INTRODUÇÃO AO STM32CUBEIDE PARA A DISCIPLINA EA801

Prof. Antonio A. F. Quevedo

FEEC / UNICAMP

Revisado em fevereiro de 2025

INTRODUÇÃO

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. 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, 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. 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 um conjunto de janelas que povoam seu ambiente de trabalho, conjunto este definido 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* gerado a partir do código fonte, janela do terminal serial, etc.

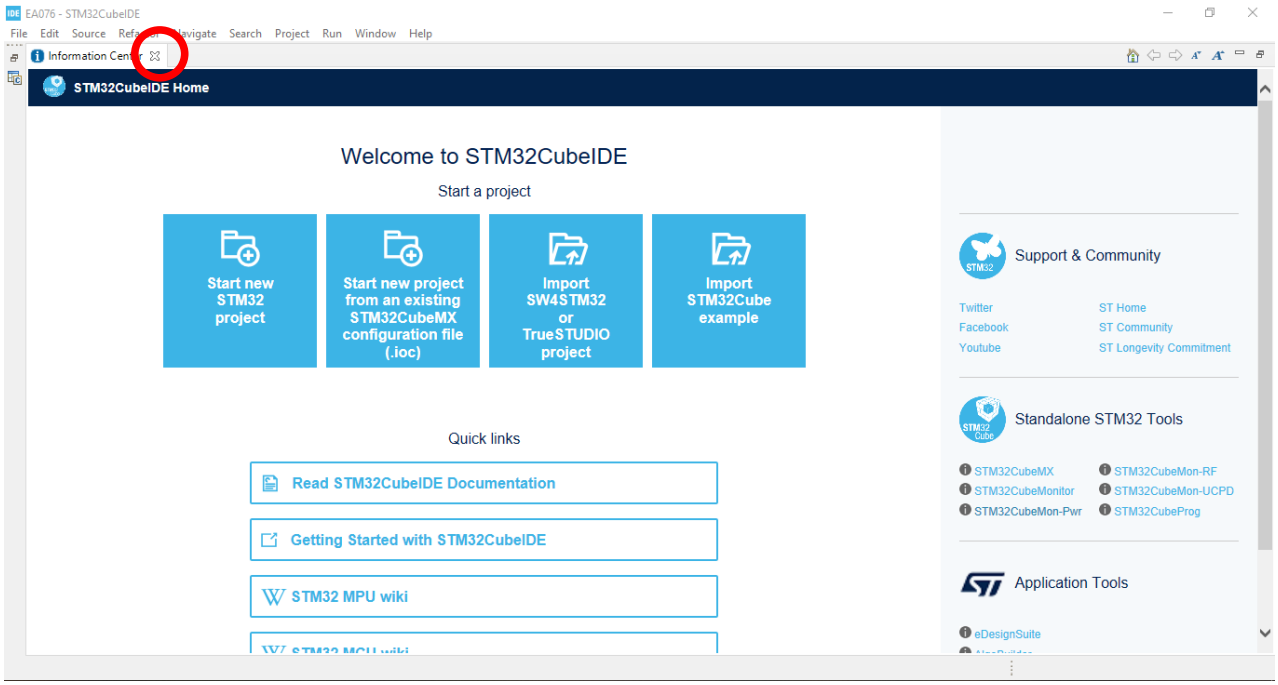
Estes conceitos se desenvolverão melhor com o uso da ferramenta. Seguindo o passo-a-passo das próximas páginas, certamente os alunos estarão usando adequadamente a ferramenta em um curto espaço de tempo.

Obs: A aparência de algumas janelas pode ser ligeiramente diferente em alguns computadores, mas as diferenças não são suficientes para prejudicar este tutorial.

CRIANDO UM NOVO PROJETO

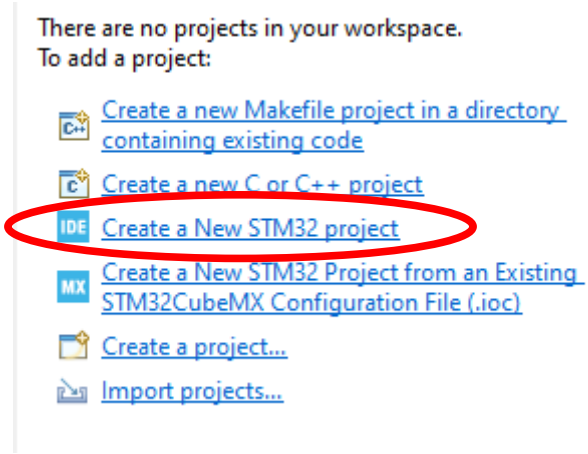
1. Após instalar o programa e iniciá-lo, será perguntado o diretório para o *workspace* que deseja utilizar. Há ainda a opção de usar sempre este *workspace* sem precisar perguntar a cada inicialização.

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.

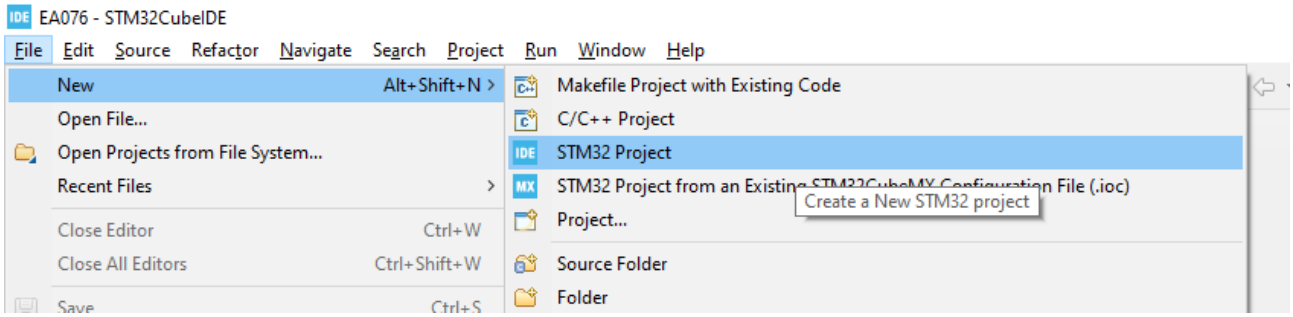


3. O IDE vai abrir na perspectiva de programação. Inicialmente no painel à esquerda aparece o painel do “Project Explorer”, com várias opções de uso frequente.

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



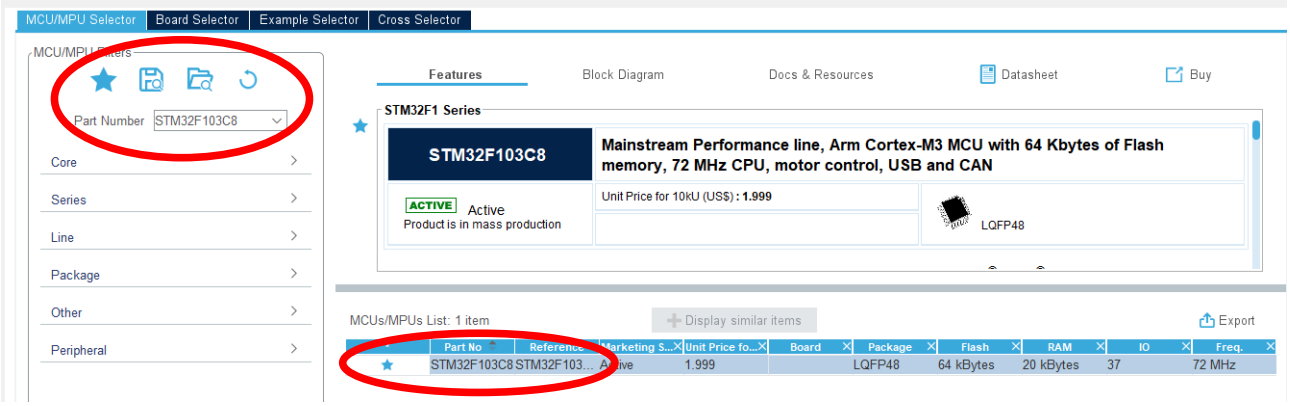
Alternativamente, no menu superior selecione *File – New – STM32 Project*.



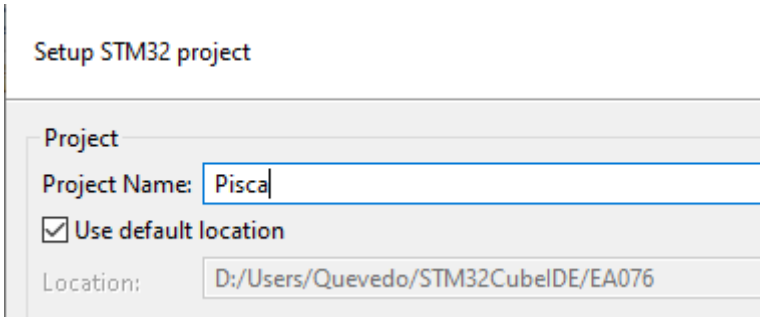
5. Após alguns segundos, uma nova janela abre, chamada “Target Selection”. Nela vamos definir o modelo de microcontrolador que vamos utilizar. À esquerda temos uma série de filtros que

podemos usar, para seleção do núcleo de processador ARM, série, linha, tipo de encapsulamento, etc. Entretanto, a forma mais fácil de selecionar seu microcontrolador é buscando por texto, no campo “Part Number”. Neste campo, digite “STM32F411CEU6”, e verá, a medida que digita, as opções à esquerda e embaixo se reduzindo até restar apenas duas, uma com um sufixo “TR” adicional. Clique sobre a linha com o nome da MCU **sem** o sufixo “TR” para selecionar o modelo.

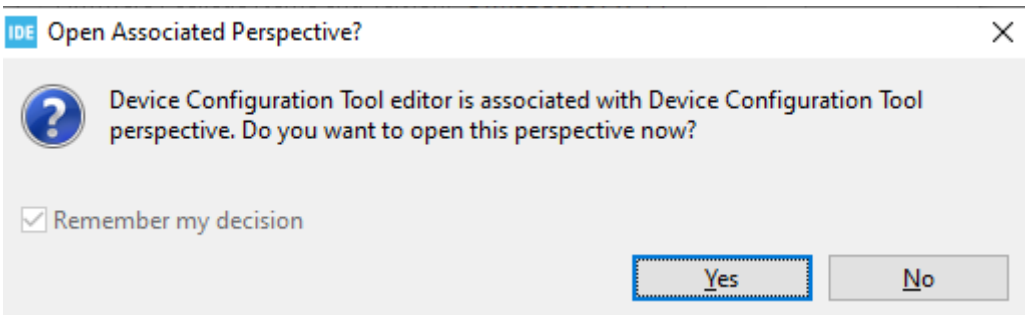
À esquerda do nome, há uma estrela com o interior branco. Ao clicar nela, você inclui este microcontrolador na sua lista de favoritos (a estrela fica azul). Depois, você pode ir diretamente à lista de favoritos clicando na estrela acima do campo “Part Number”.



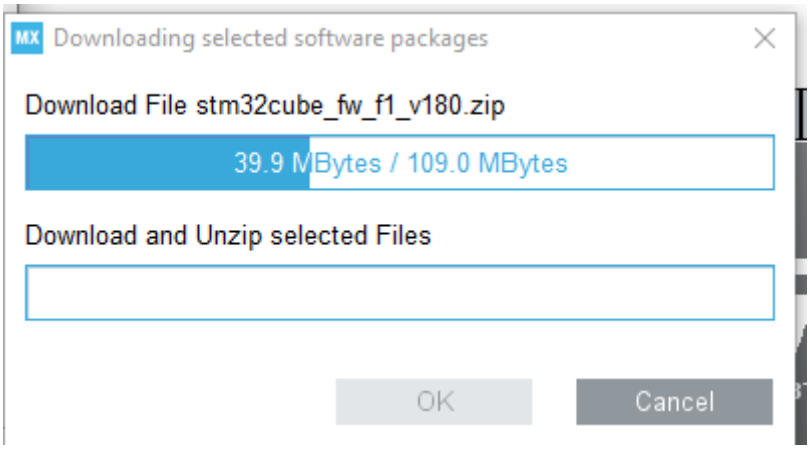
6. Ao clicar no botão “Next”, aparece a janela na qual se dá nome ao projeto. Digite “Pisca” no campo de “Project Name”. As demais opções desta e da próxima janela serão as opções padrão, assim basta clicar em “Finish” para que o projeto seja criado.



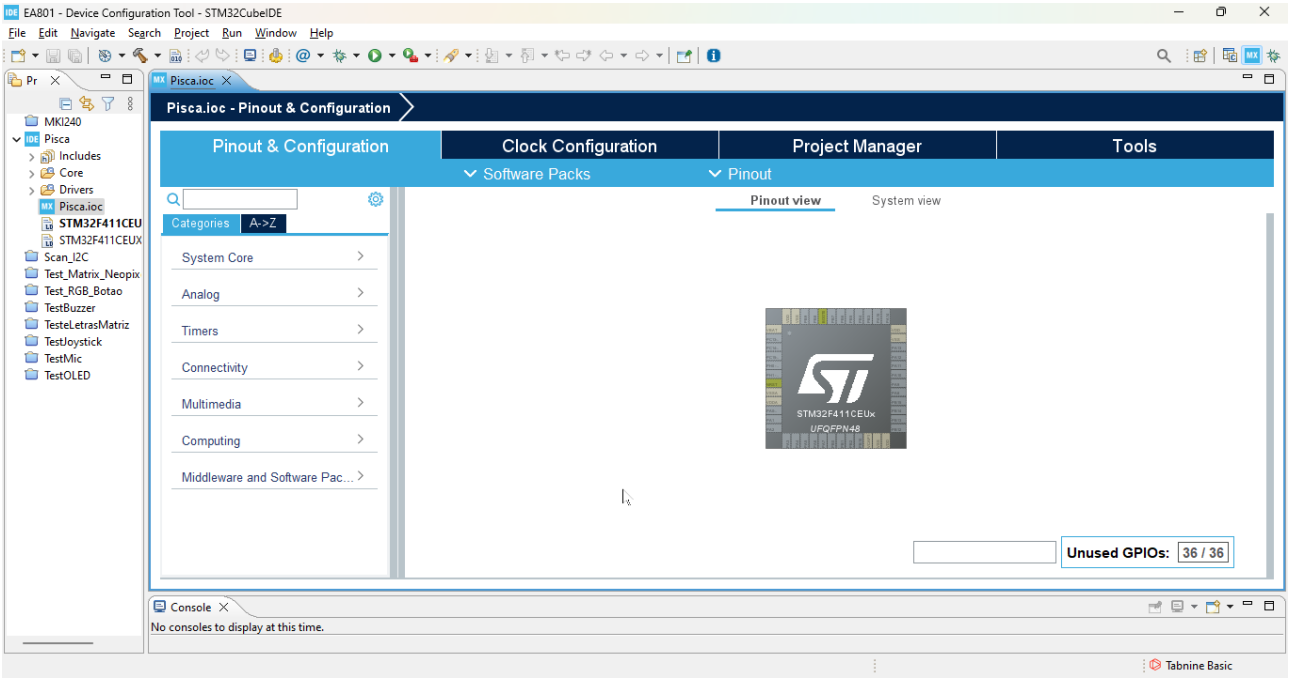
Deve aparecer uma janela recomendando mudar a perspectiva. Clique no botão “Yes” (apenas na primeira vez que se usa o IDE, mantendo a opção “Remember my decision” marcada).



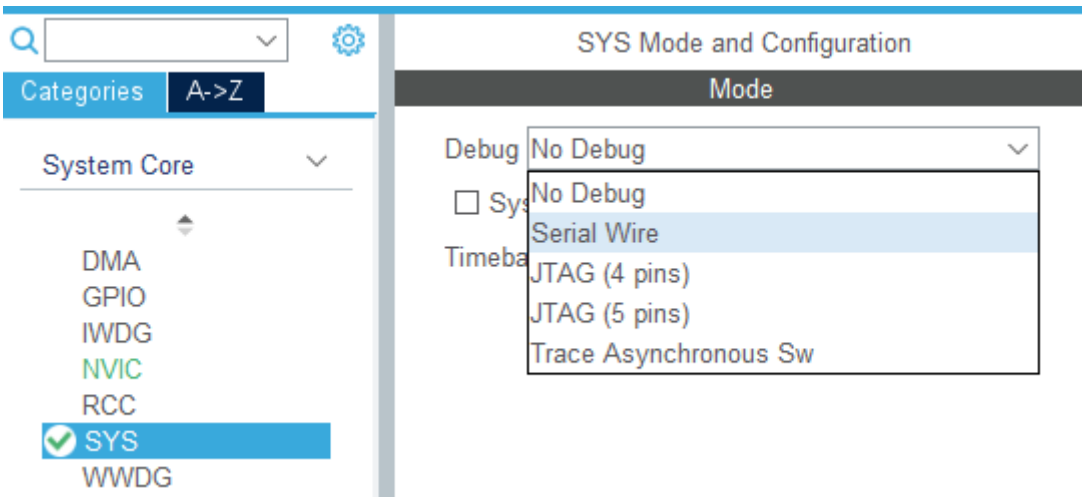
Na primeira vez que uma determinada família de microcontroladores é usada, o IDE baixa os pacotes de suporte àquela família:



Após a instalação dos pacotes de suporte, o IDE entra na perspectiva de Inicialização. A aba mostra o arquivo “Pisca.ioc”, representado de forma gráfica. Dominando a tela, aparece uma representação física do microcontrolador, com controles de zoom. À esquerda, há um painel com uma lista dos diversos periféricos do microcontrolador, bem como eventuais algoritmos de cálculo e recursos de *Middleware* (sistemas de arquivos, dispositivos USB, etc).

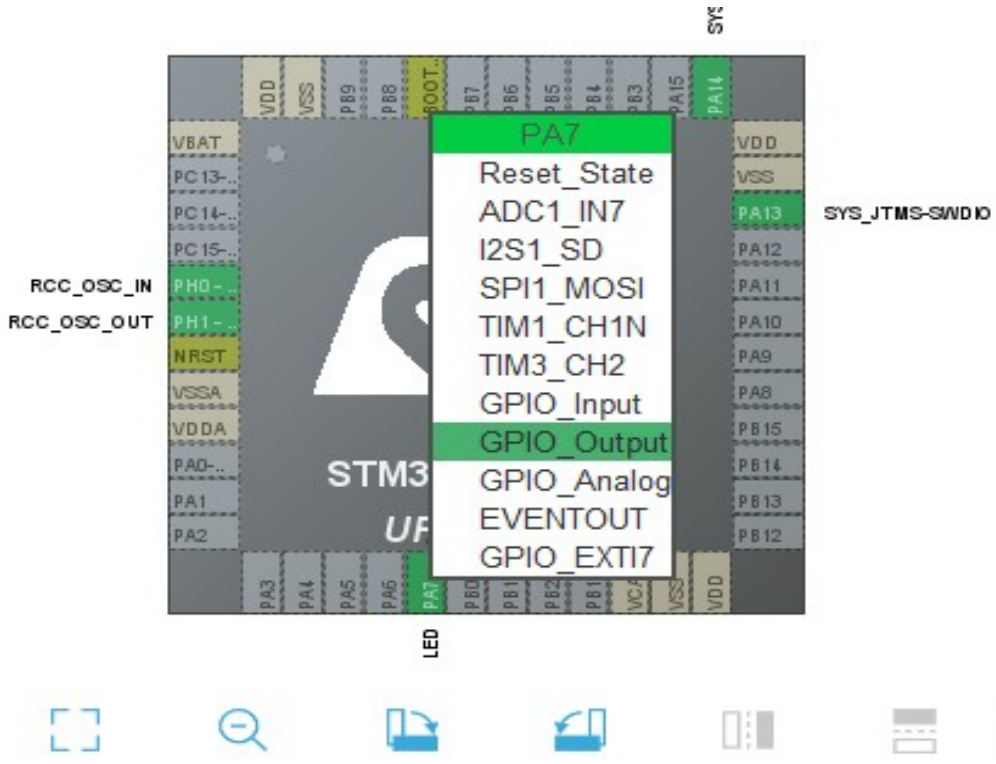


7. Dê um zoom na imagem do circuito integrado (lupa com símbolo +) para que ele preencha a tela. Pode também clicar com o mouse sobre ele e arrastar para ajustar a posição. Vamos começar a selecionar os periféricos que vamos utilizar. Clique sobre o item “System Core” e verá um novo conjunto de itens abrir. Selecione “SYS” e um novo painel irá abrir. Precisamos ativar o “debug” do microcontrolador, e para isso na lista do item “Debug”, selecione “Serial Wire”

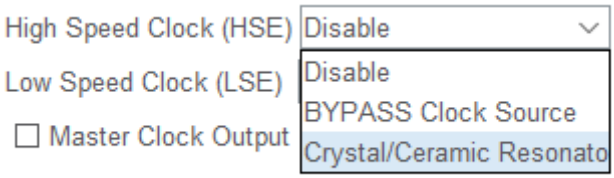


Assim, o modo SWD de *debug* é ativado (o modo usado pelo depurador ST-LINK). Pode-se ver que dois dos pinos da imagem do microcontrolador (PA13 e PA14) agora aparecem em verde e com "Labels" identificando as funções selecionadas. Estes pinos são ligados no conjunto de 4 pinos para ligação no ST-LINK (os outros dois são o GND e a alimentação de 3.3V).

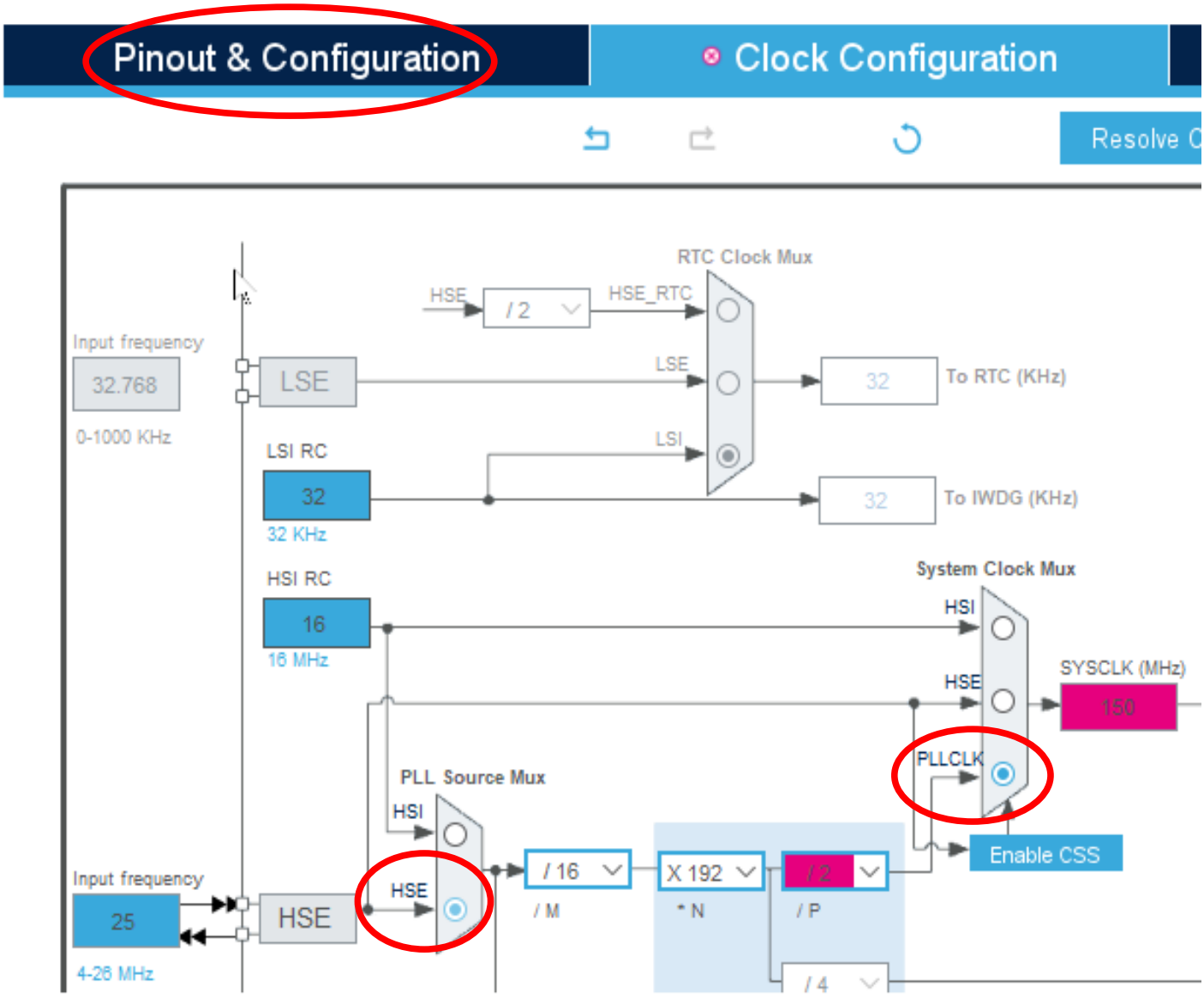
8. Agora vamos criar um pino de saída digital para controlar a cor verde do LED RGB existente na placa BitDogLab. Este LED é ligado no pino PA7 da MCU pelo seu anodo, ou seja, o LED acende quando PA7 está em nível lógico alto. Vamos configurar o pino como saída iniciada em nível alto, para que o LED inicie aceso. Procure o pino PA7 na imagem (no canto inferior direito deste painel há um campo de busca. Basta digitar o nome do pino e ele pisca na imagem). Clique com o botão esquerdo sobre o pino PA7 e na lista que aparecer selecione "GPIO_Output". Este pino também ficará verde, indicando que tem sua função determinada. Clique no mesmo pino com o botão direito e escolha a opção "Enter User Label". No campo que aparecer, digite "LED". Assim, a palavra "LED" pode ser usada no programa para representar o pino PA7.



9. Vamos definir o sistema de “clock” para usar o gerador a cristal em vez do oscilador interno, e ajustar a frequência interna para 48MHz. Inicialmente, dentro do conjunto “SYS”, clique em “RCC” (geradores de clock). Em “High Speed Clock”, selecione “Crystal / Ceramic Resonator” para ativar o oscilador a cristal. Mais dois pinos na imagem se tornam verdes (onde o cristal é ligado).

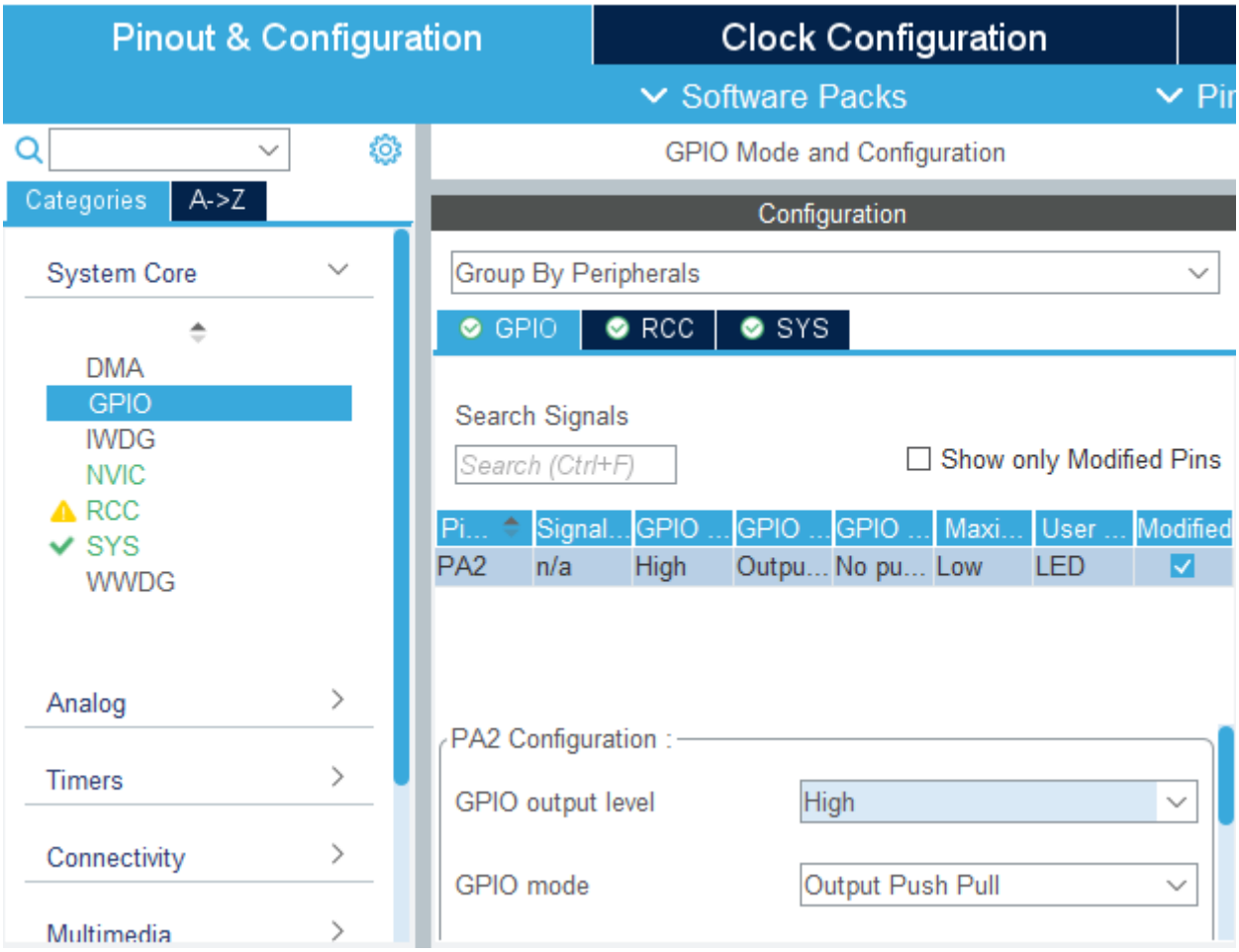


Mais acima no painel, há uma aba chamada “Clock Configuration”. Clicando nela, a imagem muda para a “árvore” de “clock”. Embaixo à esquerda, pode-se ver o bloco do cristal de 8MHz e o HSE (*High Speed External*). É necessário selecionar os MUX para que o sinal de 8MHz passe pelo PLL e tenha a frequência multiplicada, e esta frequência seja usada pelo sistema. Para isso, clique nos pontos marcados na imagem abaixo.



Veja que com essas configurações, o *clock* final seria de 150MHz, acima do máximo permitido. Isto é visível pelas caixas com fundo vermelho. Clique na caixa *HCLK* e digite “48” e “Enter”. O IDE irá calcular os valores de PLL para que o *clock* do sistema seja de 48MHz.

10. Voltando à aba “Pinout & Configuration”, precisamos ainda ajustar o pino do LED para inicializar em nível alto. Dentro de “System Core”, selecione “GPIO”, e no painel do meio aparecerá uma lista dos pinos alocados para esta finalidade. No caso temos apenas PC13, basta clicar na linha da lista e nas opções que aparecerem, selecionar “High” em “GPIO Level Output”.



11. Agora temos a configuração completa para o projeto. Ao clicar no botão de “Salvar”, é perguntado se deseja gerar o código. Basta clicar em “Yes”, e o código será gerado. Depois disso será solicitada nova mudança de perspectiva. Selecione “Yes” (deixando a opção de lembrar a decisão marcada, para que esta solicitação ocorra apenas no primeiro uso).

12. Agora estamos na perspectiva de Programação. À esquerda podemos ver a estrutura de arquivos do projeto, e os demais projetos do *workspace*. O ideal é manter apenas um projeto aberto de cada vez. Para fechar um projeto, basta clicar sobre ele com o botáo direito e escolher a opção “Close Project”. Para abrir um projeto fechado, basta dar um duplo-clique sobre ele.

13. Dentro do arquivo “main.c”, podemos ver a árvore de estrutura à esquerda (*outline*), e linhas de comentários marcando BEGIN e END de várias seções. Qualquer código escrito entre as linhas BEGIN e END da mesma seção será mantido, mesmo que haja modificações na perspectiva de

Inicialização (e consequente re-geração de código). Há áreas para “includes” de usuário, “typedefs”, macros, etc. A área 0 é útil para protótipos de funções e declarações de variáveis globais (apesar de haver áreas específicas para ambos). A área 1 é indicada para a declaração de variáveis dentro da função *main*. A área 2 é útil para chamadas de funções diversas de inicialização antes do *loop* principal. A área 3 é para as instruções dentro do *loop* e a área 4 é para funções adicionais.

14. Neste exemplo, vamos apenas fazer o LED piscar dentro do *loop*. Para isto, vamos usar funções da biblioteca HAL (*Hardware Abstraction Layer*). Uma função permite especificar um *delay* de espera e outra permite inverter o valor lógico de um pino GPIO. Abaixo da linha:

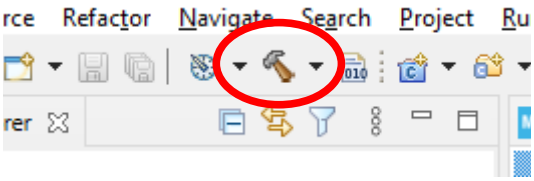
```
/* USER CODE BEGIN 3 */
```

e antes de “}”, digite as seguintes linhas:

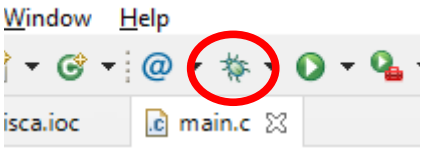
```
HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
HAL_Delay(500);
```

A primeira linha determina que o pino especificado inverte o valor lógico. Note que a função recebe como parâmetros a porta onde o pino está e o número do pino. Neste caso, o gerador de código criou macros para estes parâmetros: *LED_GPIO_Port* e *LED_Pin*.

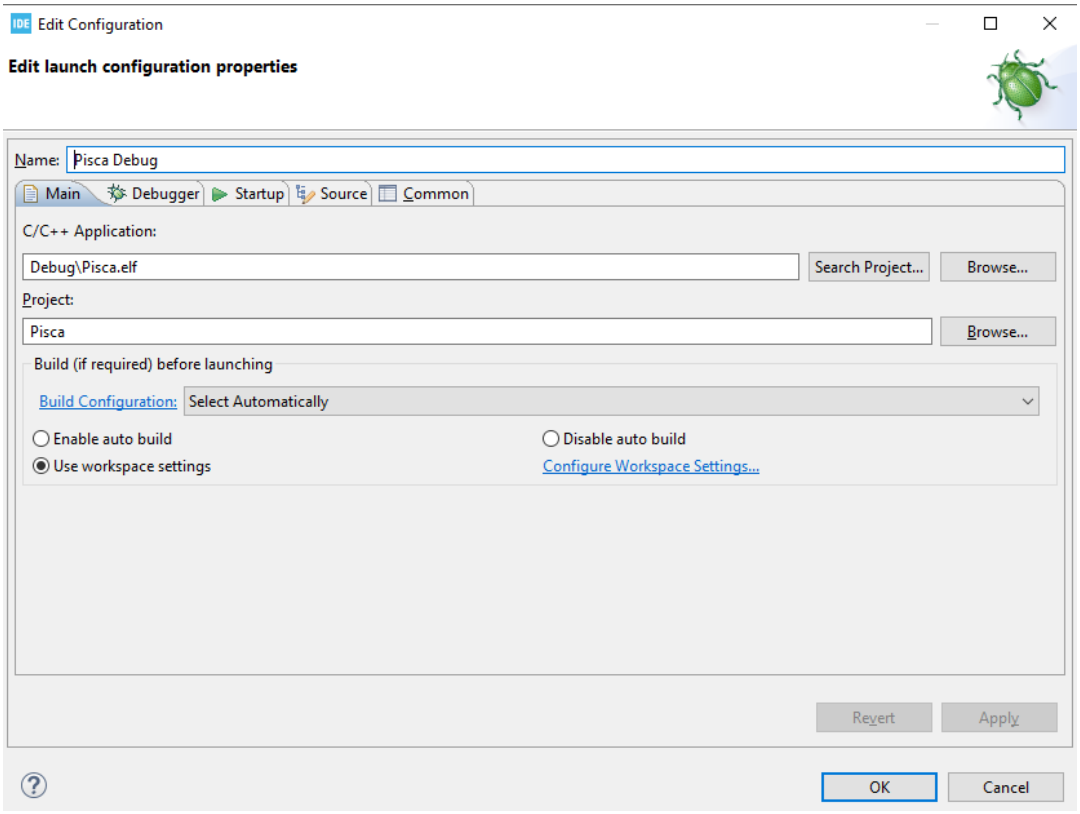
15. Não se esqueça de salvar as modificações do arquivo. Depois use o ícone de martelo para fazer o “Build”:



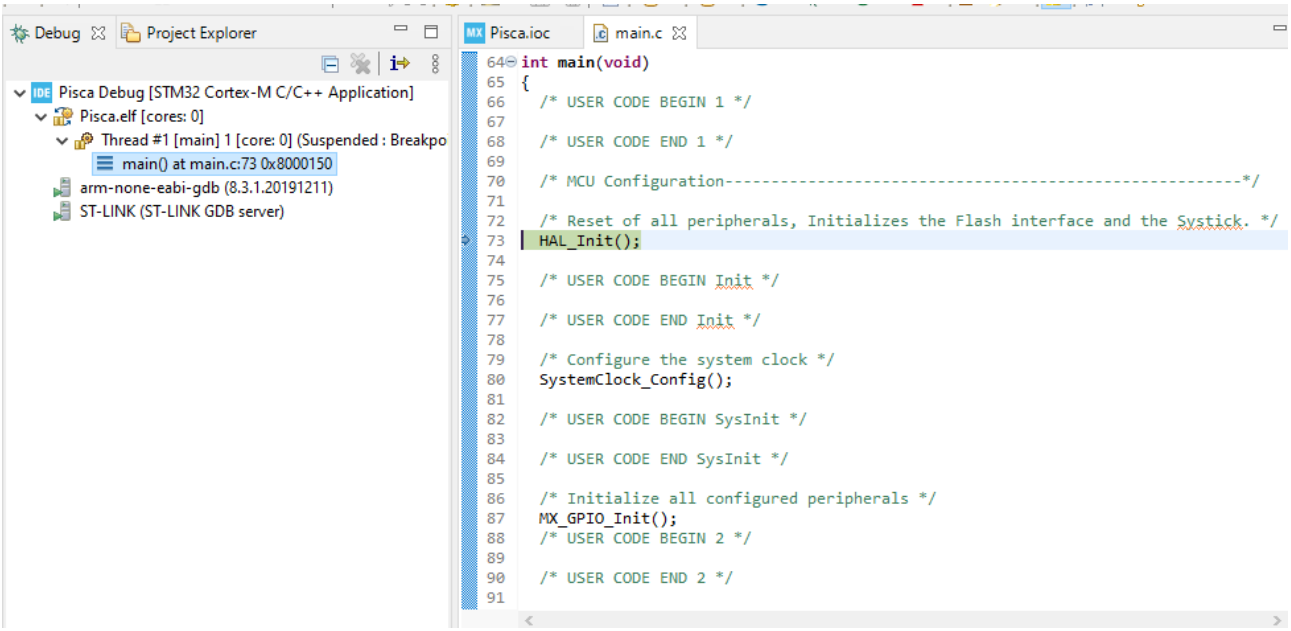
Após o “Build”, estamos prontos para carregar o programa na placa. Vamos entrar no modo e perspectiva de Debug. Clique no botão com o ícone de um besouro para carregar (antes disso, conecte o ST-LINK à placa e a uma porta USB do computador):



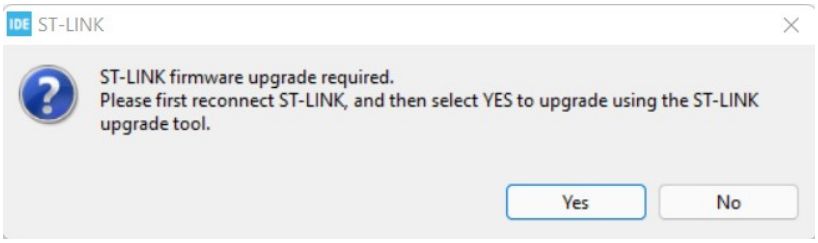
Na primeira vez que o “Debug” é chamado em um projeto novo, abre-se uma janela com uma série de configurações para o debug. Vamos manter todas as opções padrão, então basta clicar no botão “OK”. Isto só é necessário no primeiro “debug” de cada projeto. Depois disso, o IDE solicita a mudança de perspectiva.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



16. Agora vemos na janela esquerda (Debug), que a MCU está parada no início do "main" (selecionado na janela). Ao lado desta janela, aparece o código fonte do "main".

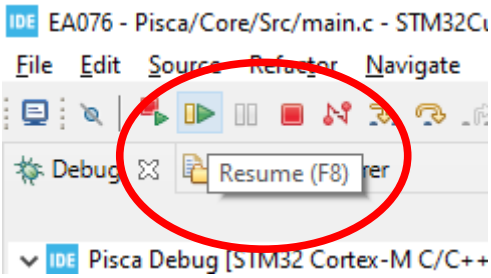


OBS: Quando há uma atualização do *firmware* do ST-LINK, o IDE não fará a carga do programa. Em vez disso, aparecerá a seguinte janela, pedindo para atualizar o *firmware*:

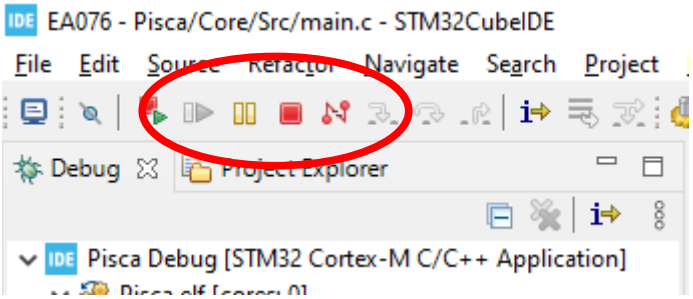


Neste caso, execute os procedimentos da seção “**ATUALIZANDO O *FIRMWARE* DO ST-LINK**”, ao final desta mesma apostila.

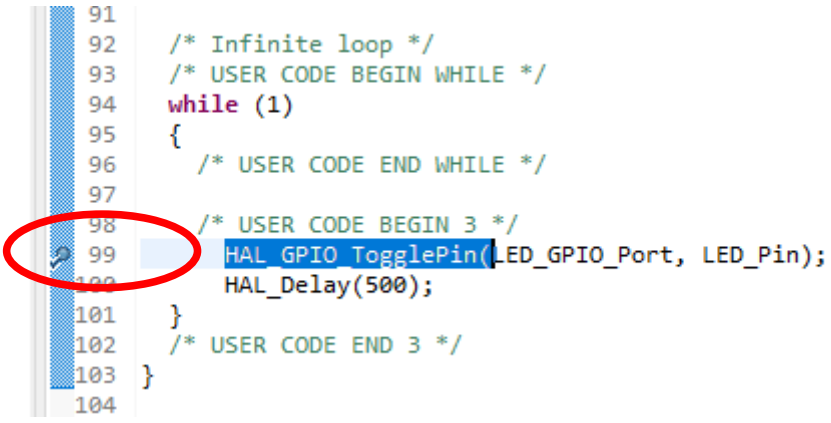
17. Nesta mesma janela, clique no triângulo verde (Resume), ou pressione F8 para iniciar a execução. O LED começará a piscar, ficando meio segundo apagado e meio segundo aceso.



18. Junto ao botão de "Resume", há ainda os botões "Suspend" (símbolo de pausa), "Terminate" (quadrado vermelho) e “Disconnect” (uma linha em forma de “N”). Estes botões permitem respectivamente 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.

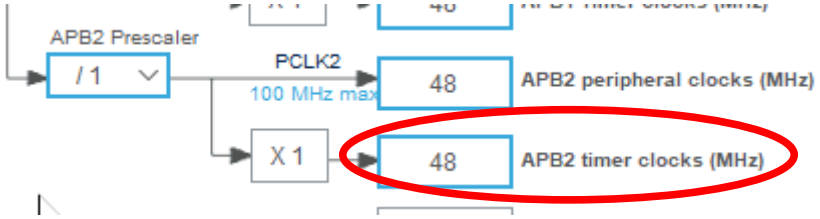


19. Pode-se estabelecer *breakpoints* no código, para que o mesmo seja interrompido toda vez que sua execução chegar a um determinado ponto. À esquerda da linha "HAL_GPIO_TogglePin...", sobre a área azulada, dê um duplo-clique. Aparecerá um pequeno círculo azul. Agora toda vez que o programa chegar neste ponto, será interrompido, podendo ser retomado através do botão "resume".

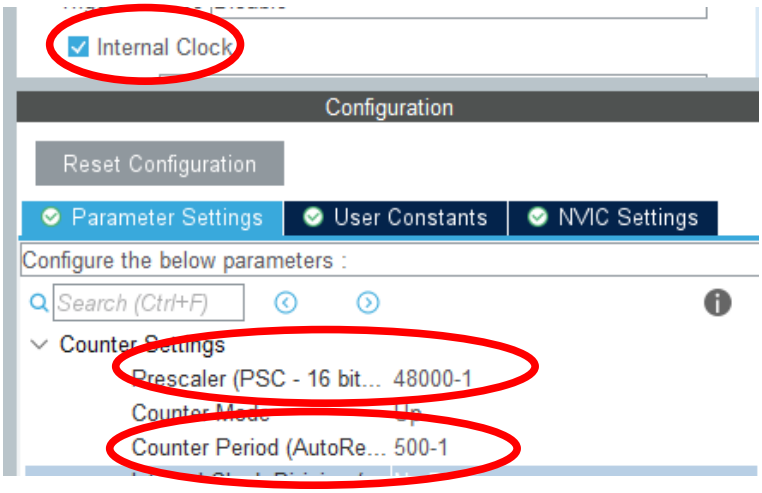


20. Vamos agora modificar o programa para usar interrupções periódicas. Voltando à aba do arquivo “Pisca.ioc”, no painel da esquerda existe um grupo chamado “Timers”. Expanda o grupo e selecione “TIM9”. O *Timer* 9 (TIM9) é de uso geral, e será usado para produzir uma interrupção periódica.

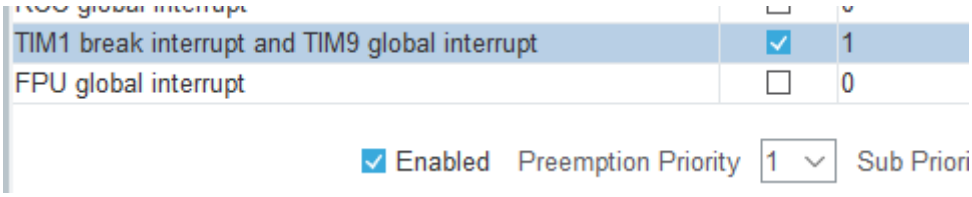
21. Na aba “Clock Configuration”, à direita, pode-se ver as frequências de *clock* para os vários periféricos. O TIM9 está ligado no barramento APB2, portanto usa como referência o *clock* daquele barramento. No caso o *clock* dos *timers* no APB2 é de 48MHz.



Voltgando à aba “Pinout & Configuration”, no painel central vamos configurar TIM9. Marque a caixa “Internal Clock”, para ativar o *clock do timer*, e o próprio *timer*. Nas configurações, precisamos ajustar os parâmetros. Queremos uma interrupção a cada 500ms (equivalente ao *delay* usado anteriormente). Assim, podemos usar o *prescaler* para que o *clock* do TIM9 seja de 1kHz (período de 1ms), e contar 500 ciclos de *clock* para gerar a interrupção. No campo “Prescaler”, digite “48000-1”, para que o *prescaler* divida a frequência do *clock* por 48000 (O fator de divisão é sempre o valor no registrador PSC mais 1, e assim teremos 48MHz / 48000 = 1kHz). No campo “Counter Period”, digite “500-1”, para que a contagem seja feita de 0 até 499, ou seja, 500 ciclos de *clock*.



22. Precisamos agora ativar e configurar a interrupção gerada a cada *reset* do TIM9. No painel à esquerda, expanda novamente o grupo “System Core” e selecione o NVIC (*Nested Vectored Interrupt Controller*). As várias opções de interrupção aparecem. Há uma linha com a interrupção “TIM1 break interrupt and TIM9 global interrupt”. Marque a caixa de seleção nesta linha, e com a linha selecionada, na parte inferior da janela mude a “Preemption Priority” de 0 para 1. Números menores indicam prioridades maiores, e o ideal é manter as prioridades máximas para interrupções padrão, como *Hard Fault*, *Illegal State*, etc. Assim, usamos a prioridade 1 para esta interrupção.



Agora podemos gerar novamente o código de inicialização salvando o arquivo e permitindo a geração do código.

23. Agora vamos modificar o *main.c*. Remova as duas linhas adicionadas anteriormente. O *loop* principal do programa irá ficar vazio, e o *toggle* do LED ocorrerá no tratamento da interrupção, que se repete a cada 500ms.

24. O *Cube* gera o código de configuração do *timer*, mas não inicia sua contagem. O início de contagem deve ser adicionado ao programa. Logo abaixo da linha `/* USER CODE BEGIN 2 */`, adicione:

```
HAL_TIM_Base_Start_IT(&htim9);
```

Essa função irá iniciar a contagem no TIM9, habilitando as interrupções.

25. Precisamos adicionar o tratamento da interrupção. O *Cube* cuida do tratamento de todas as interrupções, inclusive limpando os *flags*, e chama funções de *callback*. O manual de referência das funções HAL apresenta todas as funções de *callback* para cada tipo de periférico. No caso vamos usar a função *PeriodElapsedCallback* (página 1097). Veja que a função é declarada como *void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)*. Ela existe em um dos arquivos do projeto, com o qualificador *weak*. Assim, se o programador não a declarar, será usada a versão *weak*, que é vazia. Caso o programador a declare, a versão do programador se sobrepõe à gerada pelo *Cube*.

Abaixo da linha `/* USER CODE BEGIN 4 */`, escreva a função:

```
void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim){
	if(htim->Instance == TIM9) {
		HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
	}
}
```

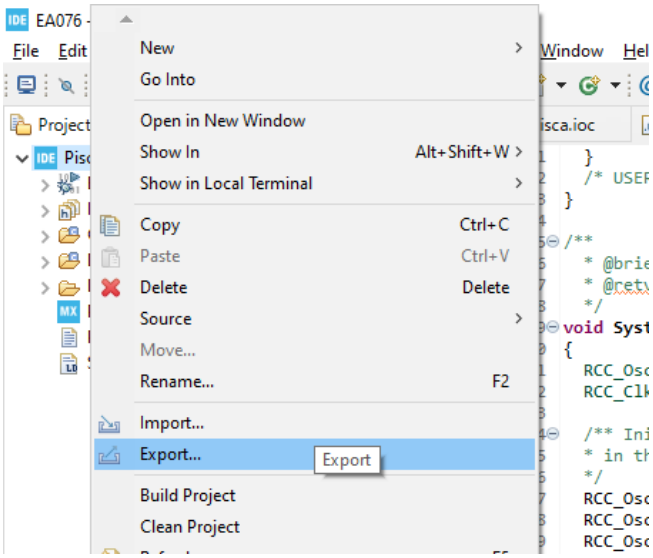
Na função, é sempre recomendável testar a instância do periférico. Se mais de um *timer* for usado, a mesma função de *callback* será usada para todos, e a identificação do *timer* que gerou o *callback* estará disponível no elemento *Instance* da *struct htim*. Neste exemplo, verificamos se a interrupção foi gerada por TIM9, e, caso positivo, ocorrerá o *toggle* do LED.

26. Compile novamente o projeto e realize o *debug*. O LED verde piscará como no exemplo anterior, mas agora usando a interrupção periódica. Pode-se colocar um *breakpoint* dentro do *callback* para testes.

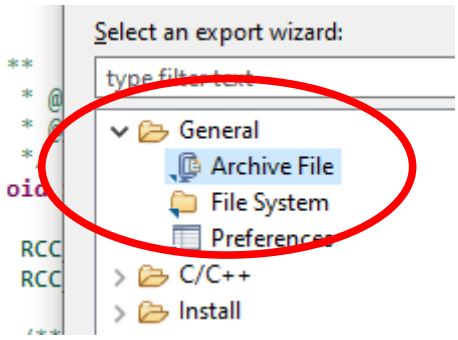
EXPORTANDO / IMPORTANDO UM PROJETO

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".

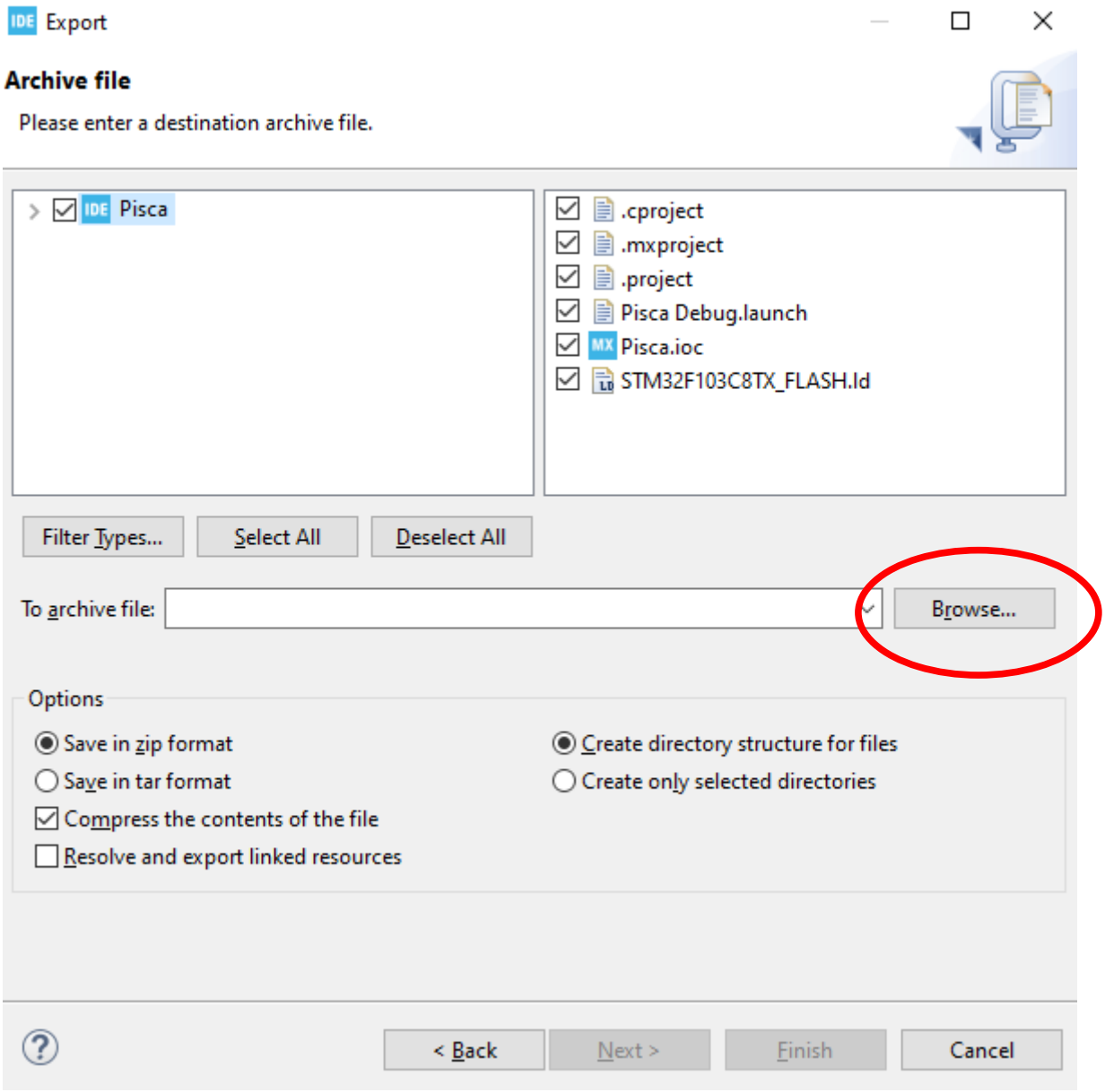
2. Clique novamente com o botão direito na pasta do projeto e selecione "Export...". Alternativamente, pode-se usar o menu "File", opção "Export…"



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

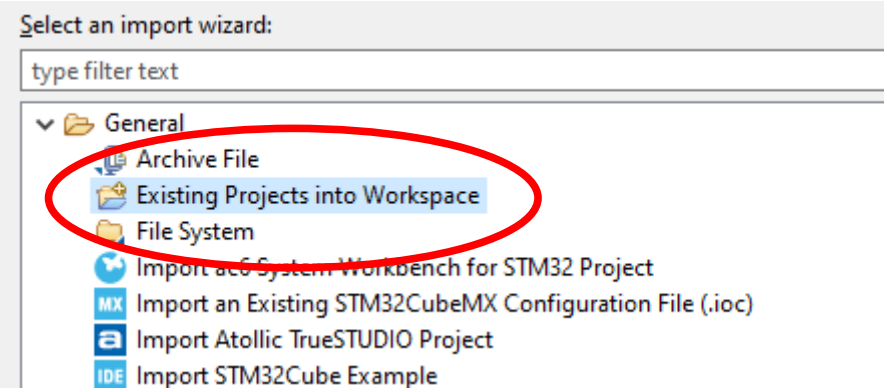


4. Nesta janela, mantenha as opções padrão, e clique no botão "Browse...". Selecione uma pasta para guardar o arquivo de exportação, e o nome do arquivo. Clique em "Finish".



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” abaixo). Se um projeto com o mesmo nome de outro projeto no mesmo *workspace* tentar ser importado, aparecerá uma mensagem de erro.

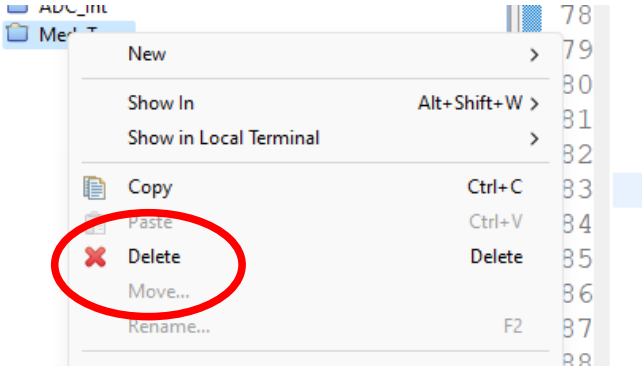
6. Para importar, clique com o botão direito na área dos projetos e selecione "Import...", ou selecione “Import Project” na janela “Commander”, ou ainda use o menu "File – Import…". Escolha a opção “Existing Projects into Workspace” dentro de “General”.



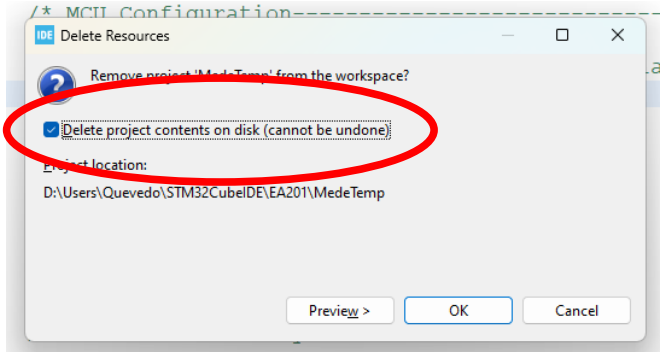
7. Marque a opção "Select archive file" e clique no botão "Browse". Localize e selecione o arquivo ZIP que contém o projeto exportado anteriormente. Confirme que o projeto está selecionado e clique em "Finish".

APAGANDO UM PROJETO

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”.

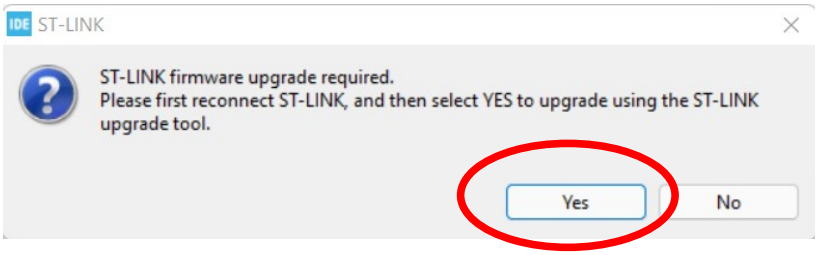


2. Na caixa de diálogo que aparece, selecione a opção “Delete project contents on disk” e clique no botão OK. Sem marcar a opção, os arquivos do projeto permanecerão no *workspace*, mas não aparecerão na lista.

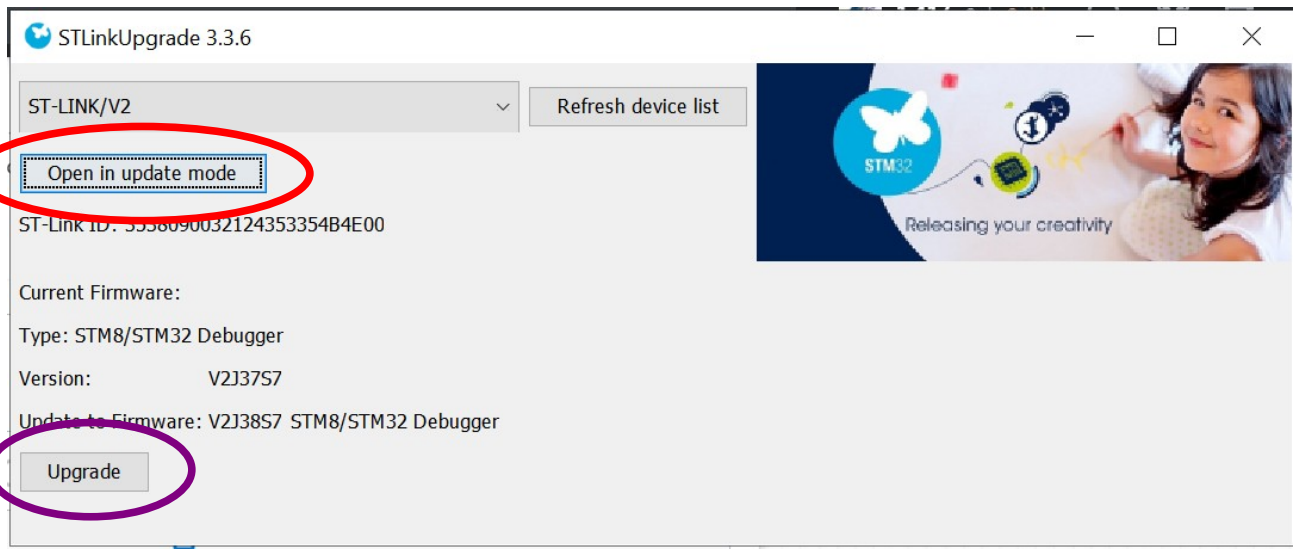


ATUALIZANDO O *FIRMWARE* DO ST-LINK

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.

6. Depois de reconectar o ST-LINK, pode-se repetir a chamada de *debug*.