GPIOB, que é 0x58020400, com o seu deslocamento, 0x00, em relação ao endereço-base. Isso resulta no endereço final de 0x58020400.

Ainda precisamos definir o tipo de saída para o pino (*push-pull* ou *open drain*). Um pino *push-pull*, também conhecido como pino de saída de potência complementar, é um tipo de saída de circuitos integrados que utiliza dois transistores complementares (um NPN e um PNP ou um NMOS e um PMOS para dirigir a saída. Um pino *open drain*, conhecido como pino de dreno aberto ou de coletor aberto, é um tipo de saída que utiliza apenas um transistor (NMOS ou NPN) para conduzir a carga, necessitando de um resistor externo para puxar a tensão para o nível alto quando o transistor não está conduzindo.

A configuração do tipo de saída é feito através do registrador *GPIO port output type register* (GPIOx_OTYPER), como mostra o trecho de texto extraído do Manual de Referência. Cada *bit* de 0 até 15 determina o tipo de saída para o pino de número correspondente. Se o *bit* é “0”, a saída é *push-pull (estado assumido no reset)*. Se o *bit* é “1”, a saída é *open drain*. No nosso exemplo, precisamos usar uma saída em *push-pull*, e, portanto, o *bit* 0 do registrador deve ser “0”. Analogamente ao processo de determinar o endereço de GPIOB_MODER, o endereço de GPIOB_OTYPER é obtido somando-se o endereço-base do módulo GPIOB, 0x58020400, com o seu deslocamento, 0x04, resultando em 0x58020404.

11.4.2 GPIO port output type register (GPIOx_OTYPER) (x = A to K)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **OT[15:0]:** Port x configuration I/O pin y (y = 15 to 0)
These bits are written by software to configure the I/O output type.
0: Output push-pull (reset state)
1: Output open-drain

Para que o pino seja colocado em nível lógico “0” ou “1”, pode-se usar o *GPIO port output data register* (GPIOx_ODR), como mostra o tracho de texto extraído do Manual de Referência. Seus *bits* de 0 a 15 determinam o nível lógico do pino correspondente, se o mesmo estiver configurado como saída. No caso, vamos acender o LED colocando o *bit* 0 em “1” e apagá-lo colocando o *bit* 0 em “0”. Da mesma forma, o endereço de GPIOB_ODR é calculado somando-se 0x58020400 com 0x14, o que resulta em 0x58020414.

11.4.6 GPIO port output data register (GPIOx_ODR) (x = A to K)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODR[15:0]:** Port output data I/O pin y (y = 15 to 0)
These bits can be read and written by software.
Note: For atomic bit set/reset, the ODR bits can be individually set and/or reset by writing to the GPIOx_BSRR register (x = A..F).

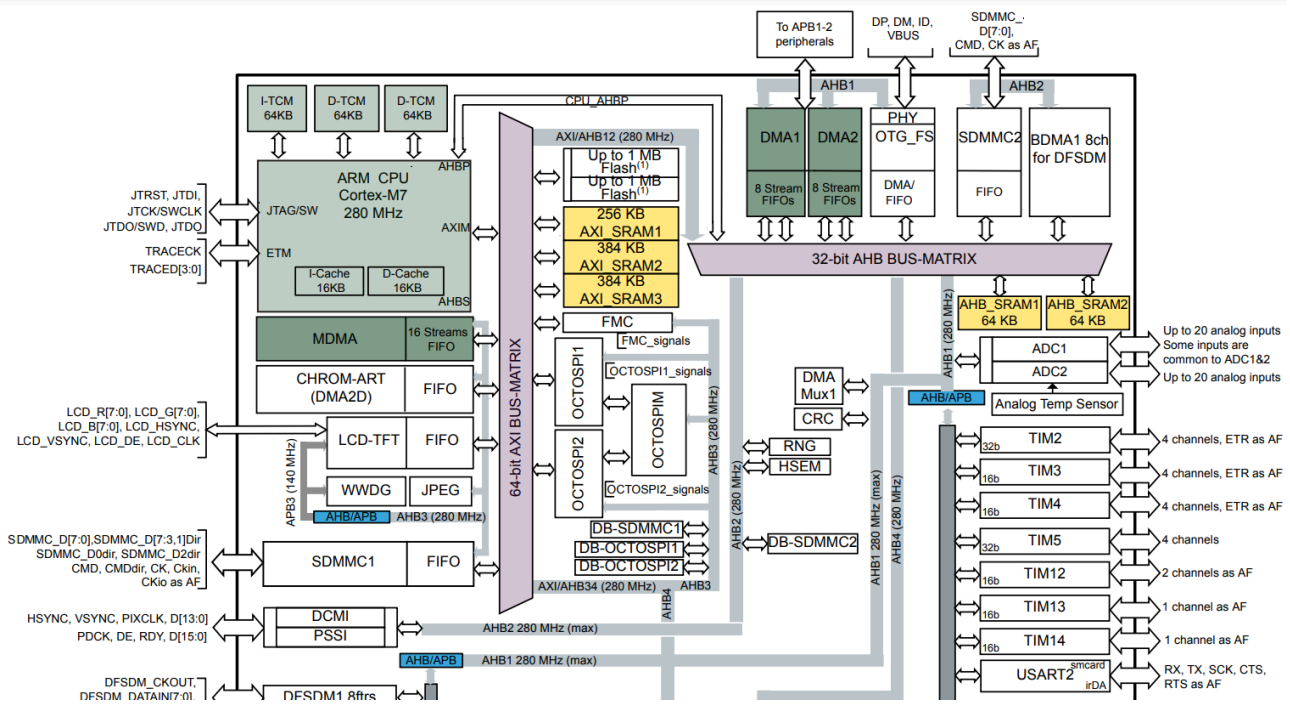
PROGRAMAÇÃO *BARE-METAL* DO NÚCLEO CORTEX-M7

Existem nos microcontroladores uma série de registradores que controlam o funcionamento dos diversos módulos integrados, como portas GPIO, conversores analógico-digital, temporizadores, UARTs, entre outros. É possível configurar manualmente cada periférico manipulando diretamente nos registradores de controle e de dados. Isso envolve, com o microcontrolador devidamente alimentado e inicializado, escrever *bits* nos registradores para definir modos de operação, tempos, pinos de entrada/saída e outros parâmetros necessários. Com o avanço tecnológico, os microcontroladores se tornaram complexos, incorporando mais funcionalidades e periféricos, tornando a tarefa tediosa e propensa a erros.

Sendo os registradores da maioria dos microcontroladores mapeados no espaço de endereçamento (memória) do processador integrado, uma abordagem alternativa e eficiente é escrever uma sequência de instruções que configure os periféricos conforme necessário. Essa sequência de instruções, conhecida como código-fonte, é compilada e armazenada na memória do microcontrolador. Quando executado, o processador integrado (CPU) busca (*fetch*), decodifica (*decode*) e executa (*execute*) as instruções sequencialmente em cada ciclo de instrução. Isso simplifica o controle dos periféricos, pois a programação abstrai os detalhes de baixo nível dos registradores, tornando-o menos propenso a erros e mais adaptável à complexidade crescente dos microcontroladores modernos, pois o acesso aos registradores de periféricos resume-se a leituras e escritas em posições específicas de memória.

Nessa perspectiva, é essencial que o desenvolvedor tenha conhecimento do modo de operação do processador, incluindo seu repertório de instruções, para programar o código-fonte. Esse tipo de programação, que interage diretamente com os registradores, é conhecida como programação *bare-metal*. A tradução literal do termo seria algo como “Metal Nu”, e seu conceito é realizar a programação de periféricos da forma mais próxima possível do *hardware*, sem camadas adicionais de abstração, como sistemas operacionais ou bibliotecas complexas. O programa resultante é denominado *firmware* ou *software embarcado*, para distingui-lo do software tradicional, que não está diretamente incorporado ao *hardware*.

O processador integrado no microcontrolador STM32H7A3 é o núcleo ARM Cortex-M7 pertencente à família Cortex-M. A figura extraída do *Datasheet* mostra, em diagrama de blocos, a interconexão desse processador, destacado em verde claro, com o restante dos módulos do microcontrolador. A arquitetura ARM Cortex-M7 define um conjunto específico de instruções, modos de endereçamento, e características de *hardware*, que compõem a ISA (do inglês *Instruction Set Architecture*) para a interação entre o processador e o *software*.



O Cortex-M7 utiliza a ISA ARMv7E-M de 32 *bits*, que suporta os modos de execução *Thread* e *Handler* em conformidade com os padrões de sistemas embarcados para uso em ambientes críticos. Essa distinção entre os dois modos de execução permite um controle granular sobre a execução do código, a gestão de recursos e a resposta a eventos em tempo real. No modo *Thread*, o processador executa tarefas normais de usuário, como a execução de código de aplicativos. Já no modo *Handler*, o processador lida com exceções e interrupções. O processador sempre inicia no modo *Thread*. Quando uma exceção ocorre, o processador transita do modo *Thread* para o modo *Handler* para tratar a exceção. Ao entrar no modo *Handler*, o processador suspende temporariamente a execução normal do modo *Thread* para priorizar e resolver a condição de exceção. Isso garante que eventos críticos sejam tratados imediatamente, mantendo a integridade e a estabilidade do sistema, retornando ao modo *Thread* após o tratamento ser concluído.

Durante a execução de um programa, há distinção entre os níveis privilegiado e não-privilegiado. A arquitetura suporta duas pilhas, garantindo que cada nível de privilégio tenha sua própria pilha para gerenciar corretamente exceções e interrupções. No nível privilegiado, o processador tem acesso completo a todos os recursos do sistema, incluindo registradores especiais, instruções privilegiadas e permissões para manipular o sistema como um todo. A pilha principal (MSP do inglês *Main Stack Pointer*) é utilizada para armazenar informações críticas do sistema, como contexto de execução do sistema operacional e dados de interrupção. No nível não-privilegiado, o acesso a recursos críticos e instruções privilegiadas é restrito. A pilha de processo (PSP do inglês *Process Stack Pointer*) é

utilizada para gerenciar variáveis locais e parâmetros de função durante a execução de código de aplicação. Em sistemas embarcados, é comum que a maior parte do tempo de execução do programa ocorra no modo *Thread* a nível não-privilegiado, garantindo assim a segurança e a estabilidade do sistema. Operações que exigem acesso privilegiado são geralmente realizadas por meio de interrupções ou exceções que mudam temporariamente para o modo *Handler*, onde o processador opera a nível privilegiado para garantir uma resposta eficiente e controlada a eventos de exceção. A tabela extraída do Manual de Programação sintetiza a relação entre os modos de execução (*Thread* ou *Handler*), os níveis de privilégio (privilegiado ou não-privilegiado) e o uso das pilhas (MSP ou PSP).

Table 1. Summary of processor mode, execution privilege level, and stack use options

Processor mode	Used to execute	Privilege level for software execution	Stack used
Thread	Applications	Privileged or unprivileged(1)	Main stack or process stack(1)
Handler	Exception handlers	Always privileged	Main stack

Em termos de instruções, processadores Cortex-M suportam três modos de codificação de instruções: **ARM**, **Thumb** e **Thumb-2**. No estado ARM, as instruções são codificadas em 32 *bits* completos, o que é ideal para operações complexas que exigem manipulação extensiva de dados ou operações de ponto flutuante. Por outro lado, o estado Thumb introduz instruções de 16 *bits* mais compactas, otimizando o tamanho do código e oferecendo uma execução mais rápida em algumas situações. Já o estado Thumb-2 expande ainda mais as capacidades do Thumb, adicionando novas instruções de 16 e 32 *bits*, permitindo uma mesclagem de instruções mais flexível em comparação com o Thumb original. Isso resulta em melhorias de desempenho mantendo eficiência de tamanho de código. Apesar de ser baseado em uma arquitetura de 32 *bits*, o Cortex-M7 implementa uma versão do Thumb baseada na tecnologia Thumb-2 e opera somente no estado Thumb, como descreve o trecho extraído do Manual de Programação.

Thumb state

The Cortex®-M7 processor only supports execution of instructions in Thumb state. The following can clear the T bit to 0:

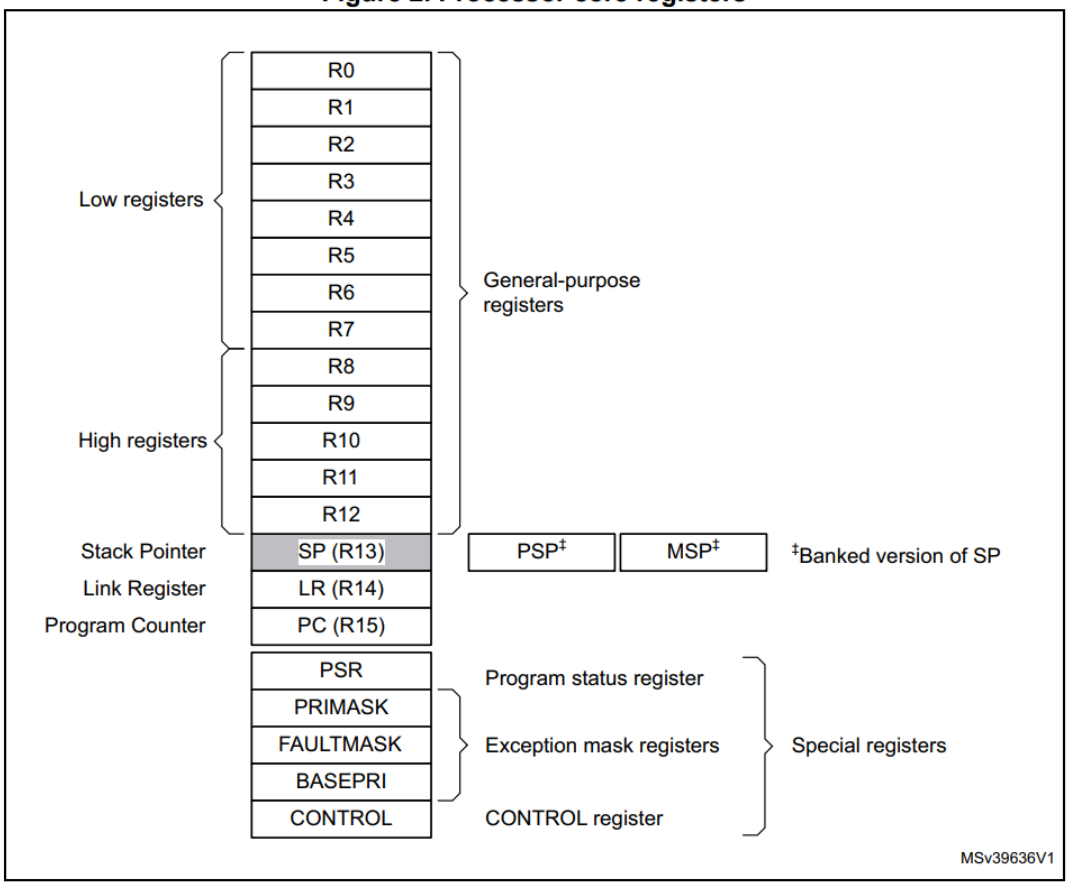
- Instructions BLX, BX and POP{PC}.
- Restoration from the stacked xPSR value on an exception return.
- Bit[0] of the vector value on an exception entry or reset.

Attempting to execute instructions when the T bit is 0 results in a fault or lockup. See *Lockup on page 49* for more information.

No modelo de programação do Cortex-M7, é adotado um paradigma baseado em registradores conhecido como arquitetura *load-store*. Todas as operações de dados são realizadas exclusivamente entre os 13 registradores de propósito geral de 32 bits (R0 – R12) no modo de endereçamento imediato e por registrador. Os primeiros oito registradores, R0 – R7, são frequentemente referidos como "registradores baixos". Devido às limitações de codificação das instruções de 16 *bits*, muitas delas só podem acessar esses registradores baixos. Já os registradores R8 – R12, conhecidos como

"registradores altos", podem ser utilizados tanto por instruções de 32 *bits* quanto por algumas instruções de 16 *bits*, como a MOV. Os valores iniciais dos registradores R0 a R12 não são especificados na inicialização do processador.

Figure 2. Processor core registers



O acesso a periféricos e memória no Cortex-M7 ocorre através de instruções dedicadas de *load* (carregar) e *store* (salvar). Devido às limitações na codificação de endereços efetivos dos operandos, utiliza-se o modo de endereçamento indexado para especificar esses endereços. No modo de endereçamento indexado, o endereço efetivo é calculado somando-se um deslocamento ao conteúdo de um registrador, como o contador de programa (PC) ou o ponteiro de pilha (SP), conforme especificado na instrução. Neste caso, codifica-se apenas a identificação do registrador especial e o deslocamento na instrução. O repertório completo de instruções do Cortex-M pode ser consultado no capítulo 3 do Manual de Programação.

Para ampliar a faixa de deslocamentos representáveis, duas estratégias são aplicadas: primeiro, no modo *Thumb*, onde os endereços são sempre pares, o *bit* menos significativo não é codificado, pois é sempre 0. Segundo, como o PC é incrementado automaticamente em 4 após a execução de um par de instruções de 16 *bits*, os endereços de desvio são sempre múltiplos de 4, resultando nos dois *bits* menos significativos sempre sendo 0 e, portanto, não precisando ser codificados.

O Manual de Programação mostra que o Cortex-M7 é capaz de lidar com tipos de dados inteiros, com ou sem sinal, de 8 *bits* (*byte*), 16 *bits* (*halfwords*), 32 *bits* (*words*), e valores em ponto flutuante de precisão simples (32 *bits*) e precisão dupla (64 *bits*).

A ARM disponibiliza o padrão CMSIS (do inglês *Cortex Microcontroller Software Interface Standard*) para simplificar o desenvolvimento de *firmware*. Esse padrão proporciona uma camada de abstração que uniformiza o acesso aos periféricos, ao núcleo do processador e aos recursos do sistema, independentemente do fabricante do microcontrolador. Com o CMSIS, os desenvolvedores podem escrever código que é facilmente portável entre diferentes microcontroladores Cortex-M7 de diversos fabricantes, utilizando uma API comum para acessar periféricos como UART, SPI, I2C, temporizadores, ADC, entre outros. Além disso, o CMSIS oferece acesso direto aos registradores do núcleo do Cortex-M7 por meio de nomes padronizados, permitindo configurações detalhadas e controle preciso sobre operações críticas.

STM32CubeIDE

O STM32CubeIDE é o ambiente de desenvolvimento de *software* desenvolvido pela ST Microelectronics para os microcontroladores por ela desenvolvidos. Este IDE (*Integrated Development Environment*) foi desenvolvido a partir do **Eclipse**.

Eclipse é um ambiente de desenvolvimento multi-linguagem que contém um IDE e um sistema de *plug-ins*. Foi lançado nos termos da *Eclipse Public License*, assim o ambiente Eclipse é gratuito e de código aberto. Vem se tornando um padrão cada vez mais utilizado em empresas que desenvolvem projetos de *software*.

Os *plug-ins* garantem funcionalidade expansível ao sistema, sem a necessidade de efetuar alterações nele. Assim, pode-se adicionar novas linguagens de programação, bem como ferramentas de teste e depuração de código, integradas ao ambiente. Pode-se ainda adicionar funcionalidades relativas a famílias específicas de processadores/controladores. Por exemplo, quando um fabricante de microcontroladores lança uma nova família, também lança um *plug-in* para o IDE usando os serviços oferecidos pelo próprio IDE, para que este possa gerar código objeto corretamente para aquela nova família. Outro exemplo é o *plug-in* de terminal serial, que permite a implementação de um terminal para comunicação serial dentro do ambiente, não sendo mais necessário o uso de outro programa para esta finalidade no computador *host*. Assim, quando se depura um programa que se comunica com um computador através de porta serial, antes era necessário ficar alternando entre as janelas de depuração do IDE e do terminal serial. Agora, as janelas de depuração e a do terminal ficam integradas no mesmo ambiente.

Um conceito fundamental do Eclipse é o ***workspace***, que é um conjunto de metadados sobre um espaço (de arquivos) de trabalho. Na prática, podemos ter *workspaces* diferentes no mesmo computador, e em cada *workspace* podemos ter múltiplos projetos. O sistema permite importar e exportar projetos individuais ou *workspaces* completos. Neste curso, podemos definir um *workspace* para cada usuário, o qual pode colocar todos seus projetos dentro do mesmo. Assim, na mesma máquina podem conviver espaços de trabalho distintos.

Outro conceito fundamental é a **perspectiva**. Esta pode ser definida como uma interface composta de um conjunto de janelas para melhor utilização do sistema, de acordo com o que está sendo feito naquele momento. Neste curso, vamos usar 3 perspectivas básicas: **Inicialização**, **Programação** e **Depuração**. Na perspectiva de inicialização, são definidos os periféricos internos do microcontrolador a serem utilizados e seus parâmetros de inicialização. Na perspectiva de programação, temos janelas para edição de código, árvore de arquivos do projeto, localização de módulos de código, recepção de mensagens de compilação, etc. Na perspectiva de depuração, temos janelas para visualização de ponto de execução, controles para execução passo-a-passo do código, janelas para apresentação de variáveis em tempo real, código *assembly*, ou linguagem de montagem, gerado a partir do código fonte, janela do terminal serial, etc.

Dois *plug-ins* serão utilizados ao longo do curso, "Eclox" e "TM Terminal". Eclox é um *front-end* do aplicativo **Doxygen** de documentação de código para Eclipse, fornecendo uma integração simples e elegante do processo de documentação de código no Eclipse e uma interface gráfica para configuração da geração de documentação. TM Terminal é, por sua vez, um terminal de linhas de comando integrável no Eclipse. Para instalar esses dois plugins, use o "*Eclipse Marketplace*" que está agrupado no item "*Help*" da barra de ferramentas superior. Procure os *plug-ins* pelas palavras "Eclox" e "TM Terminal" e instale-os.

Estes conceitos se desenvolverão melhor com o uso da ferramenta. Seguindo o passo-a-passo das próximas páginas, os alunos estarão usando adequadamente a ferramenta.

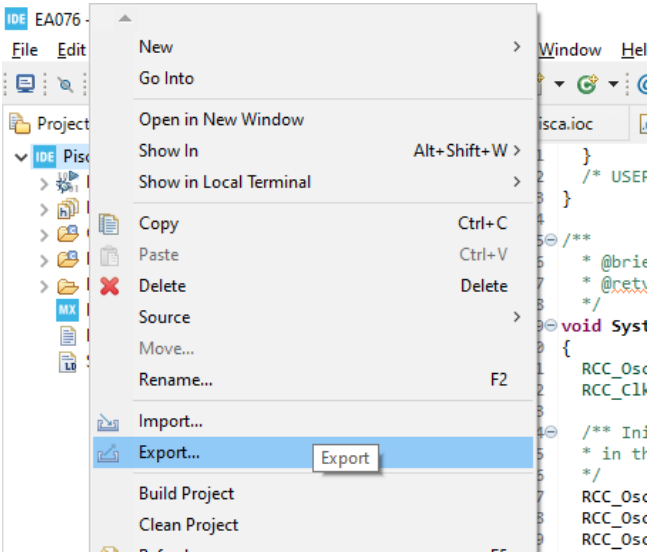
Obs: A aparência de algumas janelas pode ser ligeiramente diferente em alguns computadores e ao longo das novas versões do IDE, mas as diferenças não são suficientes para prejudicar este tutorial.

Exportando/Importanto um projeto

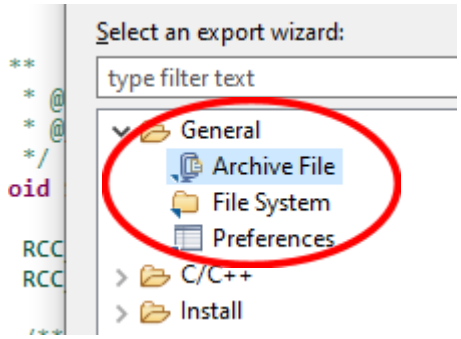
O STM32CubeIDE suporta a importação e exportação de projetos, operações essenciais para o gerenciamento e colaboração em projetos de desenvolvimento. Essas funcionalidades facilitam o trabalho em equipe e garantem a continuidade do desenvolvimento. Seguem-se os procedimentos para importação e exportação.

1. Na perspectiva de programação, feche todos os projetos abertos (conforme visto anteriormente), exceto aquele que se deseja exportar. Se o projeto desejado está fechado, abra-o, clicando sobre a pasta dele com o botão direito e selecionando a opção "*Open Project*". Para exportar apenas arquivos necessários à geração de um executável, limpe o projeto com "*Clean …*".

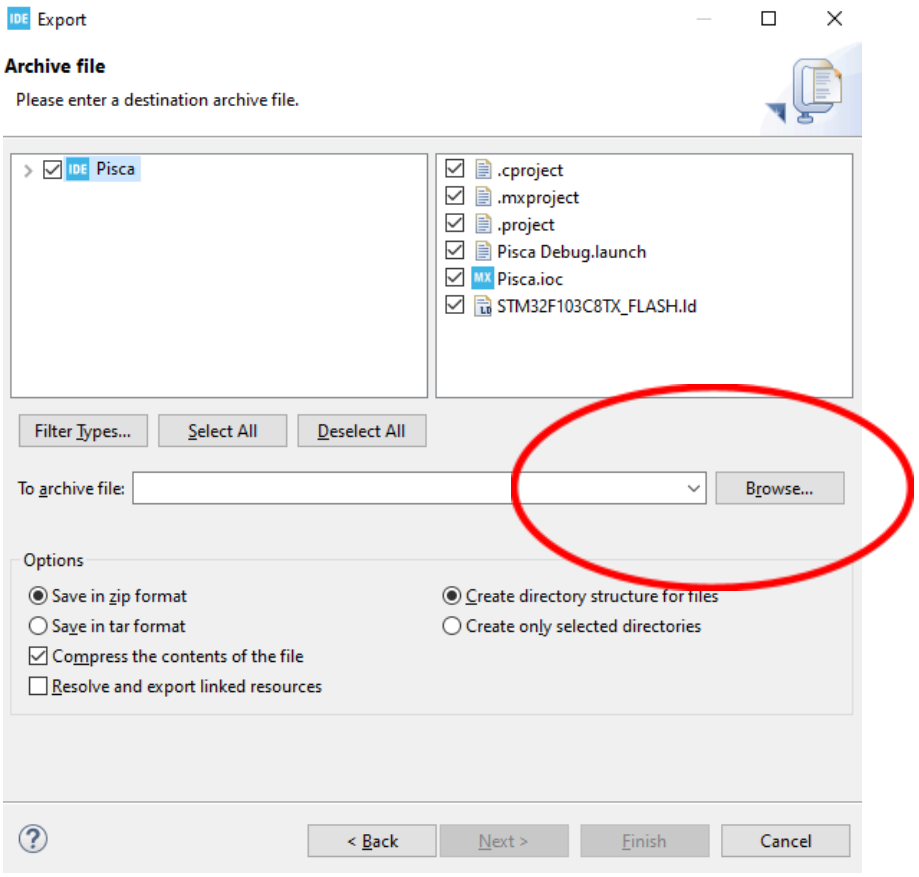
2. Clique novamente com o botão direito na pasta do projeto e selecione "*Export…*". Alternativamente, pode-se usar o menu "*File*", opção "*Export…*".



3. Uma janela se abrirá. Abra o grupo "*General*" e selecione "*Archive File*". Depois, clique em "*Next*".

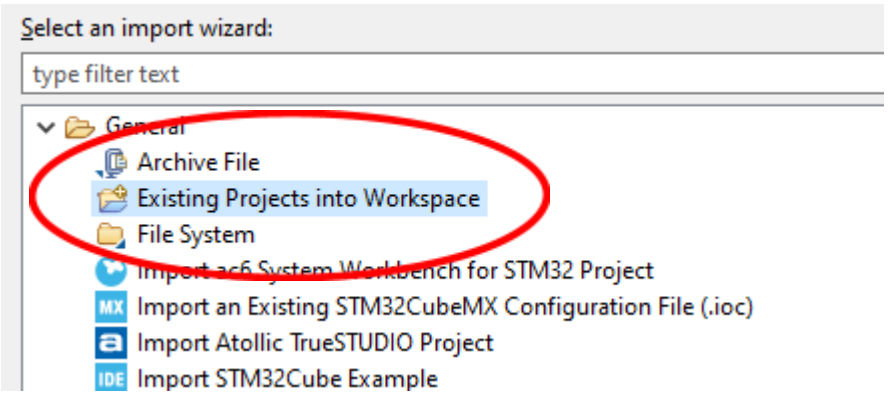


4. Nesta janela, mantenha as opções padrão, e clique no botão "*Browse…*". Selecione uma pasta para guardar o arquivo de exportação, e o nome do arquivo. Clique em "*Finish*".



5. Um arquivo ZIP com o nome escolhido será gravado no local selecionado. Para fazer a importação, apague totalmente o projeto, inclusive os arquivos originais (conforme a seção "Apagando um projeto"). Se um projeto com o mesmo nome de outro projeto no mesmo *workspace* tentar ser importado, aparecerá uma mensagem de erro.

6. Para importar, clique com o botão direito na área dos projetos e selecione "*Import...*", ou selecione "*Import Project*" na janela "*Commander*", ou ainda use o menu "*File – Import...*". Escolha a opção "*Existing Projects into Workspace*" dentro de "*General*".

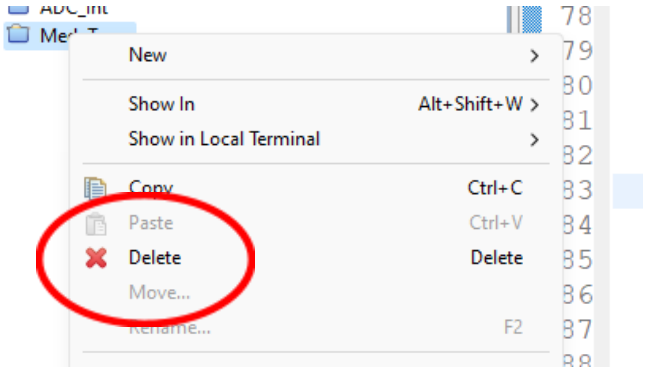


7. Marque a opção "*Select archive file*" e clique no botão "*Browse*". Localize e selecione o arquivo ZIP que contém o projeto exportado anteriormente. Confirme que o projeto está selecionado e clique em "*Finish*".

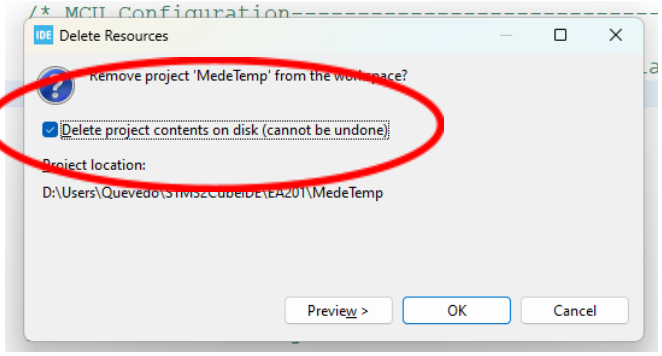
Apagando um projeto

Para manter seu ambiente de desenvolvimento organizado e livre de projetos desnecessários, é importante saber como remover projetos no STM32CubeIDE de forma eficiente. Segue-se um procedimento.

1. Selecione o projeto desejado clicando sobre o mesmo (não importa se o projeto está aberto ou fechado). Depois, use a tecla "*Delete*" no computador. Alternativamente, clique com o botão direito sobre o projeto e selecione a opção "*Delete*".



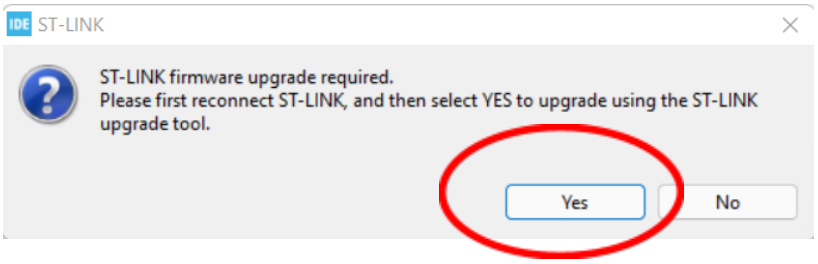
2. Na caixa de diálogo que aparece, selecione a opção "*Delete project contents on disk*" e clique no botão OK. Sem marcar a opção, os arquivos do projeto permanecerão no *workspace*, mas não aparecerão na lista.



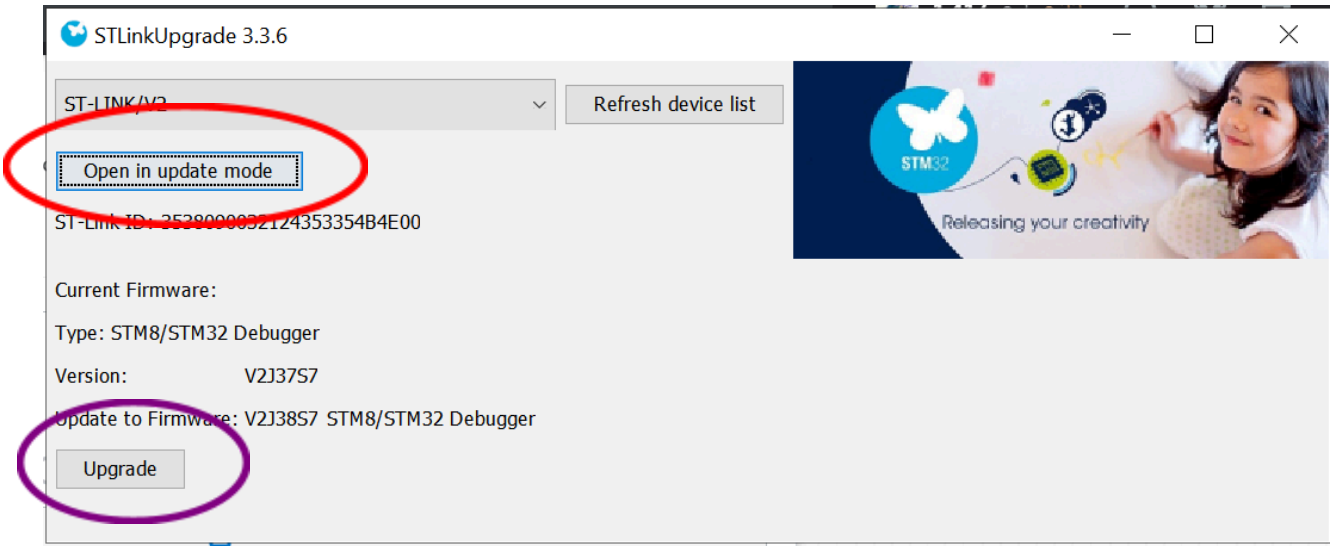
Atualizando o *firmware* do ST-LINK

O ST-LINK fornece uma interface entre o microcontrolador STM32H7A3ZIT6-Q e o ambiente de desenvolvimento STM32CubeIDE, permitindo a programação e depuração do microcontrolador por um outro computador, conhecido por *host*. Atualizar o *firmware* do ST-LINK garantirá a compatibilidade com as últimas ferramentas e melhorias, além de assegurar a estabilidade e o desempenho do sistema. Segue-se um procedimento de atualização.

1. Quando há uma atualização do *firmware* do ST-LINK, ao iniciar o *debug* de um projeto o IDE não fará a carga do programa. Em vez disso, aparecerá a seguinte janela, pedindo para atualizar o *firmware*:



2. Para realizar a atualização, inicialmente clique no botão "Yes". Uma nova janela vai aparecer (**Obs:** Nesta janela, os textos referentes a "*Version:*" e "*Update to firmware:*" serão diferentes do que a figura apresenta, referindo-se à versão atual e à nova versão do *firmware*):



3. Para que se possa colocar o ST-LINK em modo de atualização, desconecte-o da porta USB, aguarde alguns segundos e conecte novamente. Depois clique no botão "*Open in update mode*", conforme marcado pela elipse vermelha na figura acima.

4. Depois que a janela atualizar, clique no botão "*Upgrade*", conforme marcado pela elipse roxa na figura acima.

5. Aparecerá uma janela com uma barra de progressão e uma mensagem de atualização do *firmware*. Ao aparecer a mensagem de "Concluído", basta repetir o procedimento de desconectar e reconectar o ST-LINK da USB, para que o mesmo volte ao modo normal de operação, já com o *formware* atualizado.

Responda ao questionário da aula disponível no Moodle e não se esqueça de implementar as modificações propostas em cada projeto antes de responder o questionário.