Índice

Capítulo 2

Ambiente de Desenvolvimento – *Software*

Autores: Wu Shin-Ting e Antônio Augusto Fasolo Quevedo

Estendida e atualizada por Wu Shin-Ting (08/2022)

O *Integrated Development Environment* (IDE) CodeWarrior 10 é o ambiente de desenvolvimento de *software* desenvolvido pela *Freescale* para os microcontroladores e microprocessadores por ela desenvolvidos [7]. Ao contrário das versões anteriores, desenvolvidas de maneira totalmente proprietária, as versões >= 10 foram desenvolvidas com o objetivo que ele seja um *plug-in* do **Eclipse**.

Eclipse é um ambiente de desenvolvimento integrado (*Integrated Development Environment,* IDE) com um sistema de *plug-ins* que o permite suportar várias linguagens de programação. Foi iniciado na IBM e doado à comunidade como *software* livre nos termos da *Eclipse Public License*. Assim o ambiente Eclipse é gratuito e de código aberto. Vem se tornando um aplicativo cada vez mais utilizado em empresas que desenvolvem projetos de *software*.

Os *plug-ins* garantem funcionalidade expansível ao sistema. Assim, além de adicionar novas linguagens de programação, permite também integrar ferramentas de teste e depuração de código ao ambiente. Pode-se ainda incluir funcionalidades relativas a famílias específicas de processadores/controladores. Por exemplo, quando a *Freescale* lança uma nova família de processadores, também lança um *plug-in* para o CW10, 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 meta-dados 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 grupo de alunos, os quais podem colocar todos seus projetos dentro do mesmo. Assim, numa 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. O IDE suporta 6 perspectivas: *C/C++* (Programação), *CVS Repository Exploring*, *Debug* (Depuração), *Hardware*, *Resource* e *Team Synchronizing*. Nesta disciplina, 2 perspectivas serão as mais utilizadas: **Programação** e **Depuração**. Na perspectiva de programação, temos, dentre outras, janelas para edição dos códigos-fonte, árvore de arquivos do projeto, localização de módulos de código, e geração de códigos executáveis. Na perspectiva de depuração, temos janelas para visualização de ponto de execução, controles para execução passo-a-passo do código, janelas para apresentação dos valores nos registradores e na memória em tempo real, código *assembly* gerado a partir do código-fonte etc. Para cada janela, existem botões para maximizar, minimizar, restaurar tamanho original e fechar. As divisões entre janelas também podem ser deslocadas, modificando-se assim os tamanhos das diversas janelas.

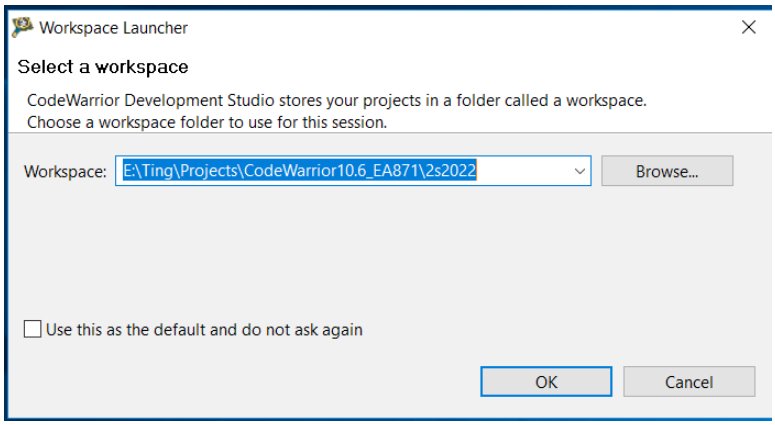
Vale uma ressalva: as janelas apresentadas neste documento são de telas do CodeWarrior 10.6/11.1, executando em Windows 10. Pequenas diferenças podem aparecer em futuras versões do programa, e em outros sistemas operacionais, mas as diferenças não são suficientes para prejudicar este tutorial.

2.1 Criação de um Projeto

O capítulo 1 em [1] apresenta, sob o ponto de vista de programação, um procedimento típico para inicializar o microcontrolador MKL25Z. Como este procedimento é comum para uma grande maioria dos projetos, o IDE CodeWarrior dispõe de um mecanismo que gera automaticamente uma estrutura básica de programas onde os códigos específicos de cada aplicativo são integrados como rotinas. Com isso, o trabalho de um projetista pode se centrar no desenvolvimento dos códigos relacionados diretamente com a sua aplicação de interesse. Nesta seção é apresentada a sequência de passos necessária para gerar, através dos parâmetros de configuração, a estrutura básica de um programa de projeto no IDE.

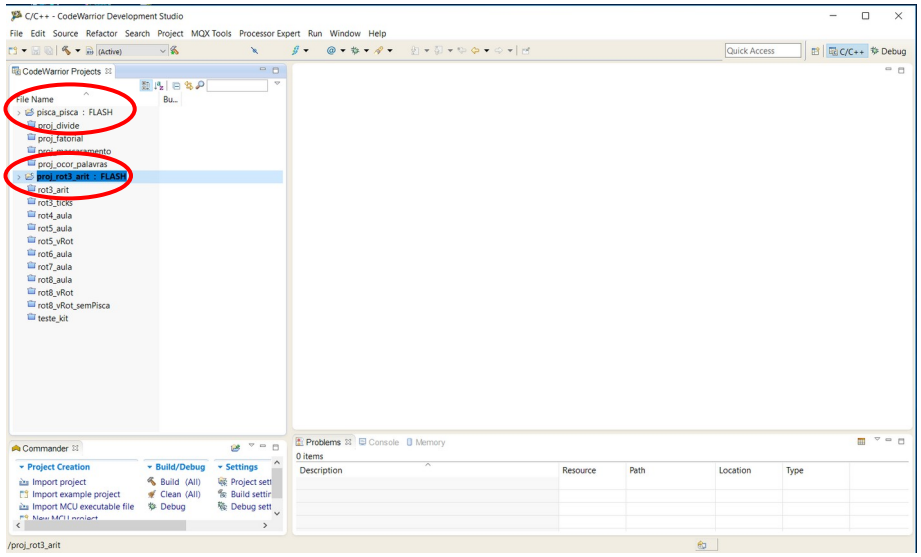
1. No Windows, vá ao menu "*Iniciar > Programas > Freescale CodeWarrior > CW for MCU v10.6 > CodeWarrior*", ou chame o programa através do ícone do CodeWarrior 10.6 na área de trabalho.

2. Aparece o *Workspace Launcher*. Aqui você determina qual *workspace* você deseja utilizar. Clique em *Browse...* para escolher uma pasta para ser seu *workspace* ou selecione uma já existente na lista *drop-down*. Clique em OK. Sugerimos que cada grupo crie seu próprio *workspace*.

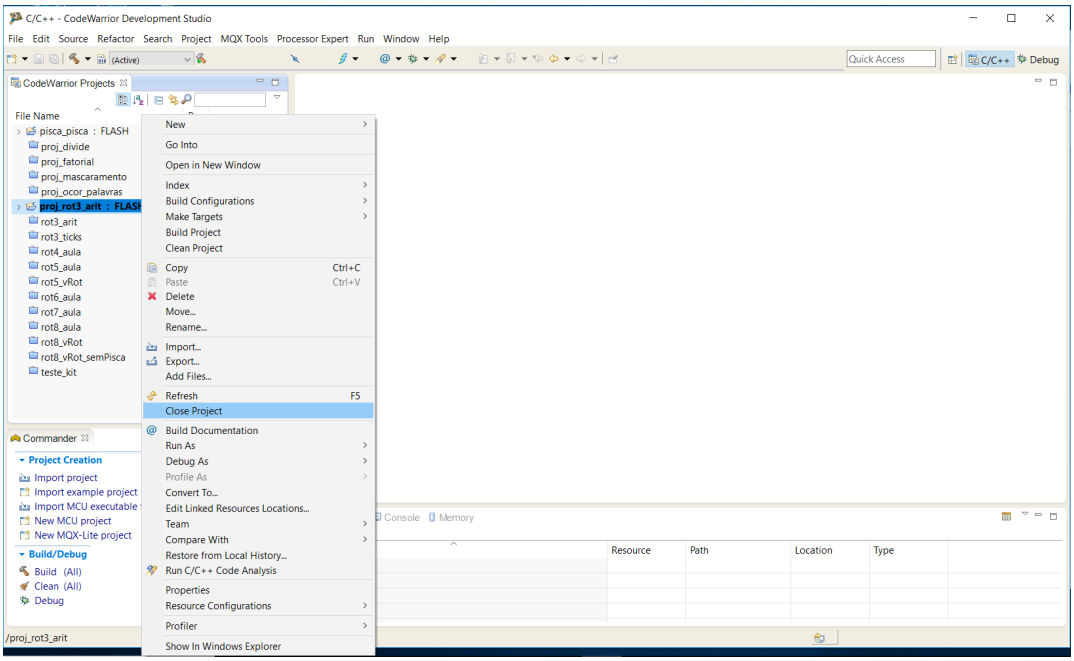


3. O CodeWarrior vai abrir na perspectiva de programação. Pode-se ver à esquerda um painel com todos os projetos do *workspace*. Uma pasta aberta significa que aquele projeto está aberto.

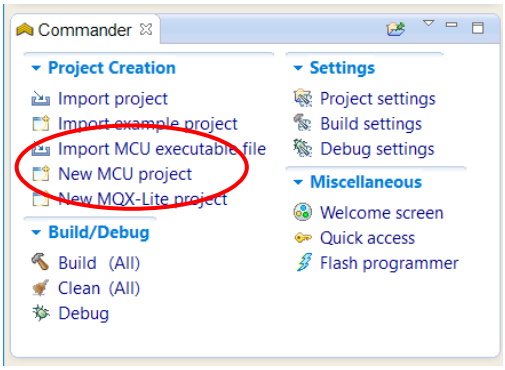
Na primeira utilização do *workspace*, ao invés da perspectiva de programação, pode aparecer um menu geral. Neste caso, basta escolher a opção "*Go to Workspace*".



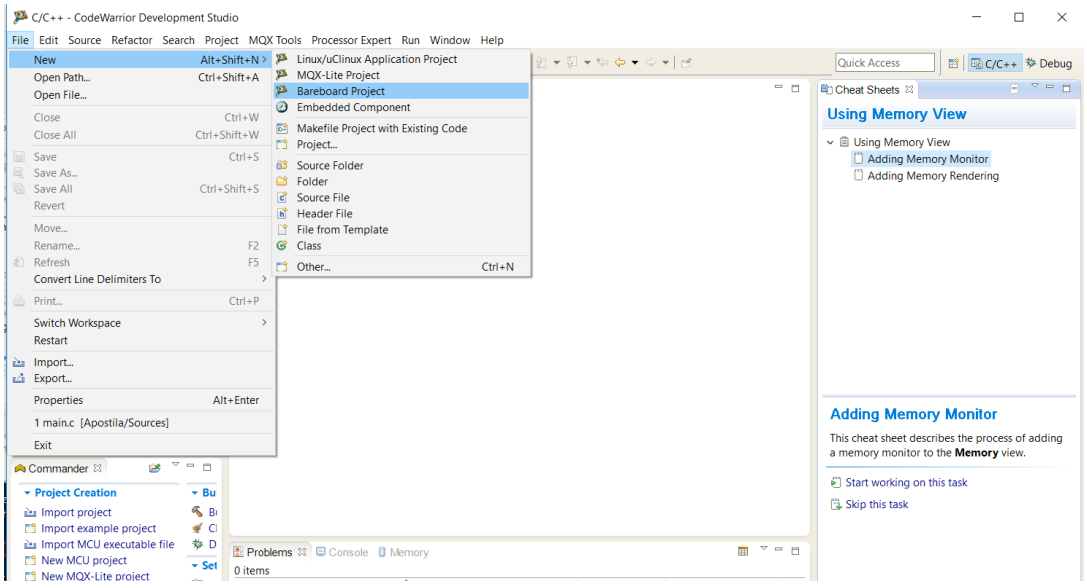
4. Evite ter mais de um projeto aberto de cada vez. Se houver diversos projetos abertos, feche os desnecessários clicando com o botão direito sobre a pasta do projeto aberto e escolhendo a opção "*Close Project*" no menu *popup*.



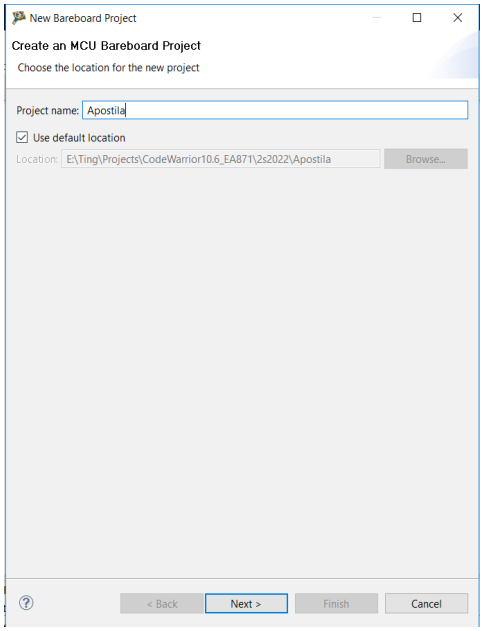
5. A janela *Commander* no canto inferior esquerdo da perspectiva agrupa os comandos mais comuns. Para criar um novo projeto, basta clicar em *New MCU project* nesta janela.



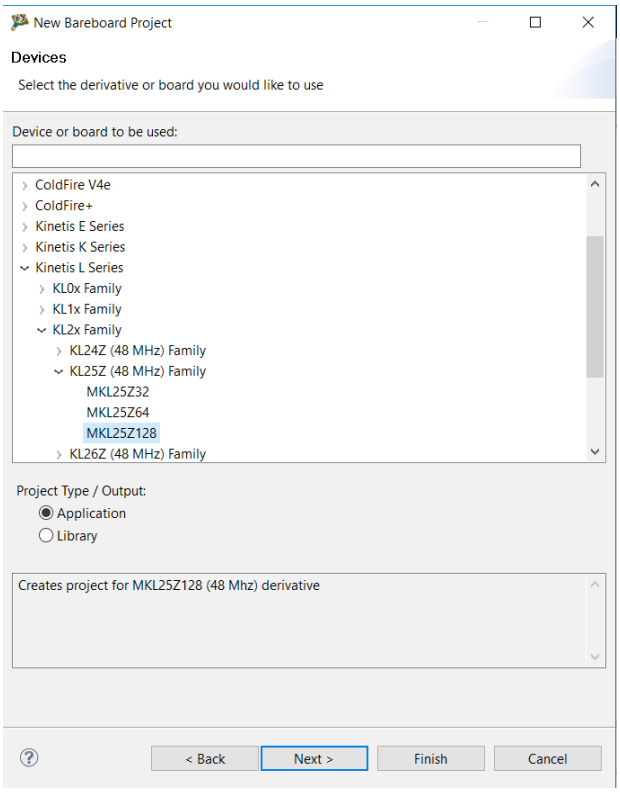
Alternativamente, no menu superior selecione File > New > Bareboard Project.



6. Na janela que se abre, escolha um nome para o projeto. No exemplo, foi escolhido o nome "Apostila". Se a caixa de seleção "Use default location" estiver selecionada, os arquivos serão salvos na pasta do workspace, que pode ser vista logo abaixo da caixa. Desmarcando a caixa, pode-se escolher outra pasta para colocar os arquivos. Depois, clique no botão "*Next*".

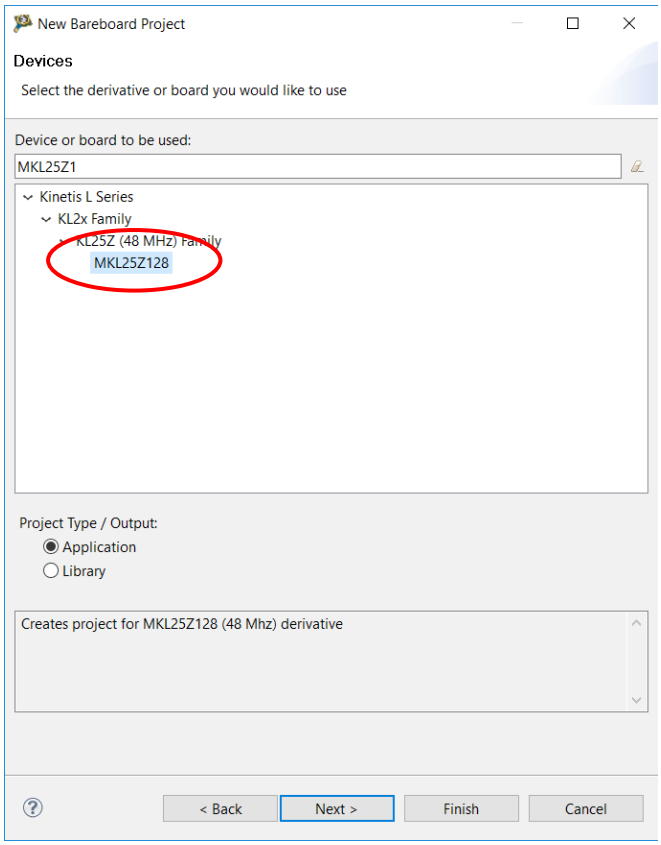


7. Agora deve-se escolher o microcontrolador ou a placa de desenvolvimento a ser utilizada. Neste curso usamos o controlador MKL25Z128, na placa FRDM-KL25. Na árvore de opções, siga o caminho "Kinetis L

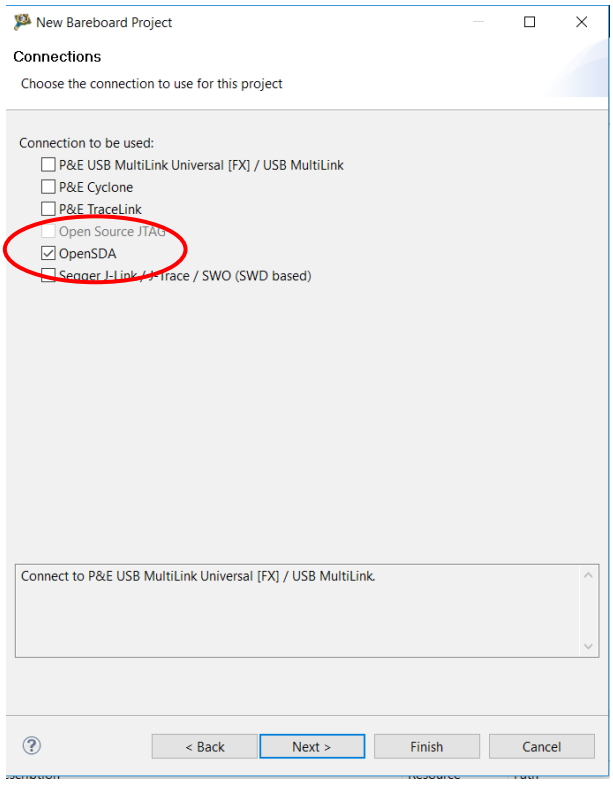


Series > KL2x Family > KL25Z (48MHz) Family > MKL25Z128”". Clique "Next" em seguida. A

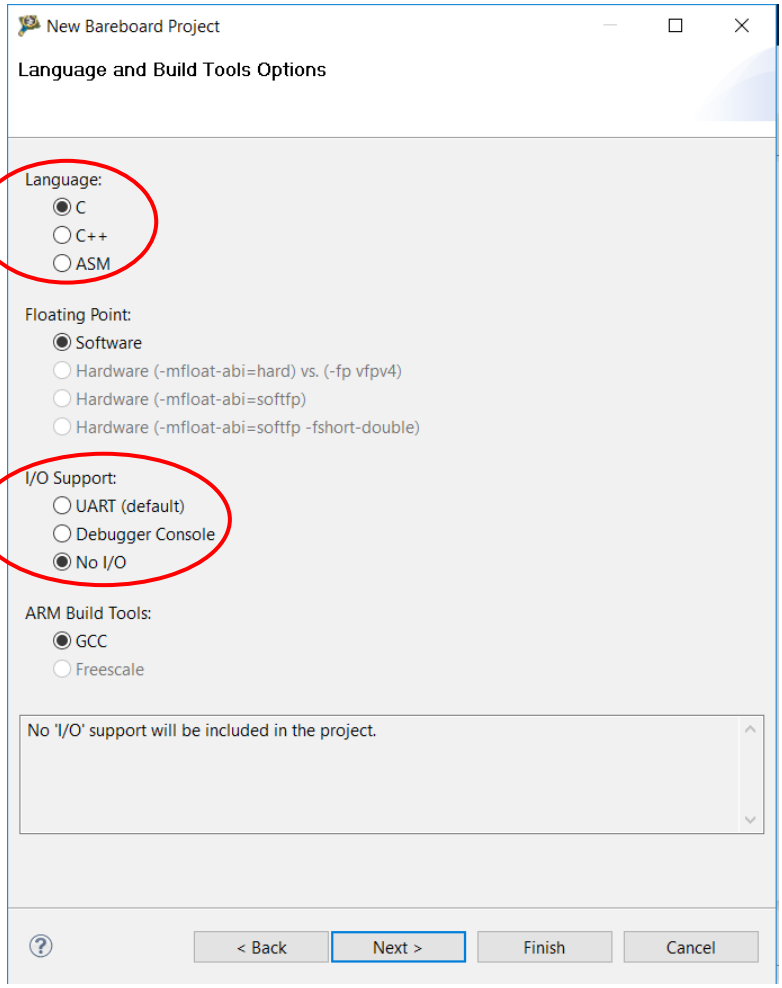
Alternativamente, pode-se digitar “MKL25Z1”, ou uma outra substring, na caixa de pesquisa (intitulada *“Device or board to be used”*). Neste caso, o programa filtra as opções, apresentando apenas a sub-árvore dos processadores MKL25Z1, e assim pode-se selecionar mais facilmente o processador-alvo.



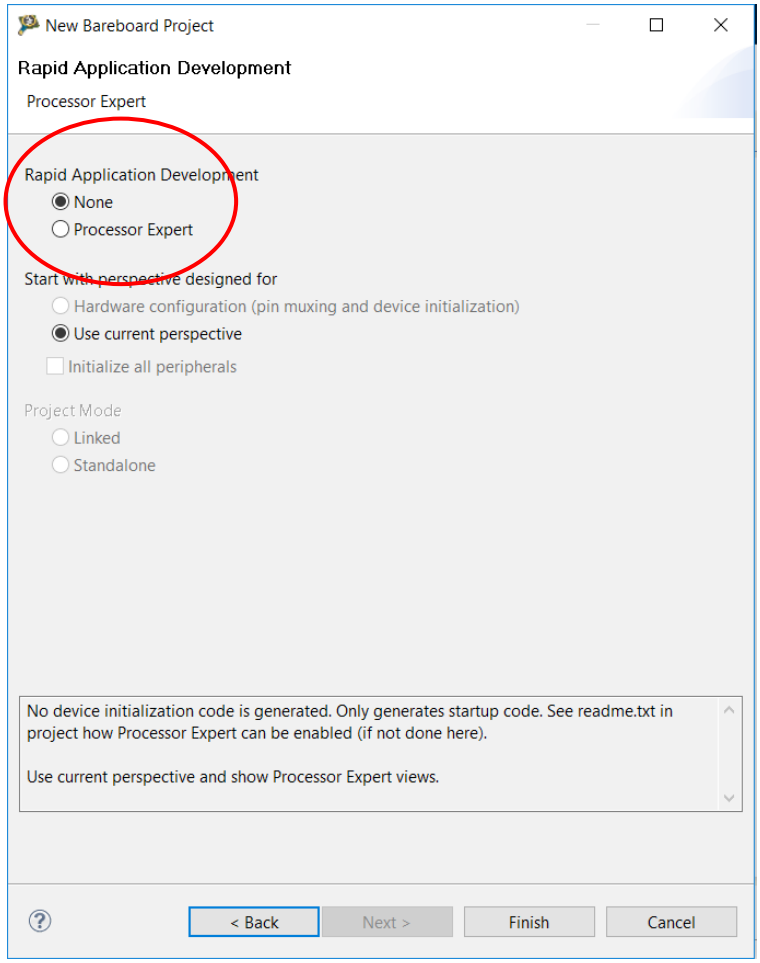
8. Agora, aparece uma lista de possíveis conexões. A conexão estabelecida entre o computador e a placa permite transferir os programas executáveis para a placa, bem como executar a depuração dos programas carregados em MCUs em tempo real. Neste caso, selecione a opção *"OpenSDA*”, que é o *hardware* de conexão existente na placa de desenvolvimento FRDM-KL25. Desmarque quaisquer outras opções selecionadas, e depois clique em "*Next*".



9. Na próxima janela, mantenha as opções no padrão e clique em "*Next*". Estas opções se referem à linguagem e o compilador a serem utilizados no projeto, o uso de ponto flutuante ou não nos códigos, e suporte ao uso de porta serial ou console de *debug*. Pela opção "*Language"* inferimos que o IDE suporta 3 linguagens, C, C++ e *assembly* (ASM). Na janela foi selecionada a linguagem C. Se o seu código-fonte for em linguagem *assembly*, marque a opção ASM. Na opção "*I/O Support"*, as alternativas *default* (UART) e *Debugger Console* acrescentam código para suporte a funções *printf* e *scanf* através da porta serial ou de uma janela de console. Quando estas funções não são necessárias, pode-se marcar a opção "*No I/O"*. Isso reduz o tamanho do código final e evita inicializações desnecessárias. Sugerimos usar a terceira opção, a menos que o experimento use as funções acima citadas.

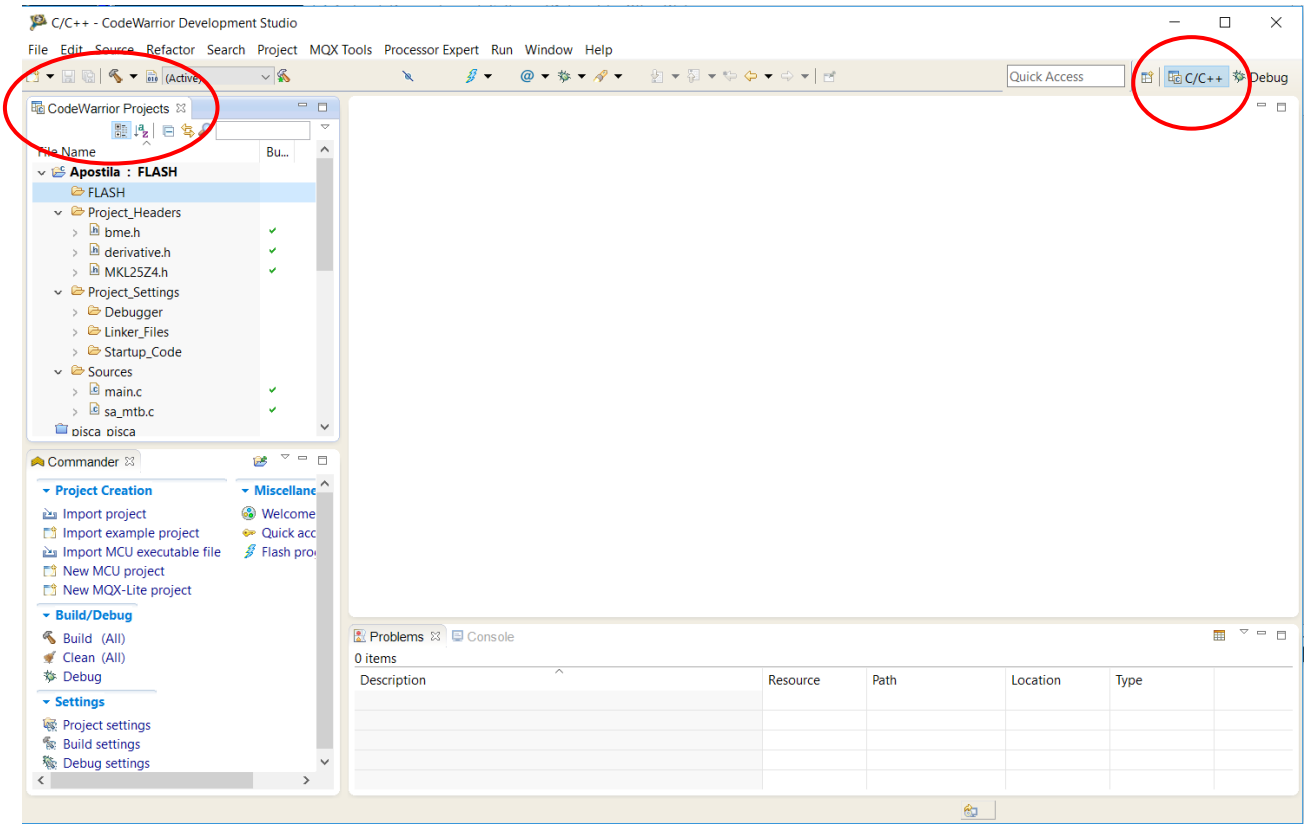


10. Nesta próxima janela, definimos se o *CodeWarrior* usará ou não uma ferramenta que facilita a geração de códigos para a inicialização dos módulos do MCU e funções em alto nível para controlar estes módulos *(Processor Expert*). Neste curso, esta ferramenta não será utilizada, e a mesma só está disponível se a linguagem utilizada for o C (quando se usa *assembly*, esta opção está indisponível). Mantenha a opção "*None"* selecionada e clique em "*Finish"*.



2.2 Estrutura de um Projeto

Como vimos na Seção 2.1.1 é muito simples criar um novo projeto no ambiente *IDE CodeWarrior*. Veja agora que, depois do procedimento, temos na aba "*CodeWarrior Projects*" da **perspectiva de programação** *"C/C++"* o projeto `Apostila` criado e aberto. A aba "*CodeWarrior Projects*" provê uma série de ferramentas de gerenciamento de arquivos e pastas relacionados aos projetos disponíveis num *workspace* do IDE. Foram gerados automaticamente pelo IDE quatro pastas: FLASH, *Project Headers*, *Project Settings* e *Sources*. Clicando no pequeno triângulo à esquerda de cada pasta, podemos expandí-la e ver seus componentes pré-criados pelo *CodeWarrior*. Com exceção da pasta FLASH, todas as outras pastas não são vazias.



Na pasta *Project Headers* temos os arquivos de cabeçalho onde estão incluídas as declarações de tipos de dados e variáveis, os protótipos de funções e as macros. Dois arquivos contendo as declarações relacionadas especificamente com o microcontrolador MKL25Z4 foram adicionadas automaticamente: `derivative.h` e `MKL25Z4.h`. A pasta *Sources* é reservada para todos os códigos-fonte do projeto. Existem pré-definidos dois arquivos, `main.c` e `sa_mtb.c`, nesta pasta. O conteúdo do programa `main.c` é a definição da rotina `main` como mostra na aba `main.c` da janela à direita. Os códigos de inicialização do microcontrolador estão na pasta *Project Settings*. Nela há 3 pastas: *Debugger*, *Linker_Files* e *Startup_Code*.

Na pasta *Debugger* encontram-se *scripts* e arquivo de configuração úteis para depuração dos códigos. Na pasta *Linker_Files* temos o arquivo de linkagem (*loader file*) *MKL25Z128_flash.ld* que especifica os espaços de endereços em que os códigos e os dados devem ser relocados na memória do microcontrolador. Este arquivo contém a definição não só da partição do espaço de memória em diferentes segmentos, como também a especificação da distribuição dos códigos e dos dados do programa nestes segmentos e o topo da pilha. São especificados os sub-espaços de memória [0x00000000, 0x000000C0], [0x00000800, 0x0001FFFF] e [0x1FFFF000, 0x20002FFF] para a tabela de vetores de interrupção (`m_interrupt`), para as instruções (`m_text`) e para os dados (`m_data`), respectivamente. Por padrão, o IDE atribui o endereço `0x20003000` como o topo da pilha de usuário (`_estack`).

MKL25Z128_flash.ld

```
/* Entry Point */
ENTRY(__thumb_startup)

/* Highest address of the user mode stack */
_estack = 0x20003000;    /* end of SRAM */
__SP_INIT = _estack;

/* Generate a link error if heap and stack don't fit into RAM */
__heap_size = 0x400;                /* required amount of heap  */
__stack_size = 0x400;               /* required amount of stack */

/* Specify the memory areas */
MEMORY
{
  m_interrupts  (rx) : ORIGIN = 0x00000000, LENGTH = 0xC0
  m_cfmprotrom  (rx) : ORIGIN = 0x00000400, LENGTH = 0x10
  m_text        (rx) : ORIGIN = 0x00000800, LENGTH = 128K - 0x800
  m_data       (rwx) : ORIGIN = 0x1FFFF000, LENGTH = 16K        /* SRAM */
}


/* Define output sections */
SECTIONS
{
  /* The startup code goes first into Flash */
  .interrupts :
  {
    __vector_table = .;
    . = ALIGN(4);
    KEEP(*(.vectortable)) /* Startup code */
    . = ALIGN(4);
  } > m_interrupts

  .cfmprotect :
  {
    . = ALIGN(4);
    KEEP(*(.cfmconfig)) /* Flash Configuration Field (FCF) */
```

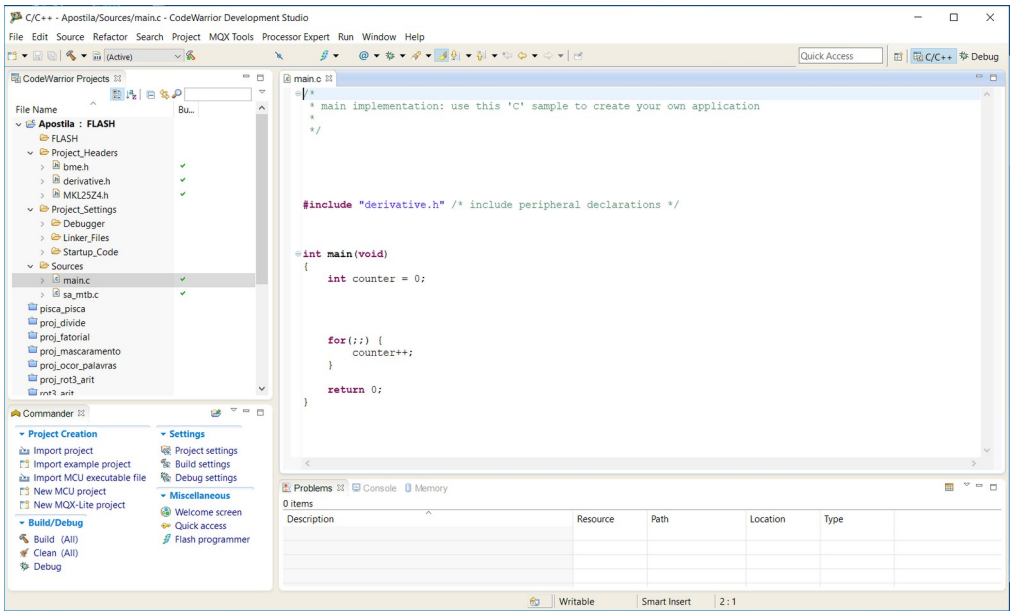
Os códigos de inicialização propriamente ditos ficam na pasta *Startup_Code*. O arquivo `kinetis_sysinit.c` contém as instruções que preenchem cada entrada da tabela de vetores de interrupção com os respectivos endereços das rotinas de serviço. É também desabilitado neste arquivo o *watchdog* do microcontrolador. E no arquivo `arm_start.c` podemos ver que a rotina `__thumb_startup` contém a sequência de instruções de inicialização apresentada na Seção 1.1.4 da referência [1], tendo como a sua última instrução a chamada à rotina `main` supracitada.

No arquivo `kinetis_sysinit.c`, o endereço da rotina *__thumb_startup(void)* é carregado no vetor 1 da tabela de vetores de exceção e, pela Tabela 3-7 da referência [2], o vetor 1 contém o endereço inicial do contador de programa (PC). Portanto, podemos concluir que todos os projetos criados com o procedimento dado na Seção 2.1.1 inicializam a sua execução pela rotina `__thumb_startup`, cujo endereço é carregado no endereço 0x00000004 do espaço de memória (Seção 1.1.3 da referência [1]). Só depois da inicialização é chamado, nesta mesma rotina, o procedimento `main` que está no arquivo `main.c` da pasta *Sources*. Esta forma de organização de códigos dispensa o desenvolvedor do trabalho de codificação dos (mesmos) códigos de inicialização padrão para projetos distintos.

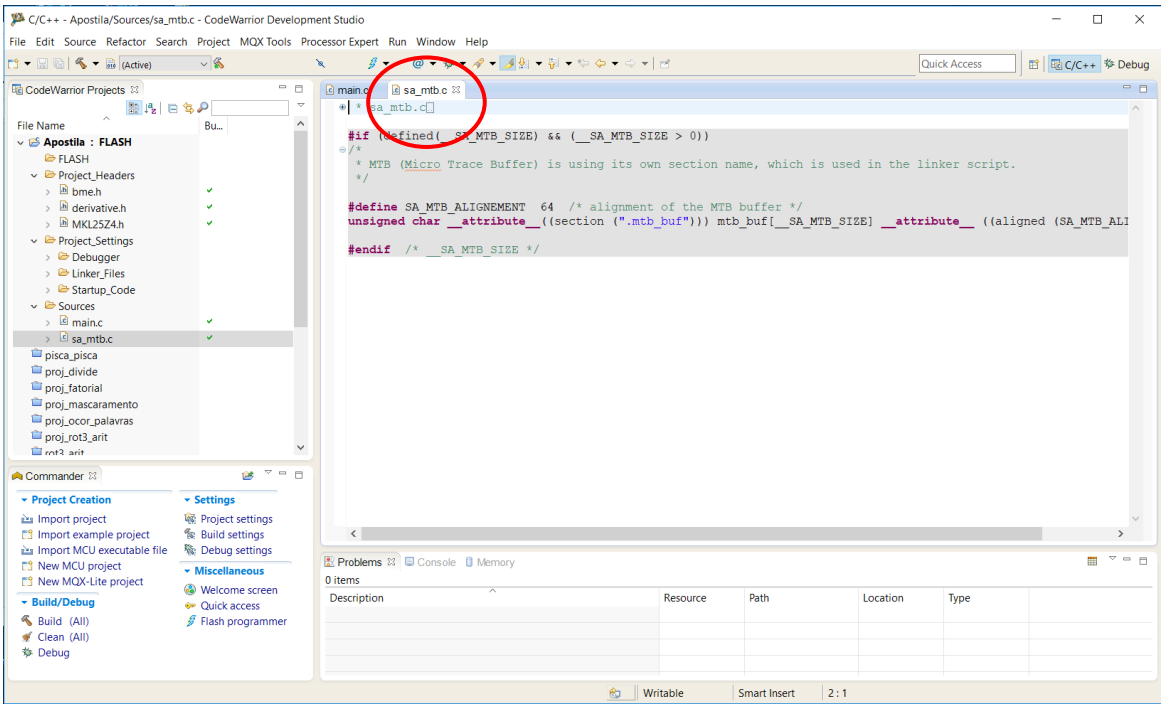
2.2.1 Edição de Códigos-Fonte

Quando se dá um duplo-clique em qualquer arquivo, ele é aberto no editor (neste caso, aba `main.c`) que ocupa a janela central. O programa mostrado no editor apresenta a estrutura básica de um código C que se integraria facilmente com o restante dos códigos gerados pelo IDE. Neste caso a rotina `main` simplesmente inicializa uma variável inteira com o valor 0 e a incrementa a cada

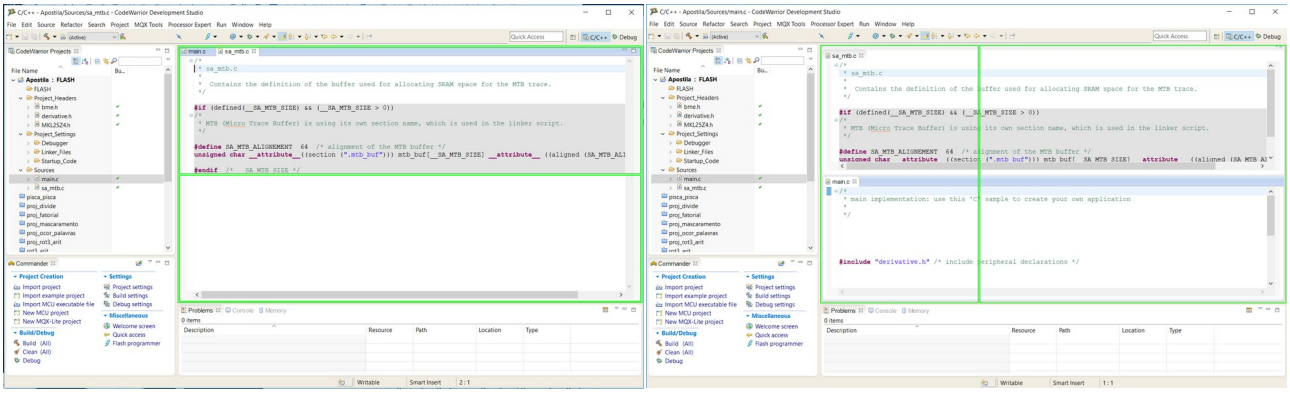
iteração de um laço infinito. Se quisermos implementar outras funcionalidades, basta substituirmos estes códigos por outros de nosso interesse. O programa inclui o arquivo de cabeçalho `derivative.h` e implementa a rotina `main` chamada na rotina `__thumb_startup`. Algumas facilidades nesta área de edição podem ser consultada em [3]. E um sumário das teclas de atalho para edição dos códigos é encontrado em [15].



Podemos ter vários arquivos abertos simultaneamente. Por padrão, eles são orgnizados em painéis sobrepostos e trazidos para o primeiro plano quando se clica na sua aba. A figura abaixo mostra que ambos os arquivos, `main.c` e `sa_mtb.c`, estão abertos e o segundo está no primeiro plano. Para trazer `main.c` ao primeiro plano, é só clicar na sua aba.



É possível dividir a área de edição em vários sub-painéis expondo o conteúdo de diferentes arquivos simultaneamente como mostram as figuras abaixo. Na figura à esquerda a área foi dividida horizontalmente, e na figura à direita, verticalmente. Para isso, basta arrastar a aba do arquivo, cujo conteúdo se quer visualizar, mantendo o botão esquerdo do *mouse* pressionado.

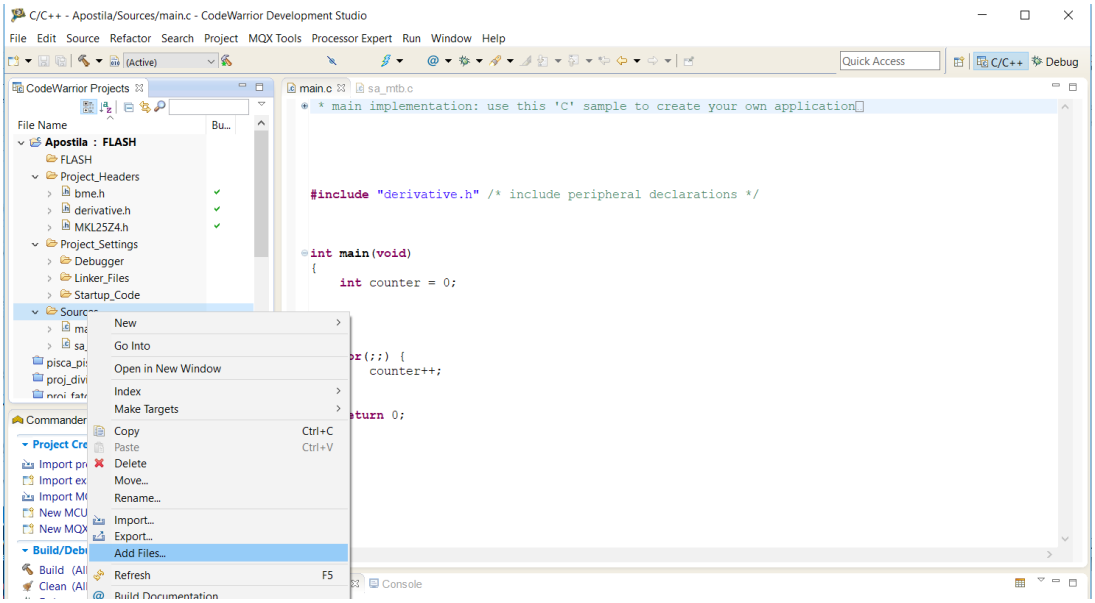


2.2.2 Adição de Novos Arquivos

É muito comum separar as funções de um programa em vários arquivos pequenos. Para adicionar um novo arquivo numa pasta, por exemplo *Project_Headers* ou *Sources*, basta selecionarmos a pasta em que queremos adicionar um novo arquivo e entrarmos o nome do arquivo seguindo o caminho de comandos "*File* > *New* > *Header Files* > *nome*" ou "*File* > *New* > *Source Files* > *nome*".

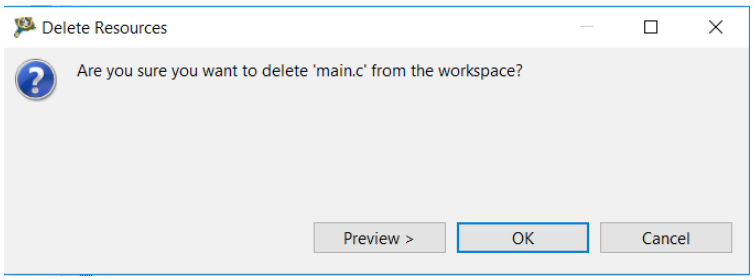
2.2.3 Inclusão ou Sobreposição de um Arquivo

Quando se quer sobreescrever um arquivo dentro de uma pasta ou incluir um arquivo na pasta, por exemplo *Sources* como mostra a figura abaixo, é só selecioná-la e clicar sobre ela o botão direito do *mouse* que aparecerá um menu *popup* contendo a opção "*Add Files ...*". Ao selecionar esta opção, mostrará na tela um explorador de arquivos pelo qual você escolhe o arquivo desejado. Outra forma é simplesmente arrastar o arquivo que se queira incluir para o ponto desejado na vista/janela *CodeWarrior Projects*.

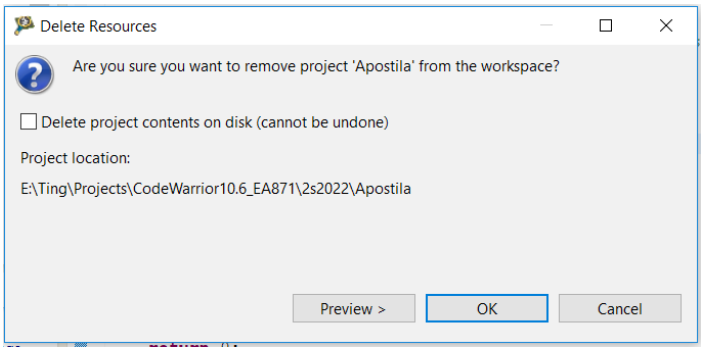


2.2.4 Exclusão de Arquivos, Pastas ou Projetos

Para remover um item da aba *"CodeWarrior Pojects"*, selecione o item clicando o botão direito do mouse sobre ele para ativar o menu *popup* mostrado na figura anterior. Escolha a opção *"Delete"*. Se o item for um arquivo ou uma pasta, aparecerá a seguinte janela. Selecionando OK o item será removido do projeto.

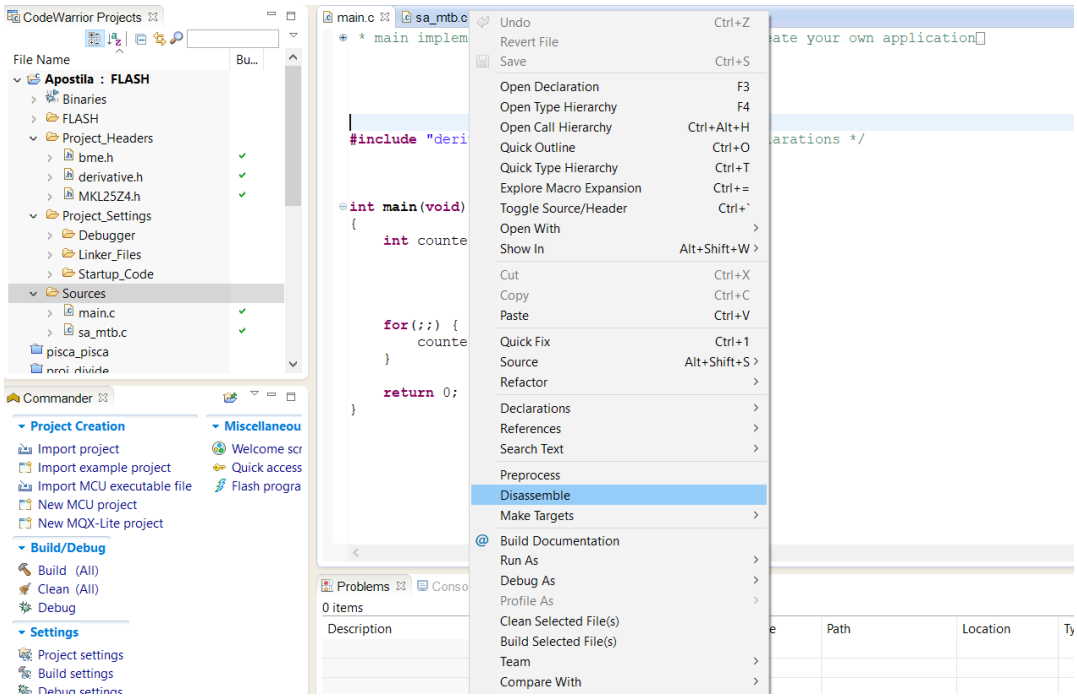


Se o item for uma pasta de projeto, aparecerá uma outra janela com a opção *"Delete project contents on disk (cannot be undone)"*. Se essa opção não for marcada, somente o nome do projeto é removido de *CodeWarrior Projects* e o projeto pode ser importado posteriormente (Seção).

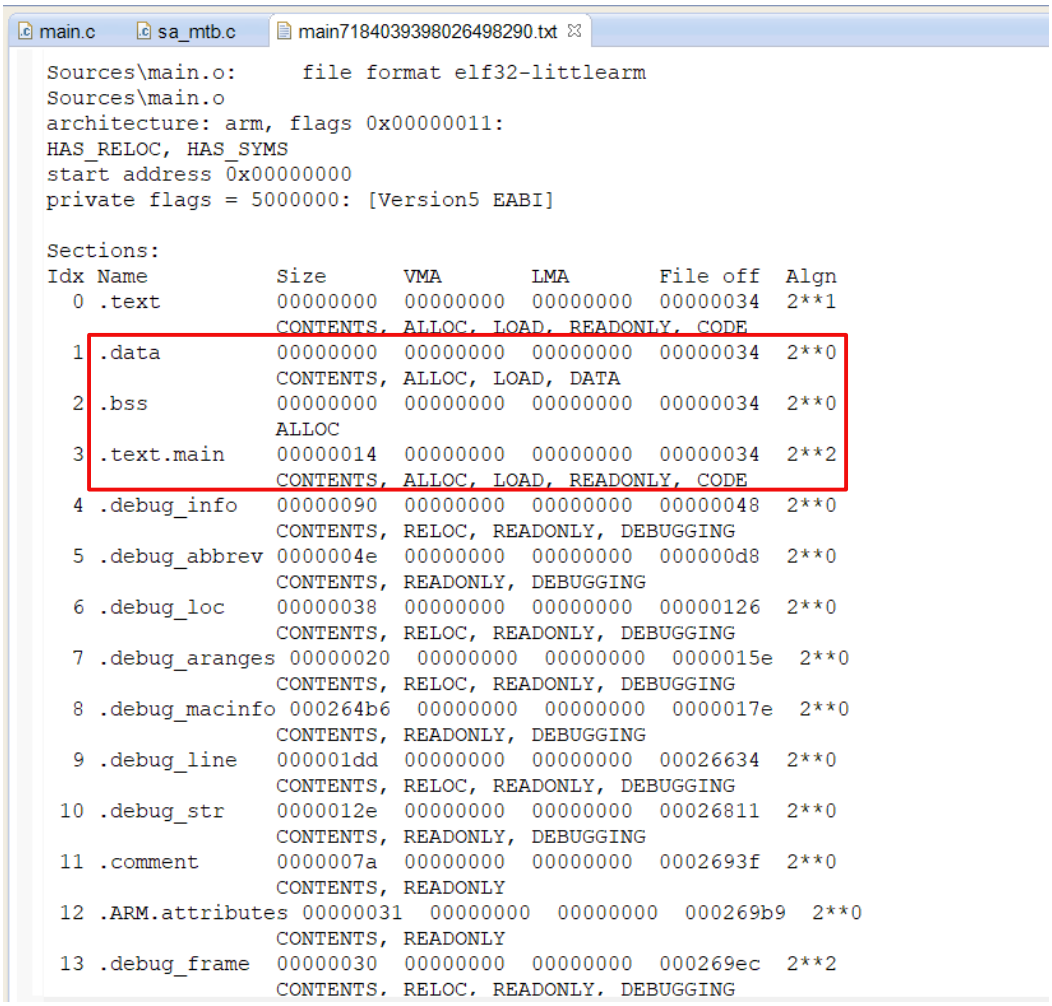


2.2.5 Mnemônicos e Instruções de Máquina em Código Hexadecimal

O IDE provê uma ferramenta *Disassemble* que nos permite visualizar o que o compilador entendeu do nosso programa e como traduziu o código-fonte para as instruções de máquina. Essa ferramenta pode ser ativada através do menu *popup* que aparece sobre a área de edição quando clicamos o botão direito do *mouse* sobre ela, como mostra na seguinte figura.



Será criada uma nova aba na área de edição contendo informações sobre as instruções e os dados do arquivo para o qual foi ativado *Disassemble*. Por exemplo, para o arquivo `main.c` são geradas as informações referentes às diferentes seções que compõem o resultado da sua "compilação". A figura abaixo mostra que há uma grande parcela de dados/instruções referentes à depuração e tais informações são simplesmente realocadas (RELOC). Novos espaços alocados (ALLOC) são para os segmentos `.text.main` (instruções e variáveis não modificáveis referentes à função main), `.data` (dados/variáveis com inicialização, de alocação "permanente" durante a execução do programa) e `.bss` (*block started by symbol* ou dados/variáveis sem inicialização, de alocação "permanente" durante a execução do programa) [16]. Como no arquivo `main.c` não foram definidas as variáveis estáticas nem globais, os tamanhos do espaço de memória alocado para `.text.main`, `.data` e `.bss` são, respectivamente 0x14, 0x00 e 0x00. Estes valores são mostrados na coluna *Size* das seções Idx = 1, 2 e 3 destacadas com a linha vermelha.

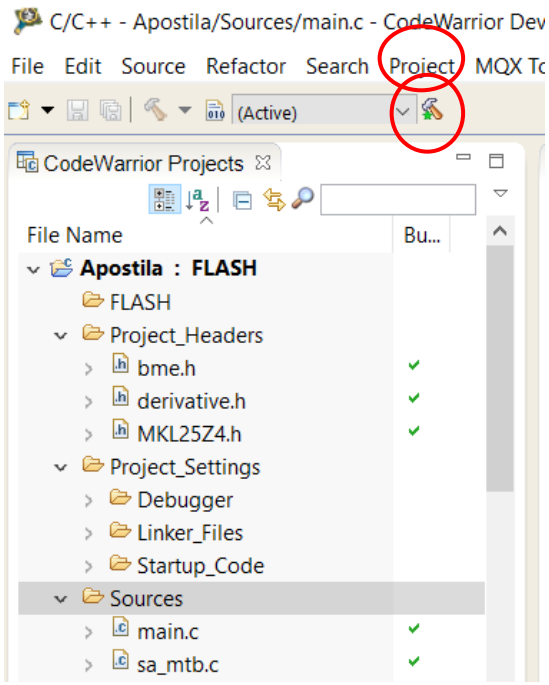


São incluídas no mesmo arquivo as instruções de máquina de `main.c` que ocupam 0x14 *bytes*, como mostra a figura que se segue. Note que é explicitado para cada instrução em C o bloco de instruções de máquina que o compilador "traduziu". Por exemplo, a instrução `counter++;` é traduzida num bloco de três instruções (*Thumb*) destacado pela linha vermelha. Na primeira coluna (destacada com linha azul) temos o endereço da instrução em relação à primeira instrução da função `main`, na segunda coluna (destacada com linha verde) temos instruções de máquina em hexadecimal, e na terceira coluna (destacada com linha lilás), mnemônicos correspondentes às instruções.

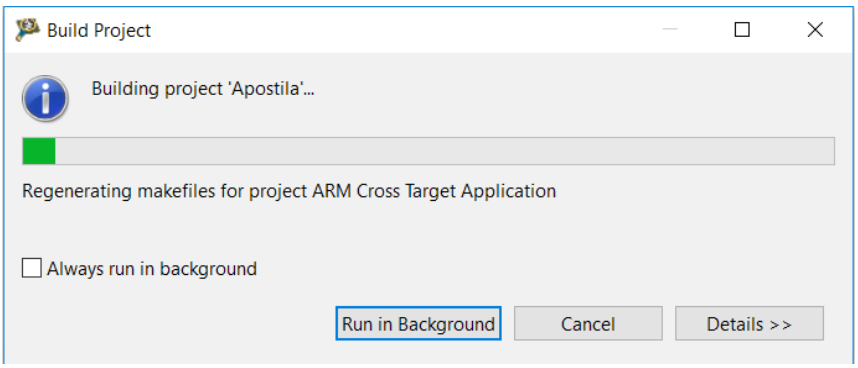
```
Disassembly of section .text.main:

00000000 <main>:
#include "derivative.h" /* include peripheral declarations */



int main(void)
{
   0:   b580        push    {r7, lr}
   2:   b082        sub sp, #8
   4:   af00        add r7, sp, #0
    int counter = 0;
   6:   2300        movs    r3, #0
   8:   607b        str r3, [r7, #4]



    for(;;) {
        counter++;
   a:   687b        ldr r3, [r7, #4]
   c:   3301        adds    r3, #1
   e:   607b        str r3, [r7, #4]
    }
  10:   e7fb        b.n a <main+0xa>
  12:   46c0        nop         ; (mov r8, r8)
```

2.3 Compilação, Linkagem e Geração do Arquivo elf

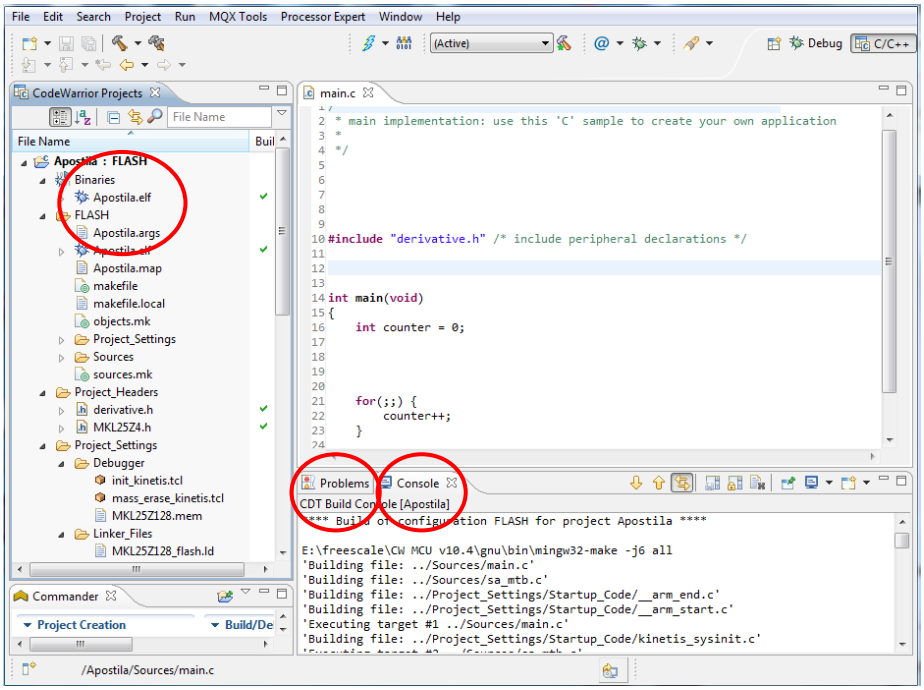
Quando finaliza a edição dos programas de um projeto, precisamos compilar todos os arquivos do projeto e linká-los para termos um código executável no formato elf (*Extensible Linking Formt*) [5.6]. Este arquivo contém as informações necessárias para linkagem, relocação de acordo com os dados contidos no arquivo *MKL25Z128_flash.ld*, e a execução do programa do projeto. Para isto, podemos clicar no ícone do martelo que aparece em vários locais, como na barra superior e na vista *Commander*. Alternativamente, podemos selecionar "*Project* > *Build Project*" na barra de ferramenta superior. Vale ressaltar que C/C++ é uma linguagem compilada. Qualquer modificação no código-fonte (C/C++/ASM) requer uma recompilação/remontagem e relinkagem (*Build Project*), pois precisa-se uma “retradução” do novo “texto” para linguagem de máquina.



Logo após, uma janela mostrando os passos do *Build* aparecerá. Aguarde até que ela desapareça.

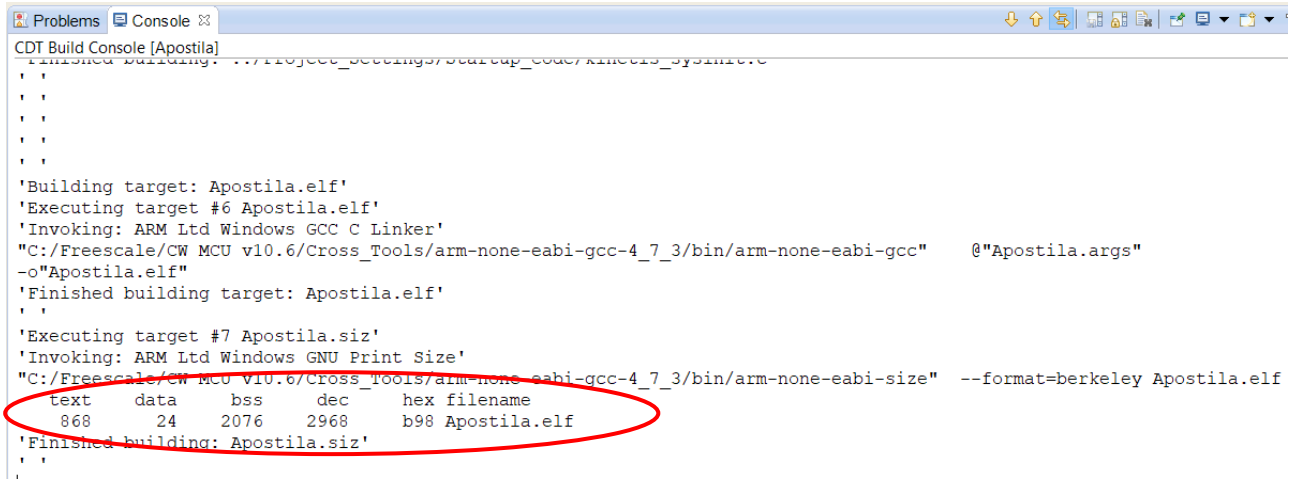


Observe na aba *Console* da janela inferior que o período em que a barra de progresso se evolui uma sequência de comandos são executados até aparecer a mensagem "*Finished building target: Apostila.elf*". Observe ainda que a pasta *FLASH* é utilizada para armazenar todos os arquivos intermediários, necessários à construção do código executável `Apostila.elf`. Mais especificamente, temos os arquivos de extensão `.args` (arquivos que contêm argumentos utilizados em linhas de comando que aparecem na aba *Console*), .d (arquivos que contêm as dependências do código-fonte para resolver os seus símbolos externos na linkagem) e.o (arquivos-objeto ou arquivos que contêm códigos resultantes da compilação dos códigos-fonte) nas pastas *FLASH/Startup_Code* e *FASH/Sources*.

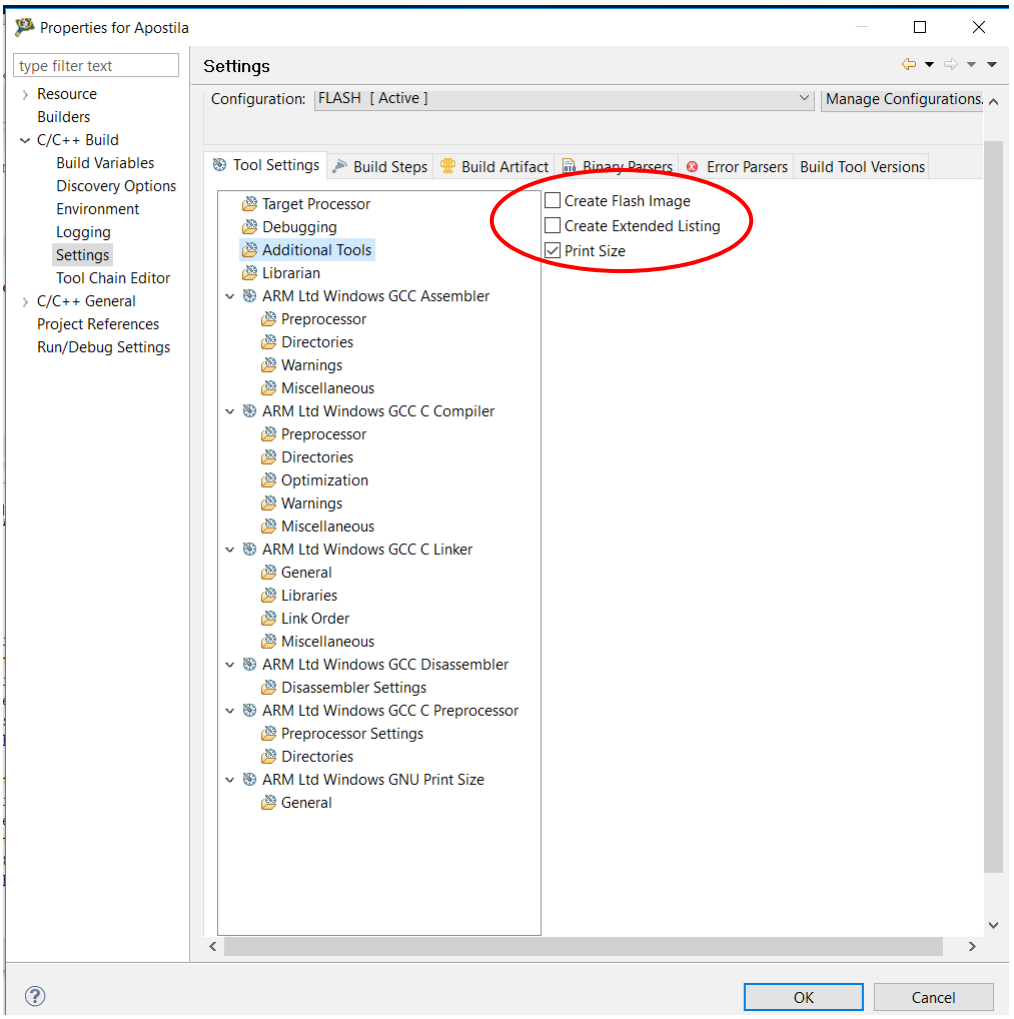


Caso ocorram erros na construção de um código executável, os problemas podem ser conferidos na aba *Problems*. Se as abas *Console* e *Problems* não estiverem visíveis no seu ambiente IDE, basta ativá-las pelo caminho "*Windows* > *Show View* > *Console*" e "*Windows* > *Show View* > *Problems*", respectivamente.

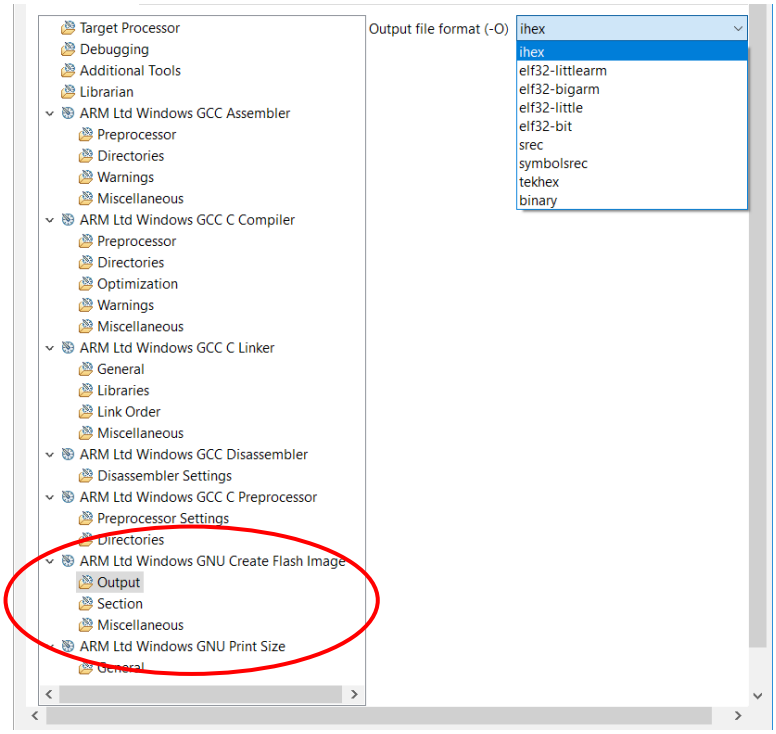
Distinguem-se em todos os *firmwares* 3 segmentos, `.text`, (instruções e valores constantes)`.data` (variáveis globais e estáticas com inicialização) e `.bss` (variáveis globais e estáticas sem inicialização) (Seção 2.2.5, [16]), Os tamanhos de cada segmento, como a soma deles em decimal (coluna dec) e hexadecimal (coluna hex), podem ser gerados no final da linkagem como mostra a figura que se segue



Para isso, é necessário ativar a função *Print Size* na janela abaixo que se abre ao seguirmos o caminho *Project* > *Properties*.



Além do formato elf (executable and linkable file), é possível gerar outros formatos, como (Intel) `hex` e (Motorola) `srec`, se ativarmos `Create Flash Image` mostrada na figura anterior. Após essa ativação, a ferramenta de geração de outros formatos é adicionada na lista de ferramentas como mostra na imagem abaixo através da qual podemos configurar mais um formato de saída (`Output file format`) além do formato padrão `elf`.



2.3.1 Diretivas de Compilação

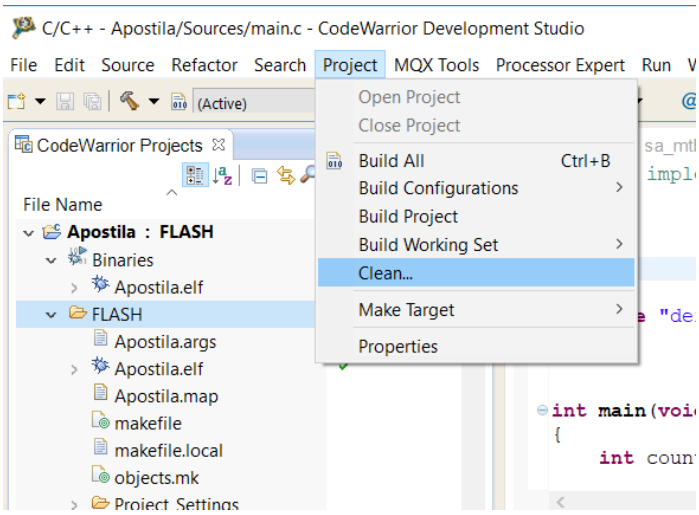
Antes de compilar o código-fonte, há um programa, conhecido como o pré-processador, que modifica o conteúdo deste conforme as diretivas de compilação. Estas diretivas são diferenciadas por terem o símbolo # no início da linha de um código-fonte. Dentre as diretivas mais utilizadas temos:

- #**include**: indica a inclusão dos códigos do arquivo especificado no programa antes da compilação.
- #**define**: indica a substituição do símbolo especificado, também conhecido como macro, pela sequência de caracteres fornecida antes da compilação.
- #**undef**: remove a definição de uma macro.
- #**ifdef**: junto com a diretiva **endif** indica a execução de diretivas dentro do escopo delimitado pelas duas diretivas quando uma macro específica estiver definida.
- #**ifndef**: como a diretiva **ifdef**, porém quando a macro especificada não estiver definida.
- #**if**: junto com a diretiva **endif** estabelece uma condição condicional para o pré-processador executar as diretivas dentro do escopo delimitado pelas duas diretivas.
- #**else**: como em linguagem C, indica as diretivas que devem ser executadas quando as outras alternativas não forem satisfeitas.
- #**elif**: equivalente ao comando "else if" em linguagem C, indica as diretivas que devem ser executadas sob uma dada condição alternativa.
- #**endif**: indica o fim do escopo de uma condição.

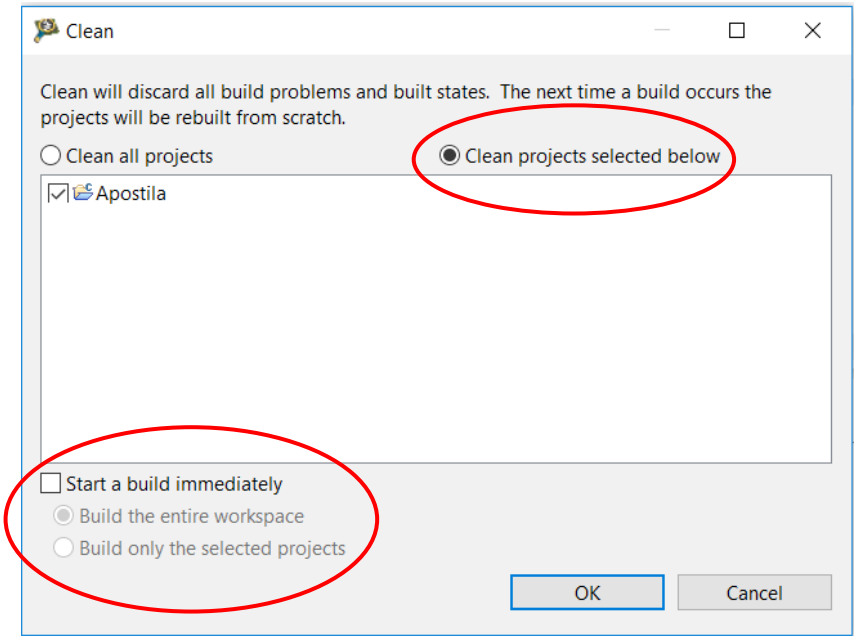
O código gerado pelo IDE utiliza a diretiva `include` para que o pré-processador insira todos os códigos do arquivo de cabeçalho `derivative.h` no arquivo `main.c` antes da compilação desta. Observe que esta inclusão é feita de forma recursiva até que todas as diretivas na hierarquia de inclusão sejam resolvidas. Vale ressaltar que, como neste caso específico, a função `main` não usa as macro e os tipos de dados definidos em `derivative.h`, é recomendável remover a linha de inclusão para que `main.c` não seja "inchado" desnecessariamente.

2.3.2 Remoção do Código Executável e dos Arquivos Intermediários

Vale comentar que é uma boa prática garantir sempre que as pastas do projeto estejam "limpas" dos arquivos intermediários antes de compilá-lo. Isto significa apagar os arquivos nas pastas *FLASH* e *Binaries*, gerados por compilações anteriores. Para isso, seleciona-se "*Project > Clean*"



Na janela que se abre, selecione a opção *Clean projects selected below* e verifique se seu projeto está selecionado na lista. Se quiser que gere imediatamente após a limpeza um novo executável, marque a caixa *Start a build immediately* e selecione a opção *Build only the selected projects*. Caso queira limpar apenas o projeto, deixe desmarcada a caixa *Start a build immediately.*



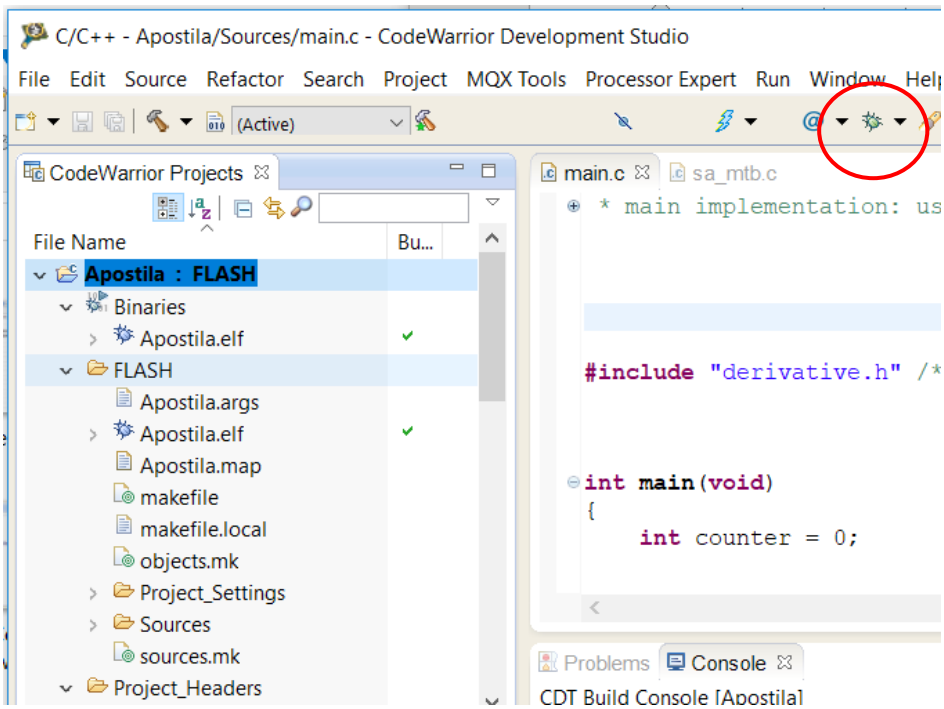
2.4 Transferência do Código para Microcontrolador

Gerado o código executável `Apostila.elf` no computador hospedeiro, precisamos transferí-lo para o microcontrolador onde ele é de fato executado. Para isso, basta conectar a porta miniUSB "*SDA*" da placa FRDM-KL25 a uma porta USB do computador-hospedeiro onde se encontra o *firmware*. Certifique-se que o projeto `Apostila` esteja selecionado/ativado (em negrito) na janela de *CodeWarrior Projects e* que o *kit* FRDM-KL25Z seja reconhecido como um dos periféricos (*OpenSDA*)

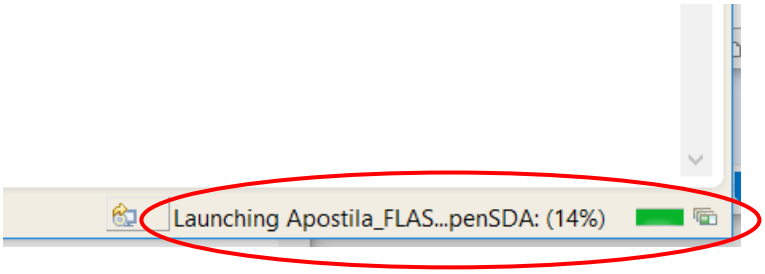
da porta serial no *Device Manager* pelo caminho de comandos do Windows *Control Panel* > *System* > *Device Manager* > *Ports (COM &LPT)*.

O IDE CodeWarrior suporta dois modos de execução de um *firmware*: sem ou com depuração (Seção 2.5). O modo de depuração é muito usado na fase de desenvolvimento enquanto o modo sem depuração é destinado para os usuários finais.

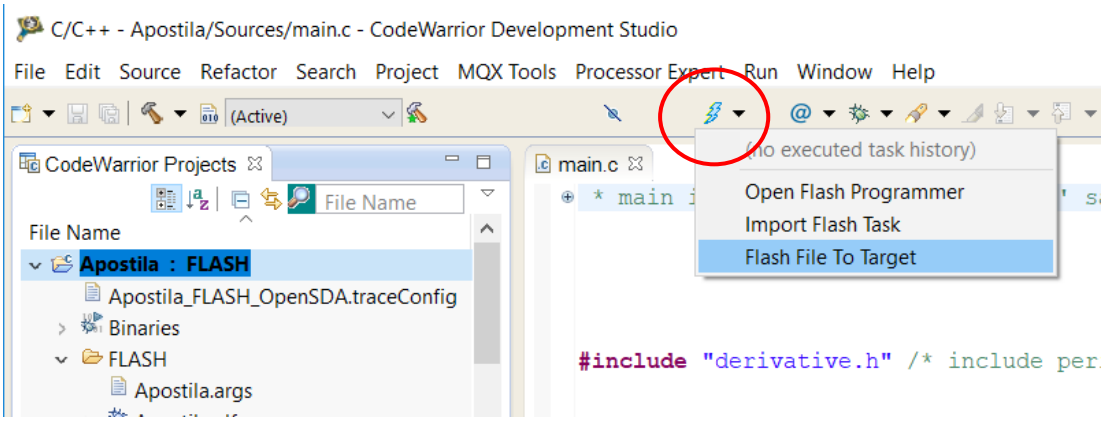
Para instalar um *firmware* no modo de depuração, a forma mais direta é clicar no ícone besouro destacado na seguinte figura. O ícone pode ser encontrado em várias vistas do IDE, como na barra superior e na vista `Commander`. Alternativamente, pode seguir o caminho "*Run* > *Debug*", ou ainda pressionar somente a tecla F11.



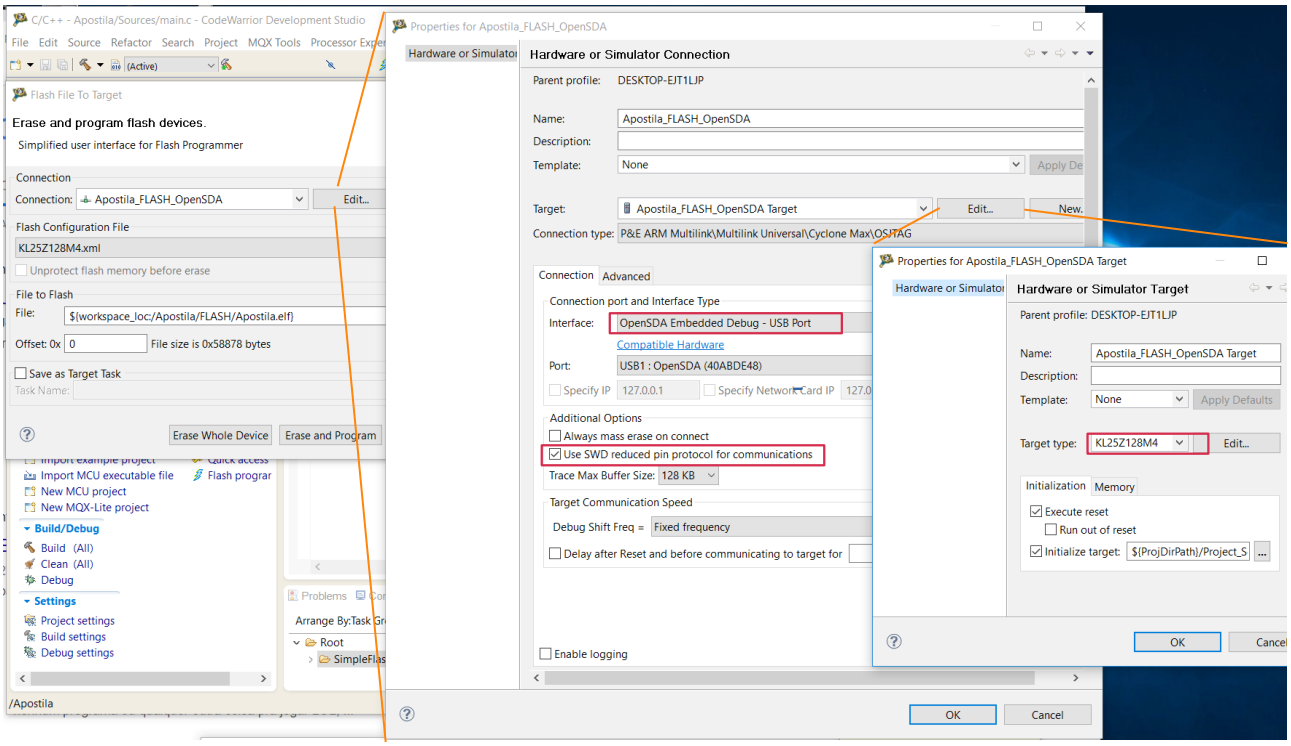
O progresso de transferência ou da carga do código no microcontrolador é indicado pela barra de progresso no canto direito inferior do IDE.



Para instalar um *firmware* no modo de usuários finais, podemos seguir o caminho *"Run* > *Run"* para transferir o executável ao microcontrolador no contexto de um projeto do IDE. Caso tenhamos um *firmware* de terceiros, é também possível transferí-lo com uso de *"Flash Programmer"* cujo ícone é destacado na figura.



Antes da programação da memória *flash*, é necessário configurar tanto o microcontrolador-alvo quanto a conexão. Para os microcontroladores do LE-30, configuramos como a conexão e o microcontrolador-alvo OpenSDA e KL25Z128M4, respectivamente.

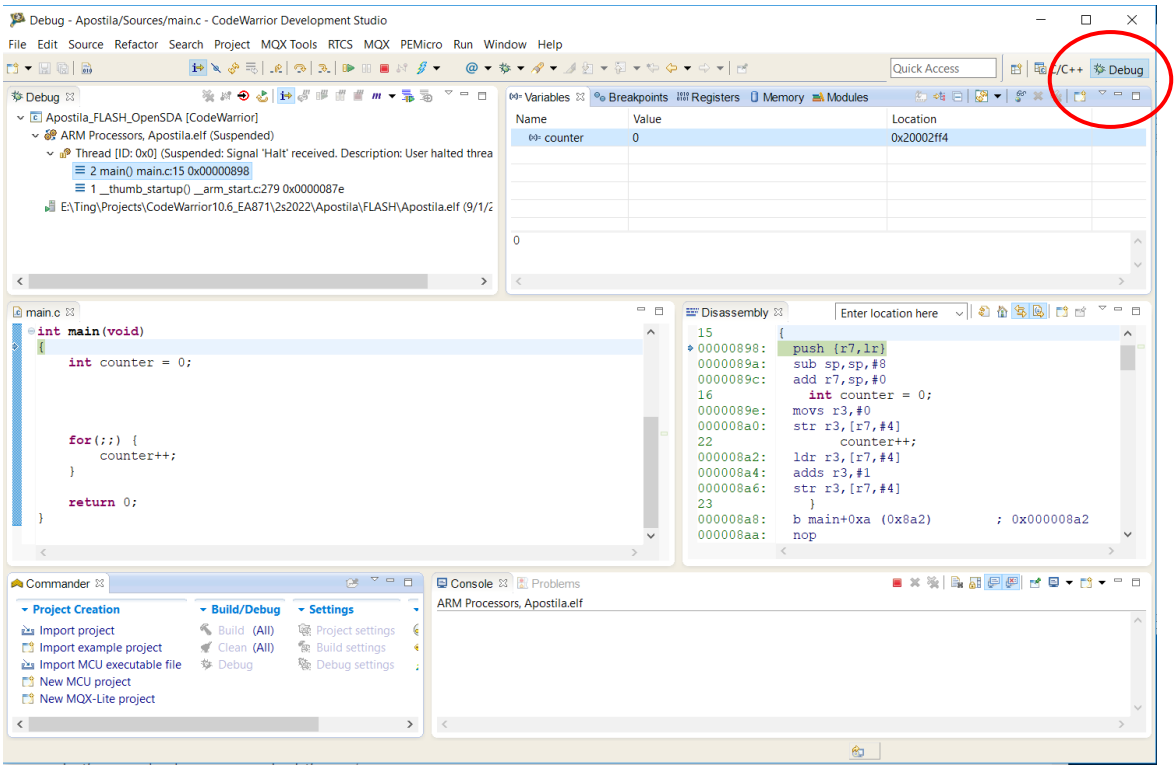


2.5 Depuração

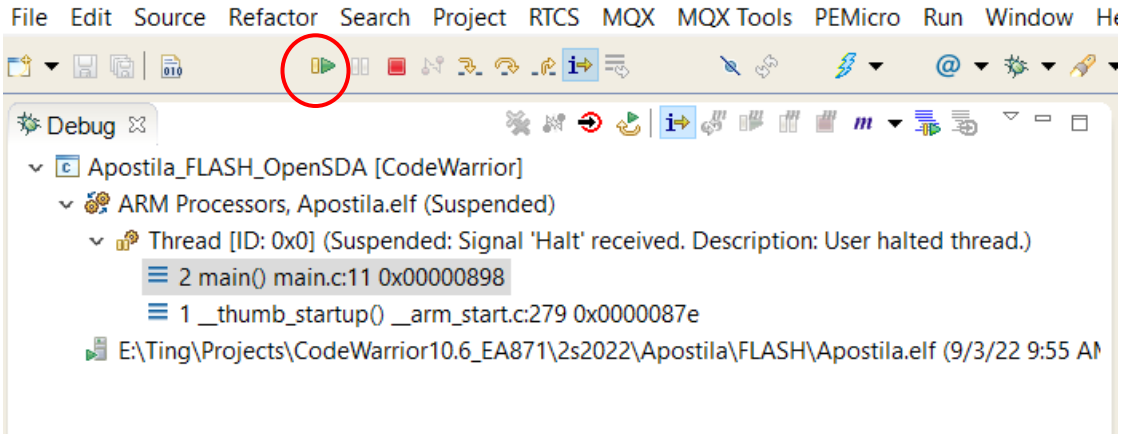
Assim que finalizar a transferência de um código executável no modo de depuração, a perspectiva é mudada automaticamente para a de **Depuração** (*Debug*). Depuradores são aplicativos que permitem ao programador monitorar a execução de um programa passo a passo, pará-lo, reiniciá-lo, ativar os *breakpoints* (pontos de parada), alterar o conteúdo dos espaços de endereços, e voltar a um ponto específico de execução.

A perspectiva de depuração é composta por várias janelas (*views*) agrupadas em diferentes regiões da área de interface. Por padrão, a aba *Debug* fica no canto superior esquerdo, aba Edição (de código-fonte, no caso, *main.c*) no lado esquerdo, um conjunto de abas *Variables*, *Registers*, *Breakpoints*, *Memory*, *Modules* agrupadas no canto superior direito, aba *Disassembly* no lado direito e as abas *Problems* e *Console* no canto inferior direito como vimos anteriormente.

Veja ainda que o programa foi automaticamente executado até o endereço 0x00000898 (linha destacada em verde na aba *Disassembly*) que corresponde ao início da rotina *main* (linha destacada em verde na aba de Edição), embora tenhamos visto no Capítulo 1 que o endereço inicial do contador de programa seja o endereço da rotina *__thumb_startup*. E na seção 1.1.3 (página 11) em [1] encontramos a informação de que os registradores SP e PC são carregados com os valores dos endereços 0x00000000 e 0x00000004, respectivamente, em *Reset* do sistema. Na seção 2.5.6 vamos ver que o conteúdo inicial de PC não é o endereço 0x00000898. Em vista disso, podemos concluir que o IDE não só carregou o programa executável no microcontrolador como também se encarregou de executar todas as instruções de inicialização parando na primeira instrução do código implementado pelo deenvolvedor.



Clicando no triângulo verde *(Resume)* na barra de ferramenta da aba *Debug*, ou pressionando F8, a execução do programa será retomada.



Da esquerda para direita, temos na barra de ferramenta da figura acima os seguintes ícones:

- *Resume*: retoma a execução contínua do programa pausado (por *breakpoints* ou por *Suspend*) com o endereço corrente do PC.

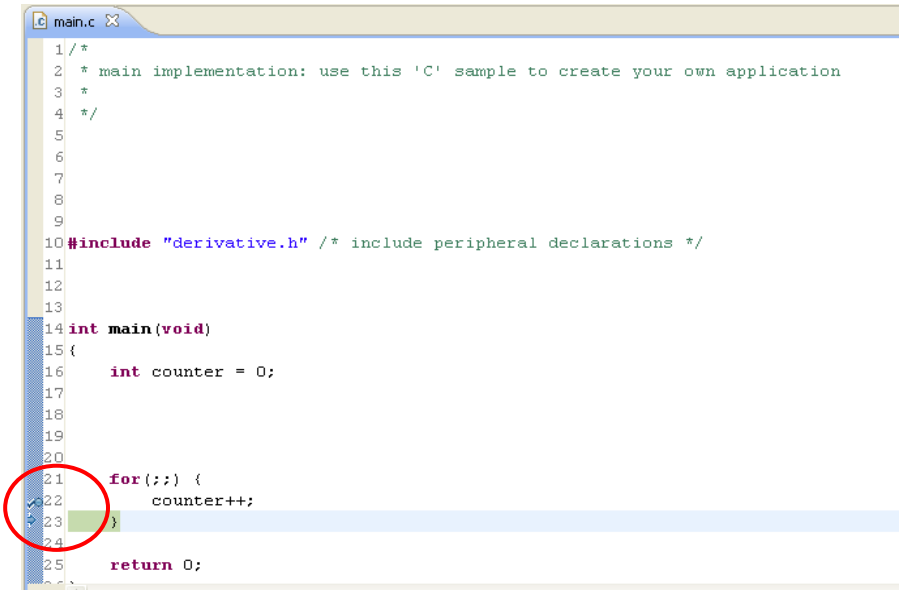
- *Suspend* (símbolo de pausa): suspende a execução do programa, ou seja, o PC não é incrementado.
- *Teminate* (quadrado vermelho): finaliza a execução do programa.
- Disconnect: (uma linha em forma de N): continua executando o programa sem a conexão IDE.
- *Step Into*: avance, no modo de fluxo de execução de controle manual, uma instrução no código em C se o modo *Instruction Stepping* estiver desabilitado. O avanço será por instrução no código *assembly* se o modo *Instruction Stepping* estiver habilitado.
- *Step Over*: avance, no modo de fluxo de execução de controle manual, uma instrução no código em C na vista/janela de código-fonte, se o modo *Instruction Stepping* estiver desabilitado. Se o modo *Instruction Stepping* estiver habilitado, o avanço será na janela *Disassembly* por instrução *assembly*. A diferença em relação ao comando anterior é que se a instrução for uma chamada à subrotina o depurador não entra para dentro da subrotina.
- *Step Return*: retrona à instrução da rotina que chamou a rotina corrente em execução no modo de fluxo de execução de controle manual.
- *Instruction Stepping Mode*: alterna o modo de avanço por instrução em alto nível (na aba de Edição) ou por instrução em *assembly* (na vista *Disassembly*). Seu ícone aparece também na barra de ferramenta da aba *Debug*.

Quando se suspende a execução contínua de um programa, podemos não só monitorar o conteúdo das variáveis como também o conteúdo dos registradores. Podemos modificar o conteúdo das variáveis ou dos registradores mapeados no espaço de endereços, editando diretamente os espaços correspondentes a cada endereço. As instruções executadas posteriormente acessarão estes novos valores. Isso nos permite isolar a detecção de erros dur`ante o processo de depuraçã`o.

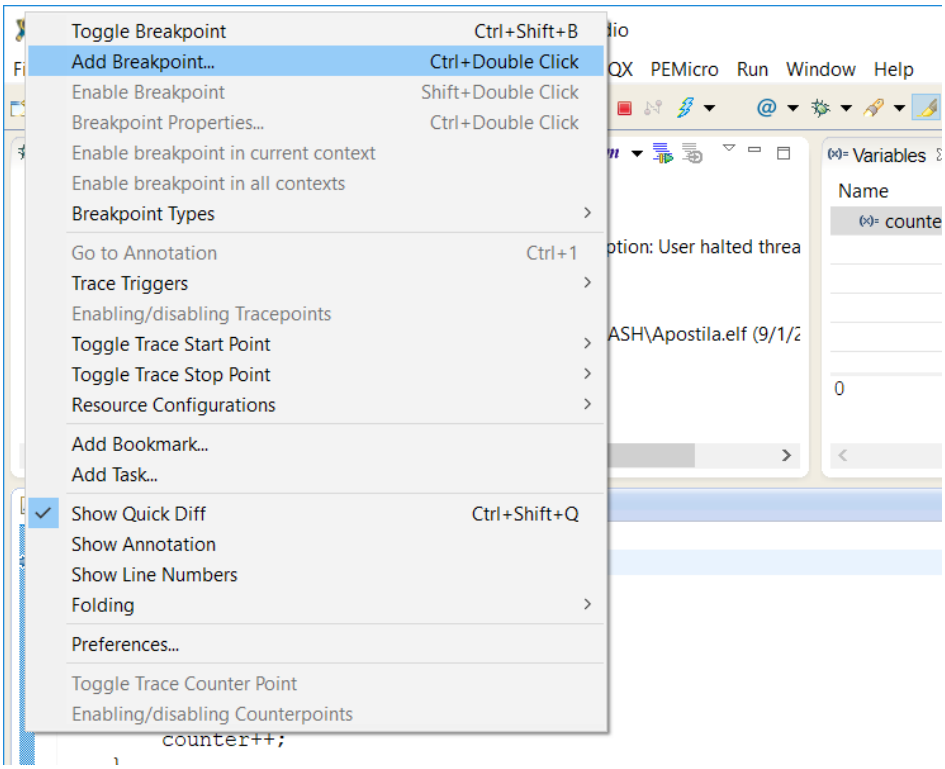
Note que os comandos de avanço manual no fluxo de controle de instruções só valem para operações que não sejam sobre pontos flutuantes. As operações em pontos flutuantes são emuladas e os códigos-fonte não são acessíveis. Portanto, é mostrada uma mensagem "`No source available for …`".

2.5.1 Aba *Breakpoints*

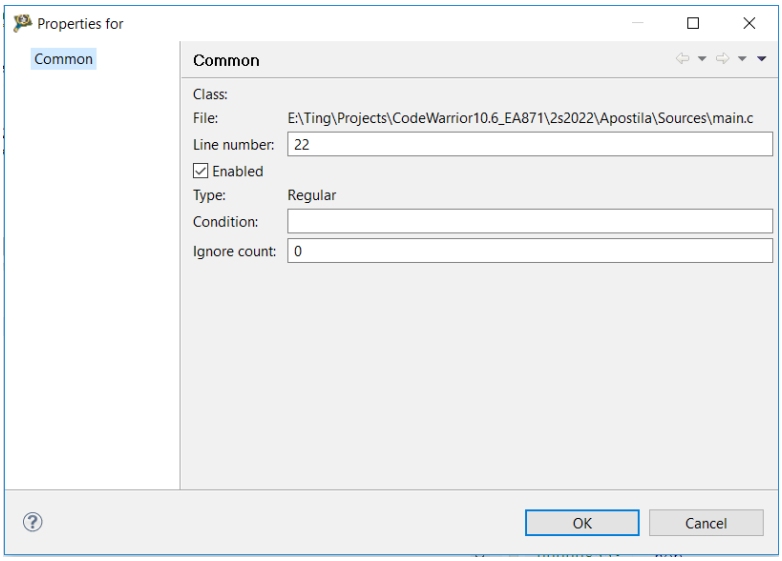
Pode-se definir *breakpoints* no código, para que o mesmo pare toda vez que sua execução chegue a estes pontos. Uma forma de especificar um *breakpoint* é através de duplo-clique no botão esquerdo do *mouse* sobre a faixa lateral azulada da área de edição e na linha de instrução que se deseja parar. Por exemplo, foi dado um duplo-clique à esquerda da linha "`counter++;`" e apareceu um pequeno círculo azul indicando que foi reconhecido pelo IDE o ponto de parada. O programa parará sempre antes da execução desta linha. Pode-se remover os *breakpoints* clicando com o botão direito sobre a faixa lateral azulada. Aparecerá um menu de opções, entre as quais está "*Toggle Breakpoint*". Selecionada esta opção, o ponto de parada será removido.



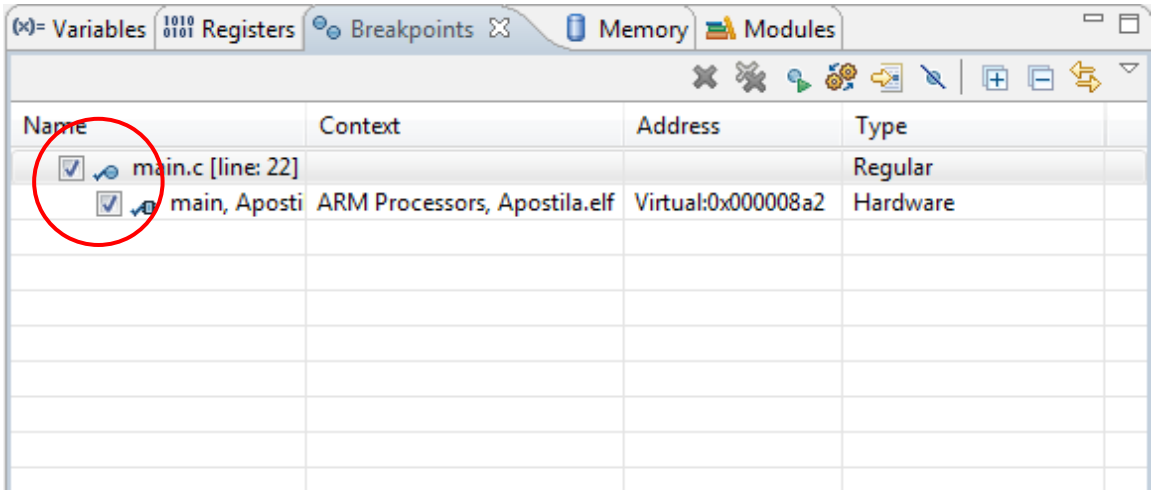
Alternativamente, podemos usar *"Add Breakpoint ..."* para definir e ativar os pontos de parada num fluxo de controle. Na figura que se segue mostra o menu *popup* que é ativado com o duplo-clique no botão esquerdo do *mouse* sobre a faixa lateral azulada da área de edição. Note que é o mesmo menu que contém o item *"Toggle Breakpoint"*.



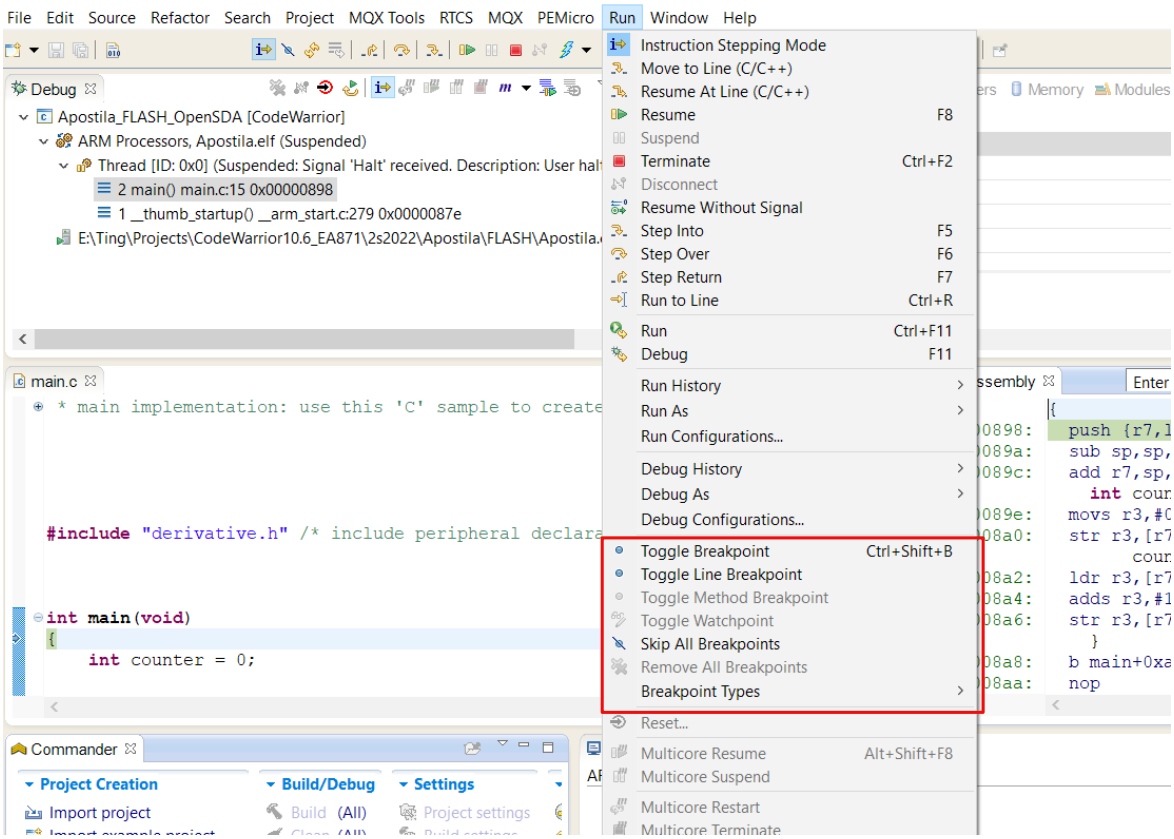
Ao selecionar o item *"Add Breakpoint ..."* surge uma nova janela mostrada na figura abaixo através da qual podemos editar explicitamente o ponto de parada pelo número da linha de instrução do código (*Line number*) como controlar a sua ativação (*Enabled*).



A aba *Breakpoints* ilustrado na figura abaixo nos permite gerenciar os pontos de parada. No nosso caso, com um *breakpoint* habilitado, temos na janela somente uma entrada. Vale ressaltar que a quantidade total de *breakpoints* ativados suportada no IDE é **2 (dois)**. Através dos *checkboxes* destacados com a linha vermelha podemos ativar e desativar dinamicamente os pontos de parada habilitados durante a execução de um programa.

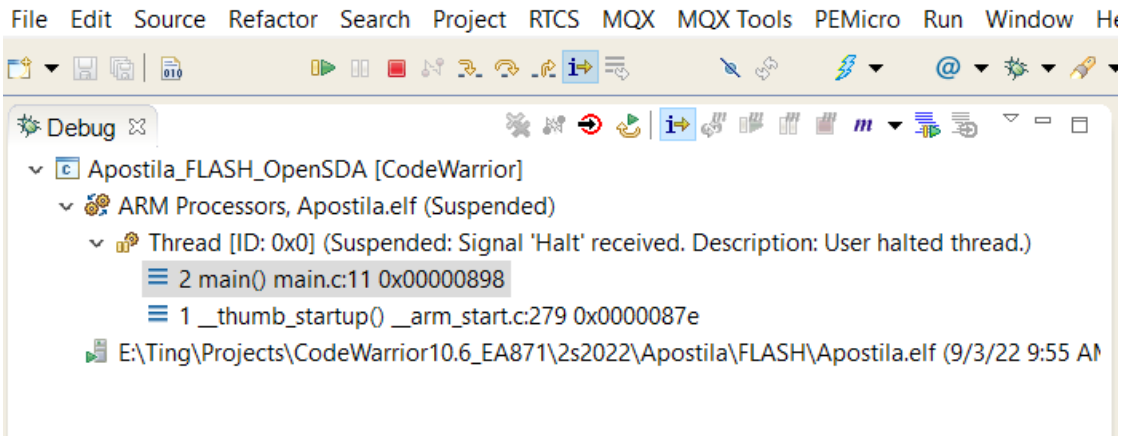


Uma outra alternativa para acionar os comados relacionados aos pontos de parada é através do item *Run* que fica na barra de ferramenta da janela principal da perspectiva *Debug*. Ao clicar sobre este item, aparece um menu *pop-up* que contém várias ações relacionadas aos *breakpoints*, destacadas pela linha vermelha na seguinte figura. O ícone *"Skip All Breakpoints"* é encontrado em várias vistas. Ao ativá-lo, todos os pontos de parada habilitados são ignorados no fluxo de execução.



2.5.2 Aba *Debug*

A barra de ferramenta da aba *Debug* é composta pelos seguintes ícones, da esquerda para direita:

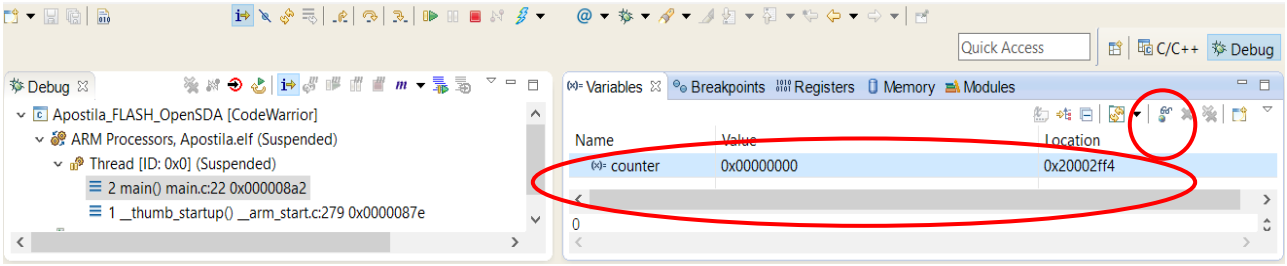


- *Reset* (círculo vermelho com seta preta): reinicializa o microcontrolador e o contador de programa (PC) é carregado com o endereço inicial da rotina *__thumb_startup* (Seção 2.4.6).
- *Restart*: retoma a execução do programa a partir da rotina `main` do programa de projeto.
- *Instruction Stepping Mode.*

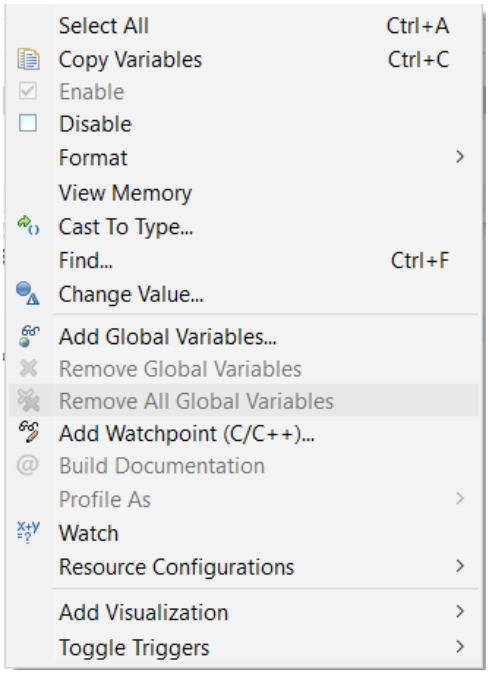
2.5.3 Aba *Variables*

Na aba "*Variables*", pode-se visualizar o valor armazenado nas posições de memória (*Value*) como também as próprias posições de memória (*Location*) que correspondem às **variáveis locais** da

rotina em execução. No nosso caso é a variável `counter`. O IDE suporta os seguintes formatos: *default*, decimal, hexadecimal, octal e binário. O formato *default* é o formato assumido pelo compilador da linguagem usada. Para consultar o tamanho do espaço de memória alocado a uma variável, usa-se o **formato hexadecimal ou binário** em *CodeWarrior*. Foi configurado para `counter` o formato hexadecimal. Mesmo que seja 0 (1 dígito na base decimal) o seu valor, a sua representação em hexadecimal é 0x00000000 indicando a ocupação de um espaço de 4 *bytes*. Podemos incluir ainda nesta aba as variáveis globais do programa de projeto clicando no ícone óculos destacado barra de ferramenta da aba *Variables* (*"Add Global Variables ..."*).



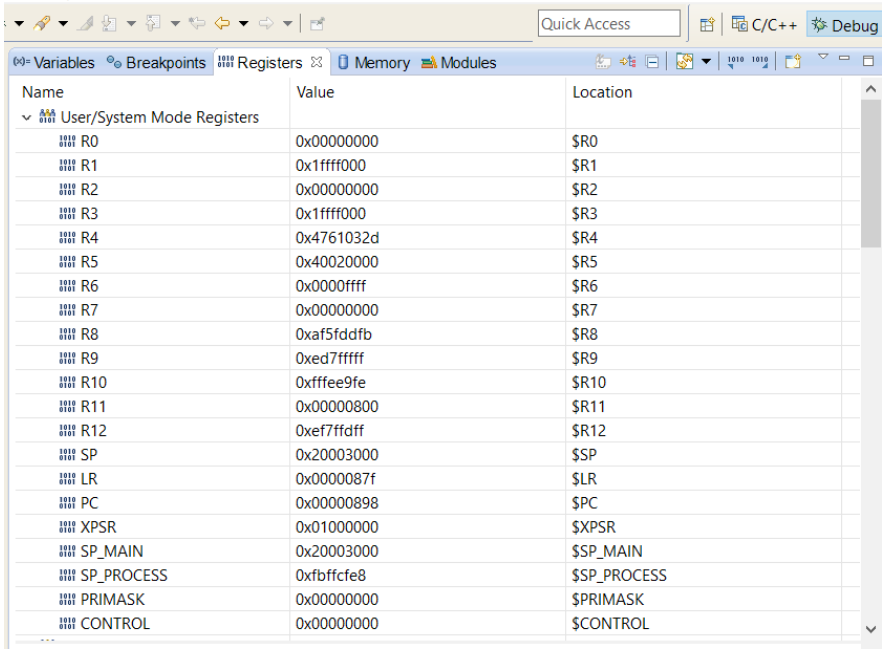
O valor da variável é sempre atualizado conforme o fluxo de execução do programa pausado. E ele pode ser modificado editando o novo valor diretamente no campo *"Value"*. Clicando o botão direito na área de variáveis, aparece um menu *pop-up*, conforme mostra a figura abaixo, através do qual pode-se alterar o valor da variável (*Change Value ...*), escolher o formato que desejamos renderizar a variável destacada por uma faixa azul (*Format*). Veja ainda que, via o menu, podemos acrescentar as variáveis globais (*Add Global Variables ...*) à lista.



O valor das variáveis pode ser também conferido na aba *Memory* com uso do endereço correspondente que aparece na (terceira) coluna *Location* da aba *Variables*.

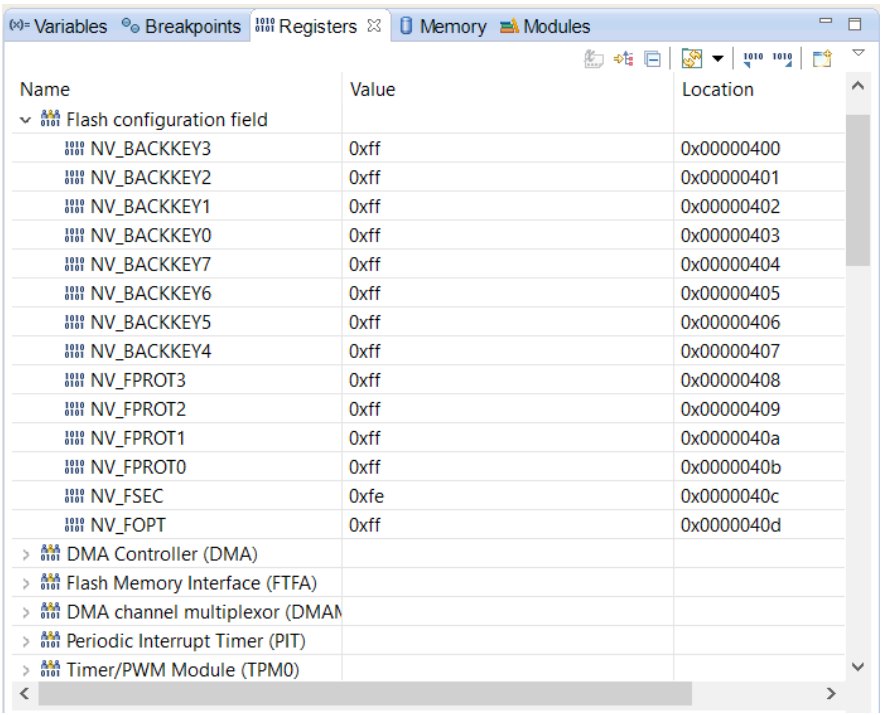
2.5.4 Aba *Registers*

Na aba "*Registers*" pode-se visualizar o valor (*Value*) armazenado em todos os registradores de todos os módulos integrados ao microcontrolador, como também os endereços em que tais registradores estão mapeados no espaço de endereços da memória. Como as variáveis, podemos modificar o conteúdo dos registradores como se fossem um elemento da memória. A figura abaixo mostra todos os registradores que ficam no núcleo (processador) do microcontrolador. São os elementos de memória que não contém endereços na terceira coluna (*Location*), pois não estão mapeados no espaço de endereços de 32 *bits* do microcontrolador.



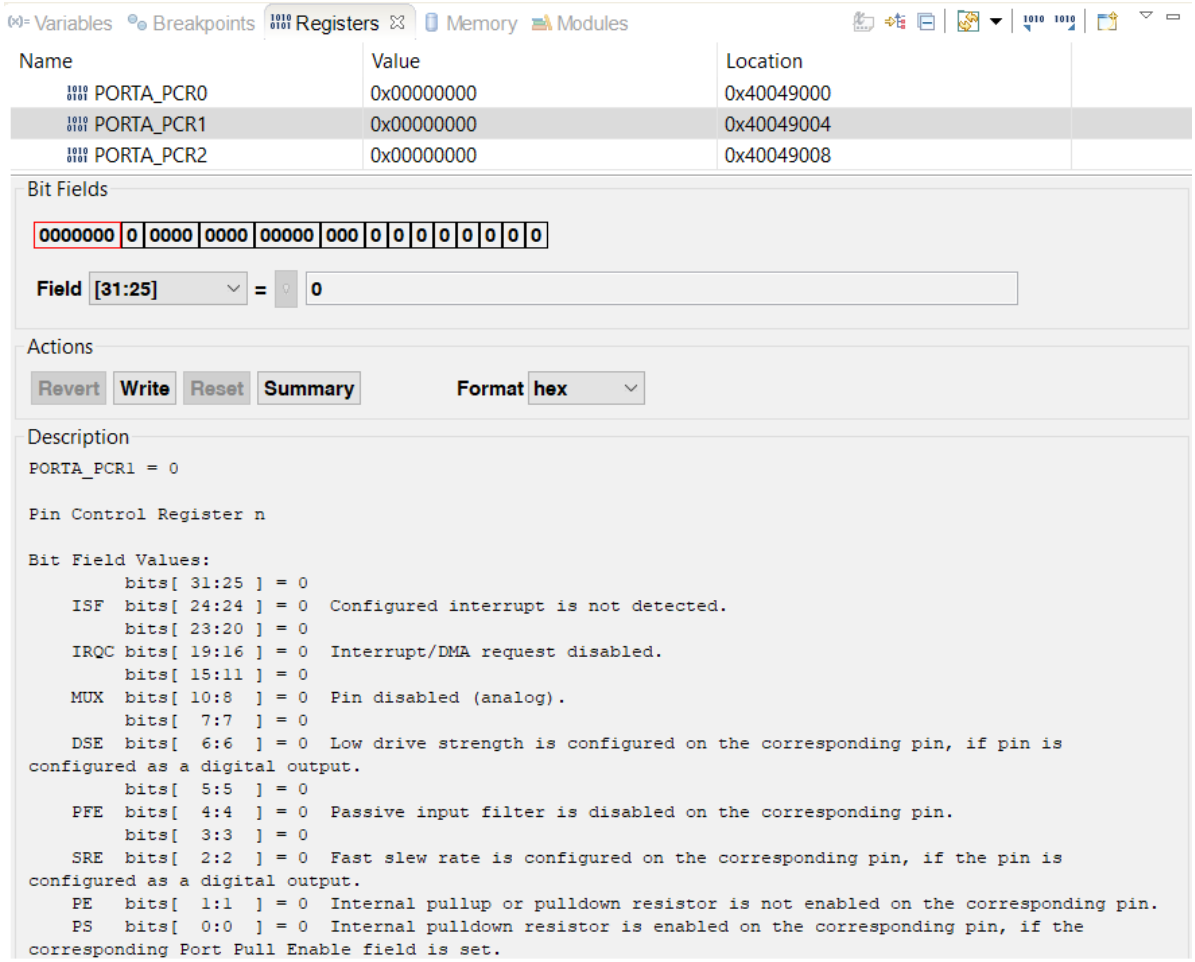
Name	Value	Location
User/System Mode Registers		
R0	0x00000000	$R0
R1	0x1ffff000	$R1
R2	0x00000000	$R2
R3	0x1ffff000	$R3
R4	0x4761032d	$R4
R5	0x40020000	$R5
R6	0x0000ffff	$R6
R7	0x00000000	$R7
R8	0xaf5fddfb	$R8
R9	0xed7fffff	$R9
R10	0xfffee9fe	$R10
R11	0x00000800	$R11
R12	0xef7ffdff	$R12
SP	0x20003000	$SP
LR	0x0000087f	$LR
PC	0x00000898	$PC
XPSR	0x01000000	$XPSR
SP_MAIN	0x20003000	$SP_MAIN
SP_PROCESS	0xfbffcfe8	$SP_PROCESS
PRIMASK	0x00000000	$PRIMASK
CONTROL	0x00000000	$CONTROL

Todos os registradores dos módulos integrados ao microcontrolador são mapeados no espaço de endereços da memória. Esses registradores são organizados por módulos como mostra na figura que se segue. Ao expandirmos um item, abre-se uma lista de registradores integrados ao módulo com o seu nome (na coluna *Name*), conteúdo (na coluna *Value*) e endereço (na coluna *Location*) mapeado no espaço de endereços da memória. Na figura foi expandido o módulo *Flash*.



Name	Value	Location
Flash configuration field		
NV_BACKKEY3	0xff	0x00000400
NV_BACKKEY2	0xff	0x00000401
NV_BACKKEY1	0xff	0x00000402
NV_BACKKEY0	0xff	0x00000403
NV_BACKKEY7	0xff	0x00000404
NV_BACKKEY6	0xff	0x00000405
NV_BACKKEY5	0xff	0x00000406
NV_BACKKEY4	0xff	0x00000407
NV_FPROT3	0xff	0x00000408
NV_FPROT2	0xff	0x00000409
NV_FPROT1	0xff	0x0000040a
NV_FPROT0	0xff	0x0000040b
NV_FSEC	0xfe	0x0000040c
NV_FOPT	0xff	0x0000040d
DMA Controller (DMA)		
Flash Memory Interface (FTFA)		
DMA channel multiplexor (DMAM		
Periodic Interrupt Timer (PIT)		
Timer/PWM Module (TPM0)		

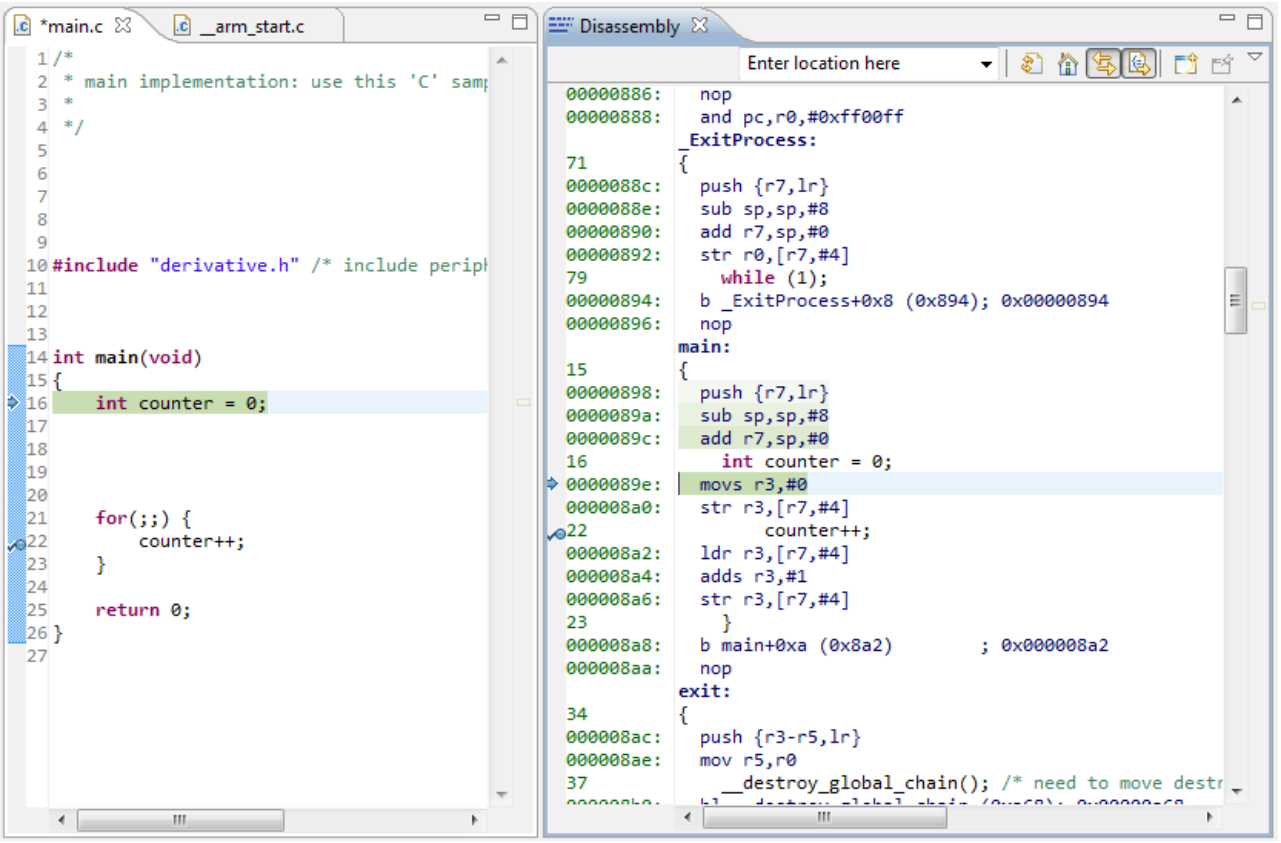
Ao selecionarmos um registrador, é exibido na parte inferior da aba o conteúdo do registrador por função associada aos seus *bits*. Podemos, através desta interface, checar e modificar manualmente as configurações da maioria dos elementos do microcontrolador. Abaixo é exemplificada a visualização da função dos *bits* do registrador `PORTA_PCR1` do módulo `PORTA` pela aba *Registers*.



2.5.5 Aba *Disassembly*

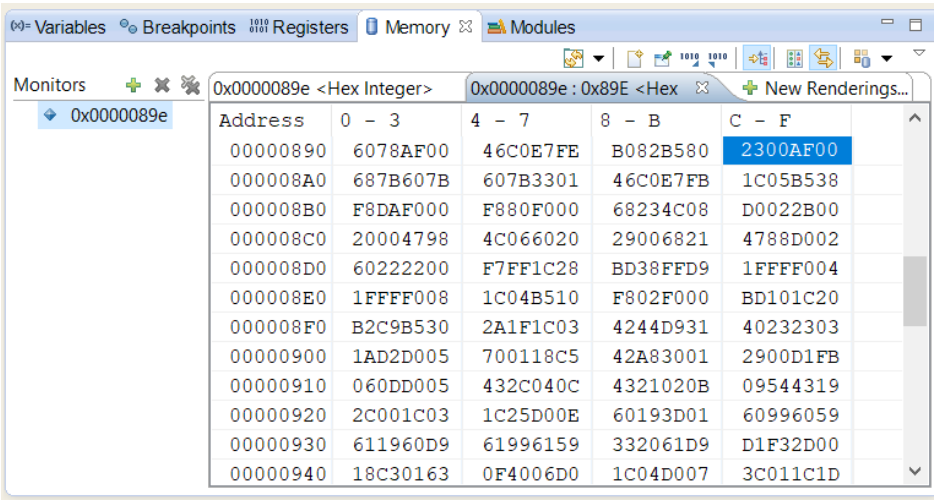
Nesta janela são exibidos os códigos em *assembly* do programa do projeto. Estes códigos estão sincronizados com os códigos em C, de forma que as linhas destacadas ou indicadas nas duas janelas são sempre as mais próximas em termos de correspondência. Na figura abaixo a instrução em C "int counter = 0;" corresponde à instrução em *assembly* "movs r3,#0" que está armazenado no endereço 0x0000089e. Como já mencionado nas Seções 2.5/2.5.2, é possível executarmos um programa por instrução de máquina no ambiente IDE, desde que ativemos o modo *Instruction Stepping* e demos passos com o comando *Step over* (não entrar na subrotina quando passarmos por ela) ou *Step into* (entrar na subrotina).

A primeira coluna, em verde, na aba *Disassembly* mostra os endereços efetivos de cada instrução de máquina e as linhas de códigos em C. Observe que o endereço da primeira instrução, que era 0x0 após a compilação na Seção 2.2.5, foi realocado para 0x00000898 quando as instruções foram transferidas para o microcontrolador. Na segunda coluna, temos as instruções em C (mais identadas) seguidas das instruções de máquina correspondentes (em azul).

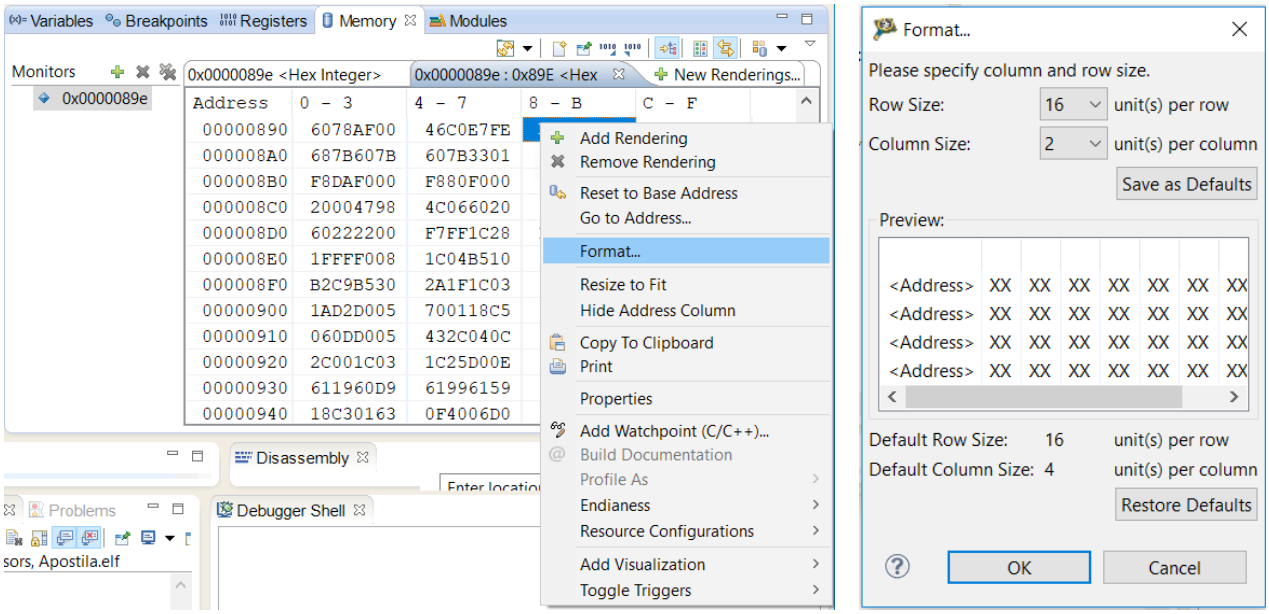


2.5.6 Aba *Memory*

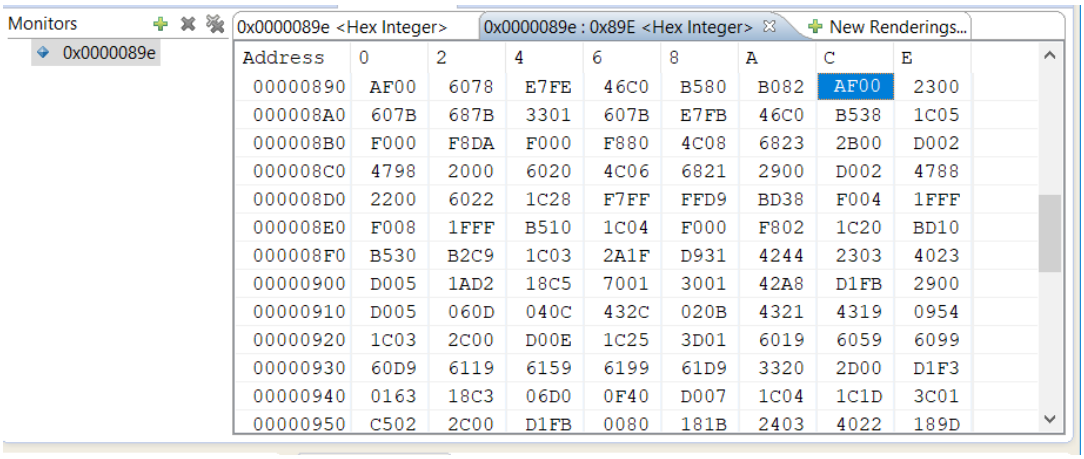
A aba *Memory* nos permite monitorar o conteúdo do espaço de endereços. Por exemplo, podemos monitorar o conteúdo de um bloco de endereços a partir do endereço 0x0000089e onde está armazenada a instrução “movs r3,#0”, clicando em + verde. Na caixa de diálogo que aparecerá será inserido o endereço de interesse. Por padrão, são renderizados 16 *bytes* por linha e os 16 *bytes* são agrupados em grupos de 4 *bytes* formando 4 colunas por linha, conforme ilustra a figura que se segue. Conforme a Seção A6.7.39 em [4], o código binário da instrução em consideração é 0b001 00 011 00000000. Em hexadecimal, o código correspondente é 0x2300. Veja o valor 0x2300 como parte de uma palavra de 4 *bytes* na célula destacada em azul. Aparentemente, a instrução está alocada no endereço 0x000089c!



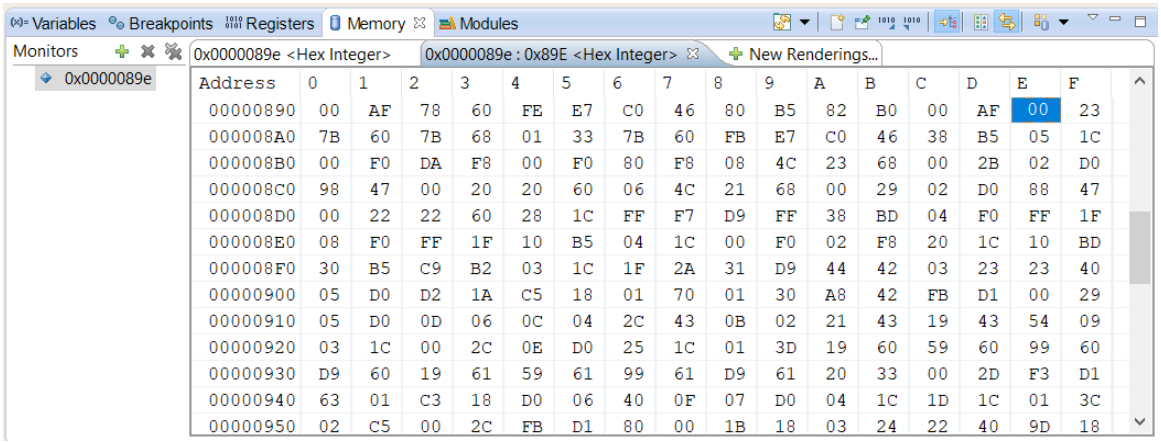
Como o repertório de instruções do processador é *Thumb* (16 *bits*), é mais conveniente agruparmos os *bytes* em grupos de 2. Para isso, podemos reconfigurar o formato de renderização clicando o botão direito do *mouse* sobre a área de dados de memória para abrir um menu *popup*. A partir da seleção de *Format* desse menu (imagem esquerda na figura abaixo), abre-se um segundo menu *popup* (imagem direita na figura abaixo), através do qual podemos configurar a quantidade de *bytes* por linha e a quantidade de *bytes* por coluna. Neste caso, mudamos *Column Size* de 4 para 2.



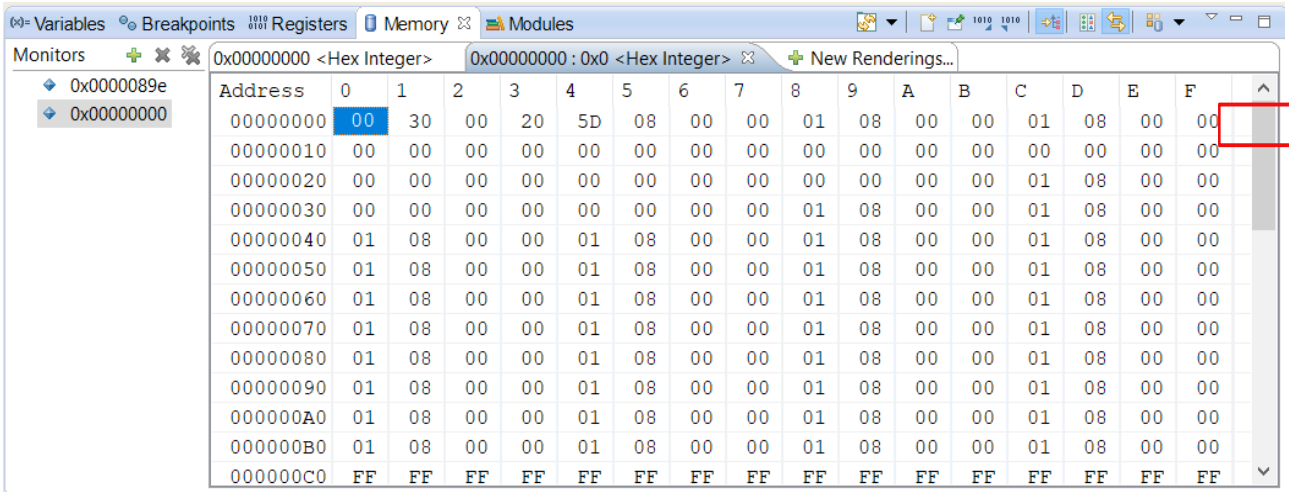
A nova renderização mostrada na seguinte figura certifica que a instrução está alocada nos endereços 0x0000089e e 0x0000089f.



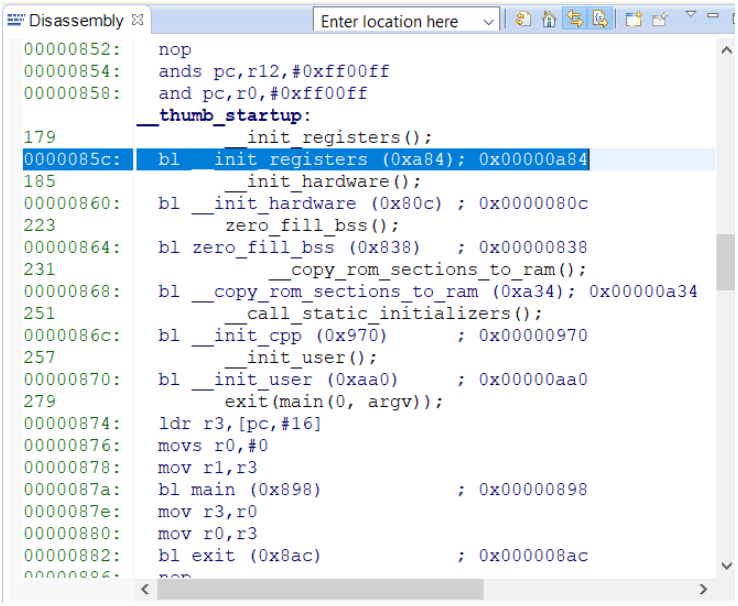
Mesmo assim, não é suficiente para inferirmos qual é a ordenação dos *bytes* na memória. O melhor formato para esta análise no *IDE CodeWarrior* é agrupamento em 1 *byte* como é feito na figura abaixo. Temos agora o valor 0x00 (*byte* menos significativo) no endereço 0x0000089e e o valor 0x23 no endereço maior. Daí podemos concluir que a ordenação dos *bytes* na memória é *little endian.*



A aba *Memory* permite visualizar e editar (se possível) o conteúdo das 3 unidades de memória, *Flash* (0x00000000-0x0001ffff), *RAM* (0x1ffff000-0x20002fff) e ROM (0xf0002000-0xf0002fff), sem que os endereços estejam associados a alguma variável (mostrada na aba *Variables*). Por exemplo, podemos ver o conteúdo de uma pilha cujo topo está armazenado no registrador `SP` e o conteúdo dos endereços 0x00000000 e 0x00000004 onde estão armazenados os valores que serão carregados em `SP` e `PC`, respectivamente. Levando em conta da ordenação *little endian*, o valor a ser carregado no `SP` é 0x20003000 e no `PC`, 0x000085D.

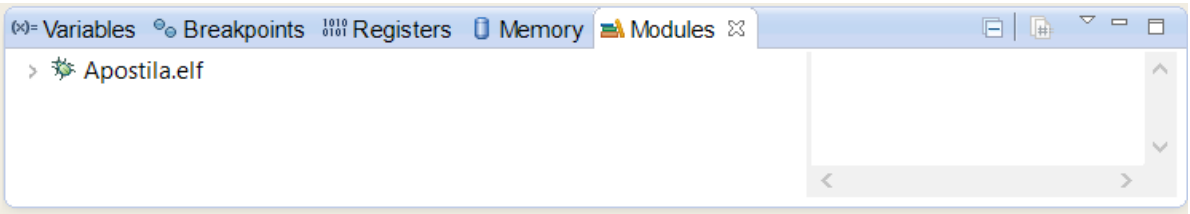


Um leitor mais atento questionaria se 0x0000085D é um endereço válido para PC. Sendo um processador de 32 *bits*, todos os endereços devem ser múltiplos de 4, ou seja, o último dígito hexadecimal dos endereços deve ser 0x0, 0x4, 0x8 ou 0xC. A explicação para esse valor aparentemente errado é que foi aproveitado o *bit* menos significativo para indicar um dos dois modos de instruções, modo ARM de 32 *bits* (*bit*=0) ou *Thumb* de 16 *bits* (*bit*=1). Nesse caso, o valor 1 indica que o processador vai operar no modo *Thumb* e o endereço efetivo a ser carregado no PC é 0x0000085C cuja primeira instrução é mostrada na figura a seguir. Note que as instruções que se seguem até a primeira instrução da rotina `main` no endereço 0x00000898 são as de inicialização típica recomendada pelo fabricante (Seção 1.1.4, página 12, em [1]).



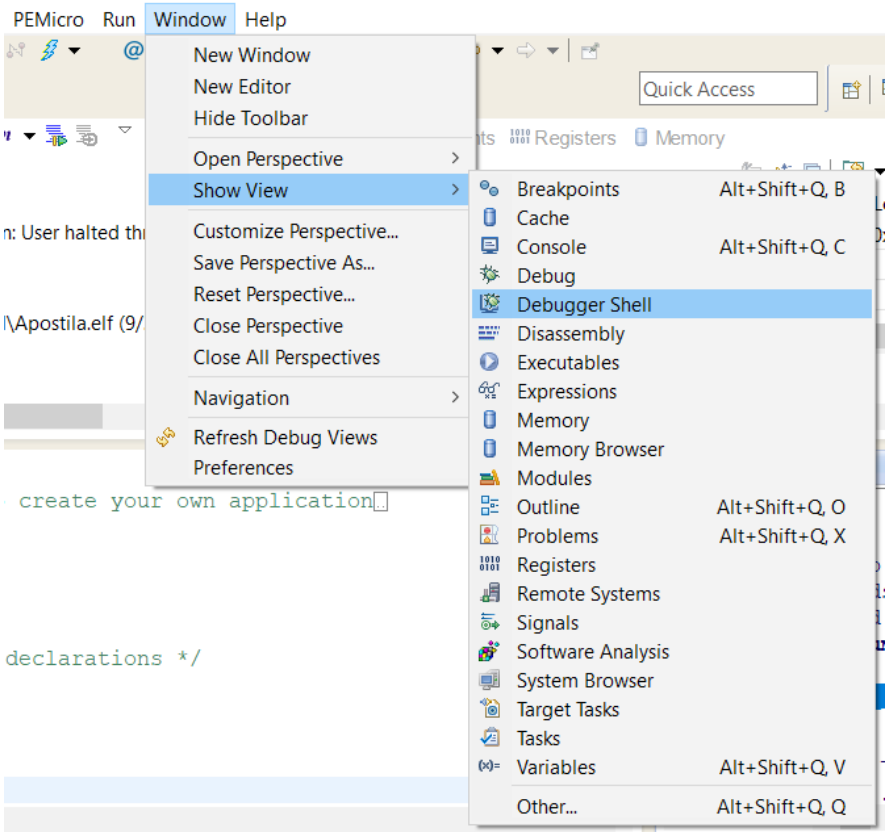
2.5.7 Aba *Modules*

Essa aba contém o nome do *firmware* em depuração. No caso, é o código executável gerado pelo projeto Apostila, `Apostila.elf`.

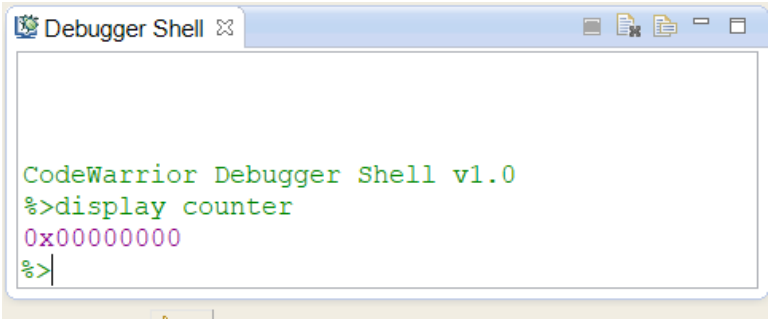


2.5.8 Aba *Debugger Shell*

Para aqueles que preferem um depurador em linha de comando, pode usar a aba *Debugger Shell*. Para que esta janela apareça na perspectiva *Debug*, vá ao menu superior e clique em "*Window* > *Show View* > *Debugger Shell*".



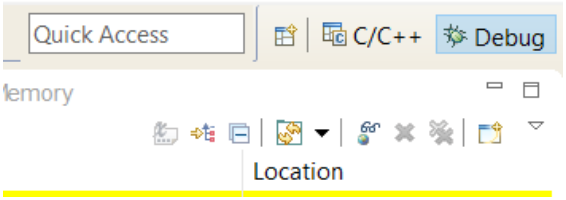
No canto inferior direito, a aba *Debugger Shell* mostrada na seguinte figura aparecerá.



Digitando *"help*", pode-se ver a lista de comandos possíveis. Digitando "*help <comando>*", pode-se consultar o formato de cada comando.

2.6 Chaveamento entre Perspectivas

É muito simples chavear entre as perspectivas quando abertas pelo comando "*Window > Open Perspective*". Todas as perspectivas abertas são visíveis no canto superior direito da interface do IDE. Quando se quer requer passar para uma outra perspectiva aberta, basta clicar no seu ícone no canto superior direito.



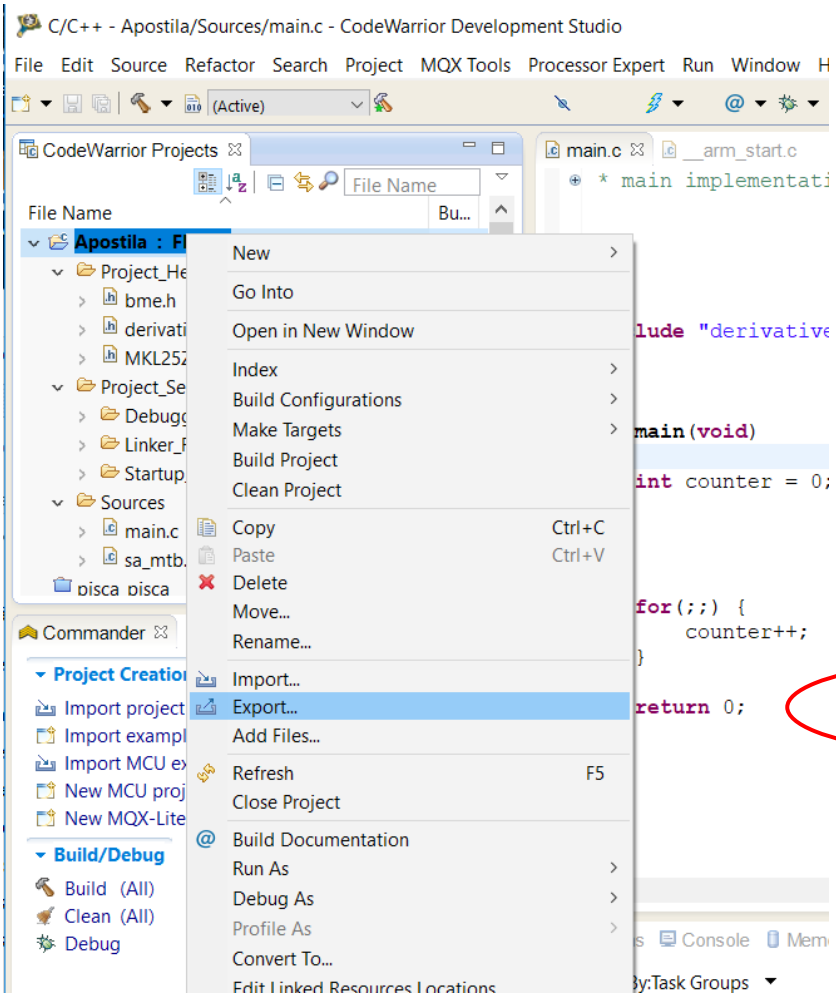
Pela sequência de comandos *"Window > Custom Perspective ..."* é possível individualizar as vistas que constam numa perspectiva. As abas/vistas/janelas/ferramentas são, porém, dinamicamente configuráveis. Podemos remover ou adicionar uma nova vista como mostra a inclusão de uma aba *"Debugger Shell"* na Seção 2.5.8. O *layout* dessas vistas numa perspectiva podem ser dinamicamente modificado, como ilustramos na Seção 2.2.1. Além disso, nesse IDE podemos sempre retornar à perspectiva-padrão seguindo o caminho *"Window > Reset Perspective ..."*.

2.7 Exportação e Importação de um Projeto

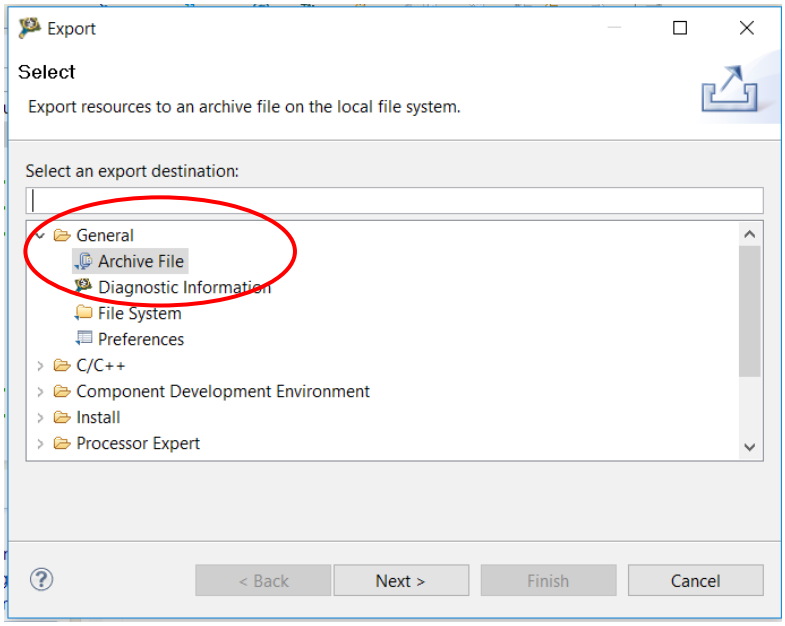
Segue-se um procedimento direto para exporta e importar um projeto no IDE CodeWarrior:

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*". Limpe todos os projetos abertos com *"Clean"* (Seção 2.2.2).

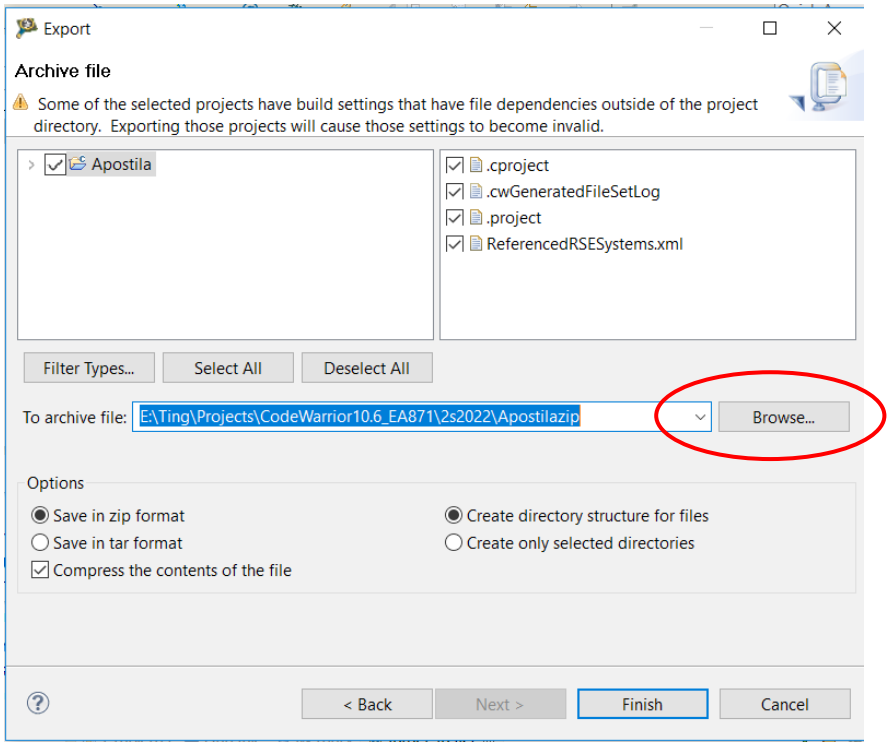
2. Clique novamente na pasta do projeto e selecione "*Export...*". Alternativamente, pode-se usar o menu "*File*", opção "*Export...*".



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



4. Na nova janela, mantenha as opções padrão, exportando todos os arquivos incluindo os dados de configuração, e clique no botão "Browse…" para selecionar a pasta que guardará o arquivo de exportação e entrar o nome do arquivo. Clique em "*Finish*".

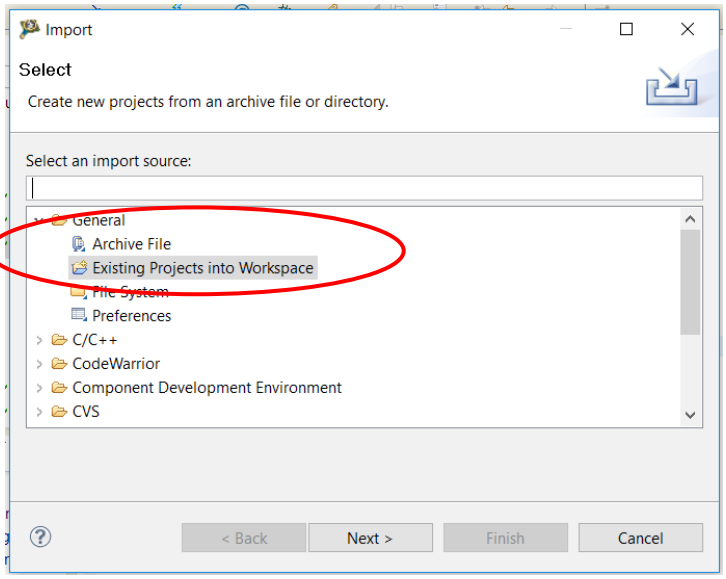


5. Um arquivo ZIP com o nome escolhido será gravado no local selecionado.

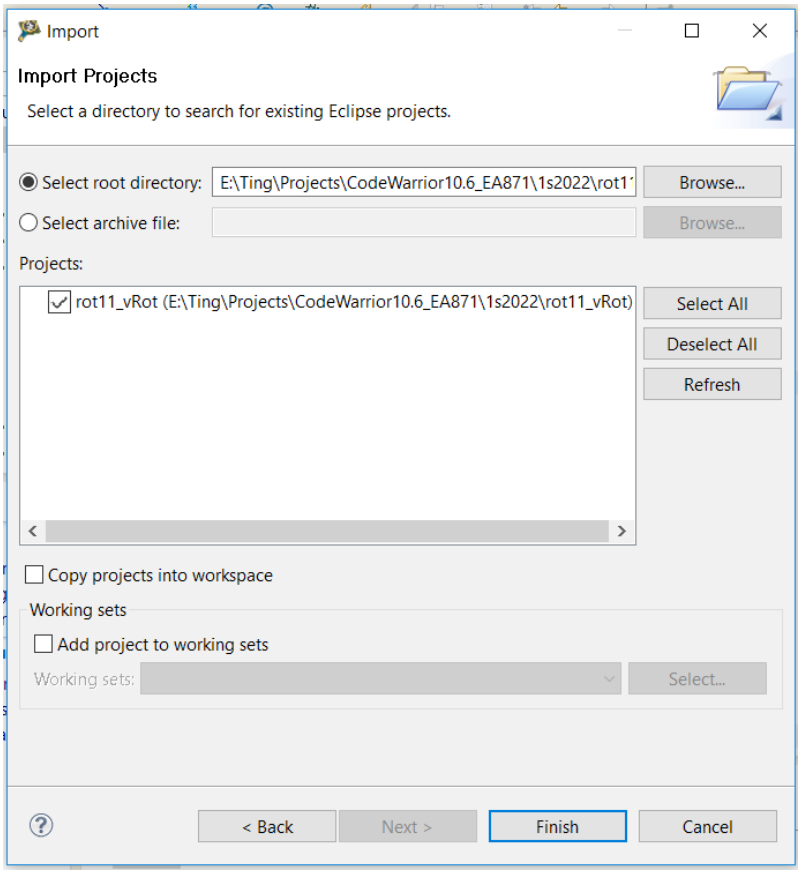
Para fazer a importação de um projeto, remova integralmente o projeto do *workspace* (Seção 2.1.6) e descompacte a pasta do projeto se ela estiver compactada. Se tentar importar um projeto com o

mesmo nome de outro projeto no mesmo *workspace*, aparecerá uma mensagem de erro. Há várias alternativas para importar uma pasta de projeto. A mais simples é arrastá-la para a hierarquia de projetos na janela *CodeWarrior Projects*. Outra alternativa é usar as ferramentas disponíveis no IDE:

6. Mova o projeto descompactado para a pasta do seu *workspace*. Clique com o botão direito na janela *CodeWarrior Projects* e selecione a opção *"Import..."* no menu mostrado em (2), ou selecione "*Import Project*" na janela "*Commander*", ou ainda use o menu "*File > Import...*". Aparecerá uma janela de configuração do tipo de importação. Como já há a pasta de projeto descompactado, selecione a opção mostrada na figura abaixo e clique em *"Next"*.



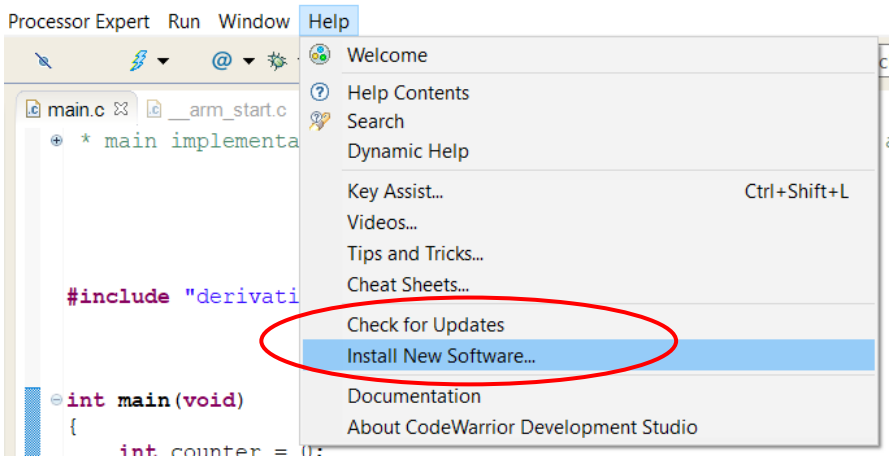
7. Na nova janela que aparece, ative a seleção da pasta de projeto clicando no botão "*Browse ...*". Localize e selecione a pasta descompactada do projeto. Se a pasta não estiver no seu workspace e quiser fazer uma cópia integral do projeto para o seu workspace, marque *"Copy projects into workspace"*. Confirme os dados do projeto a ser importado e clique em "*Finish*".



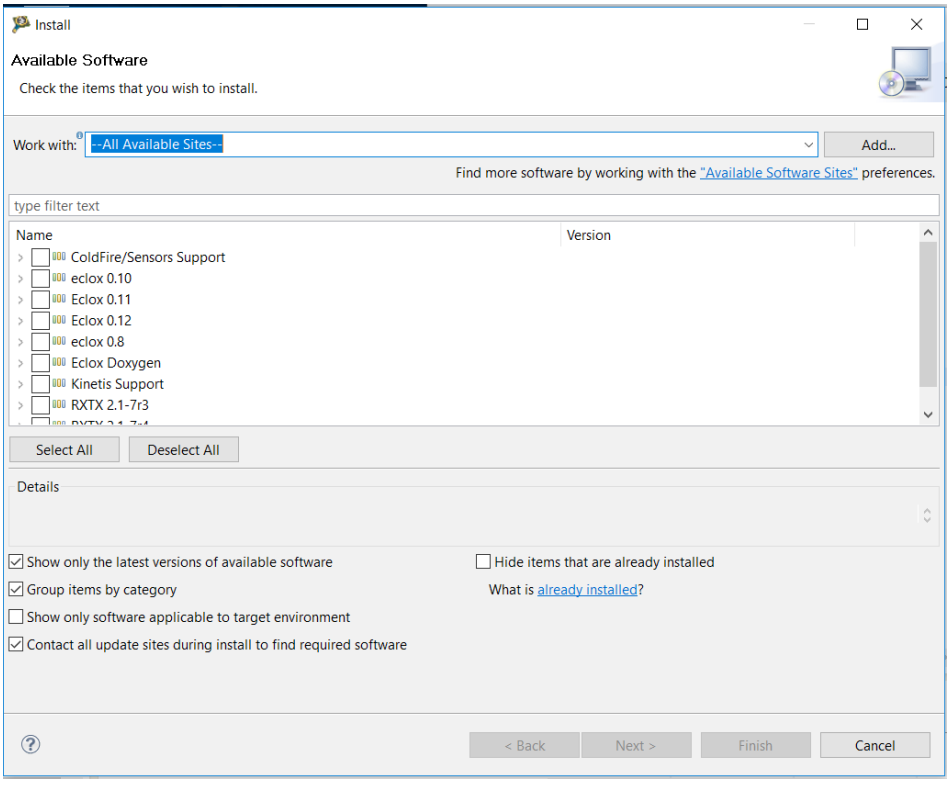
8. Pronto, seu projeto aparece agora no *workspace*.

2.8 Inclusão de um *Plugin*

Para se verificar atualizações, *patches* e outros *plugins*, usa-se a opção do menu principal "*Help* > *Install New Software...*", como o seu computador conectado à *Internet*.



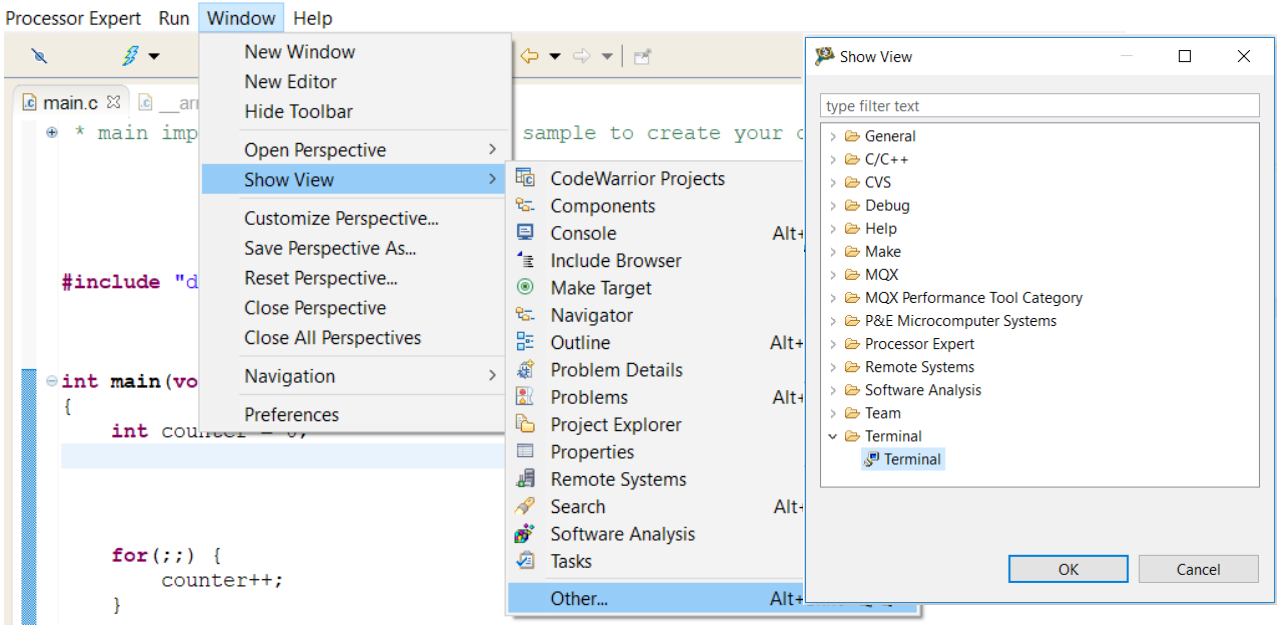
Na janela que se abre, podemos selecionar em "Work With" a opção "*All Available Sites*". O botão "*Add*" permite que se adicione pastas de arquivos ou *websites* que contenham outros *plugins*. No CW10, o padrão é a página de atualizações da *Freescale*. Pode-se marcar as opções desejadas. É interessante consultar se o item que se pretende instalar já está instalado clicando em *"already installed"*. Após a seleção das opções, clique em "*Next*" e segue um procedimento similar à instalação de um programa, que pode incluir leitura e aceitação de termos, bem como confirmação de confiança no *site* de onde foi feito o *download*. Caso o aplicativo desejado não esteja na lista, o procedimento varia entre os aplicativos conforme veremos nas subseções.



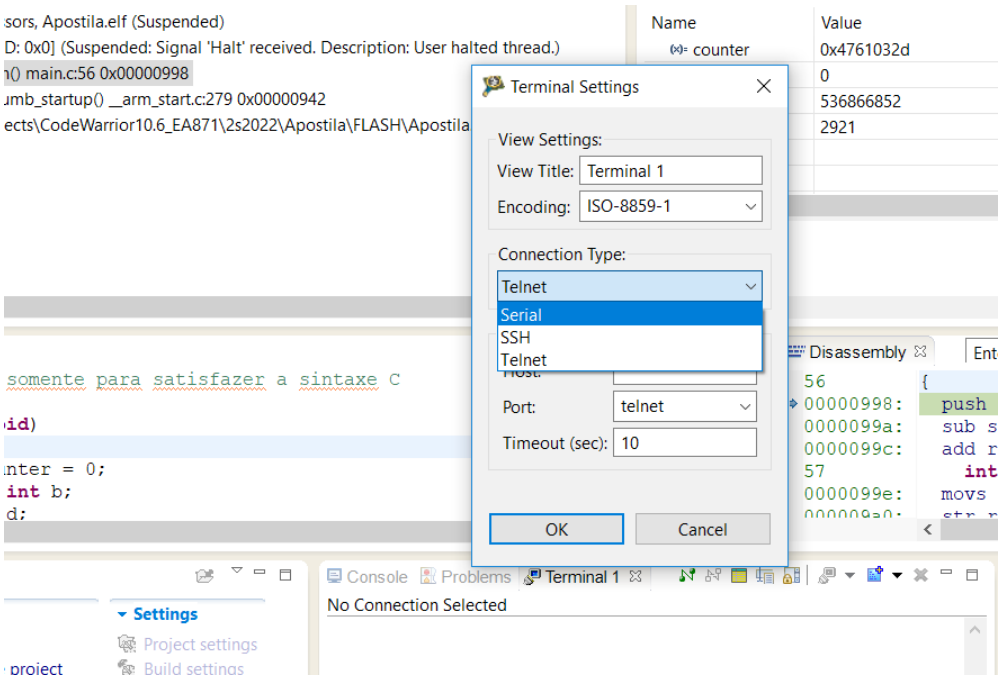
2.8.1 Terminal Serial

O aplicativo *OpenSDA* permite que um programa residente no microcontrolador se comunique com um computador via uma porta serial. E o aplicativo Terminal é um *plugin* que provê uma interface de comunicação direta, via porta serial, entre o computador onde está instalado o *CodeWarrior* e o microcontrolador onde é executado o projeto. Ele é, portanto, muito utilizado em vários experimentos. Vamos ilustrar os passos de instalação de um *plugin* com a instalação do Terminal nas versões de *CodeWarrior* maior ou igual a 10.6.

Insira no campo "Work With" na janela anterior "http://rxtx.qbang.org/eclipse", selecione o item *"RXTX 2.1-7r4"* e segue o fluxo de instalação, aceitando todas as recomendações e termos de compromisso. Finalizada a instalação e reaberto o IDE, certifique se o terminal foi instalado corretamente, percorrendo o caminho "*Window* > *Show View* > *Other ...*". O item "*Terminal*" deve aparecer no menu que se abre.



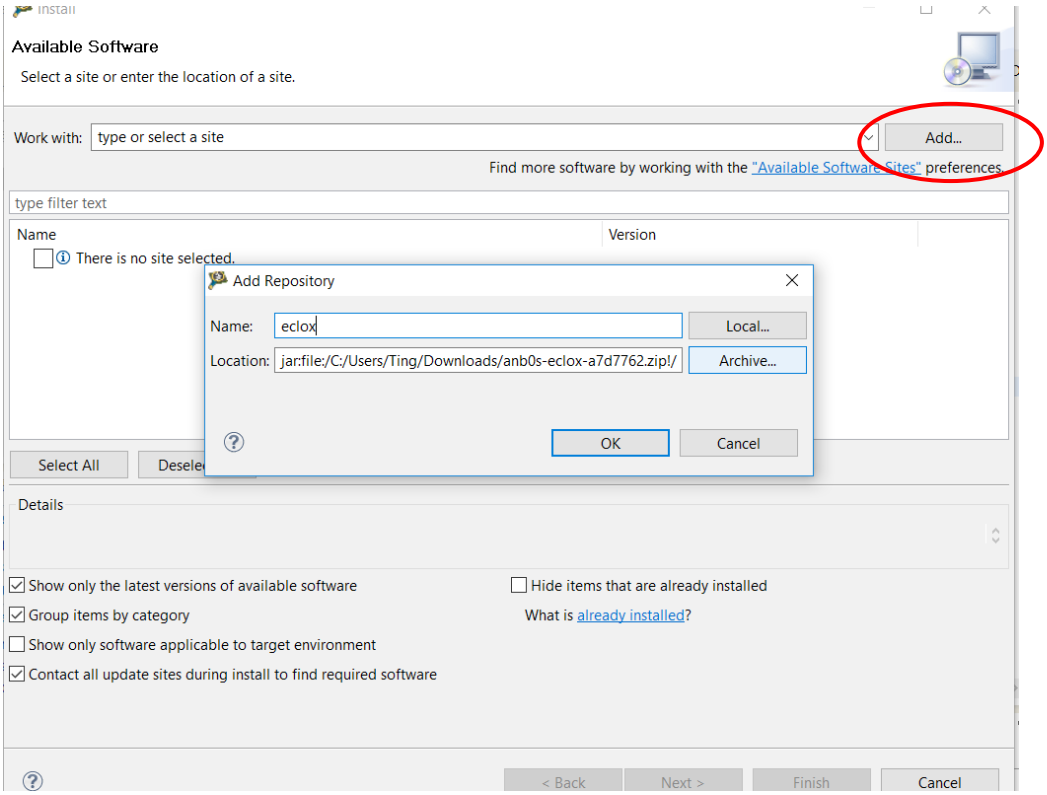
Ao clicar no *Terminal*, vai abrir um aba *Terminal*. Ative a janela *Settings* na barra de ferramentas do *Terminal* clicando no ícone destacado com a linha vermelha na figura abaixo e veja se a opção *Serial* consta na lista de "*Connection Type*".



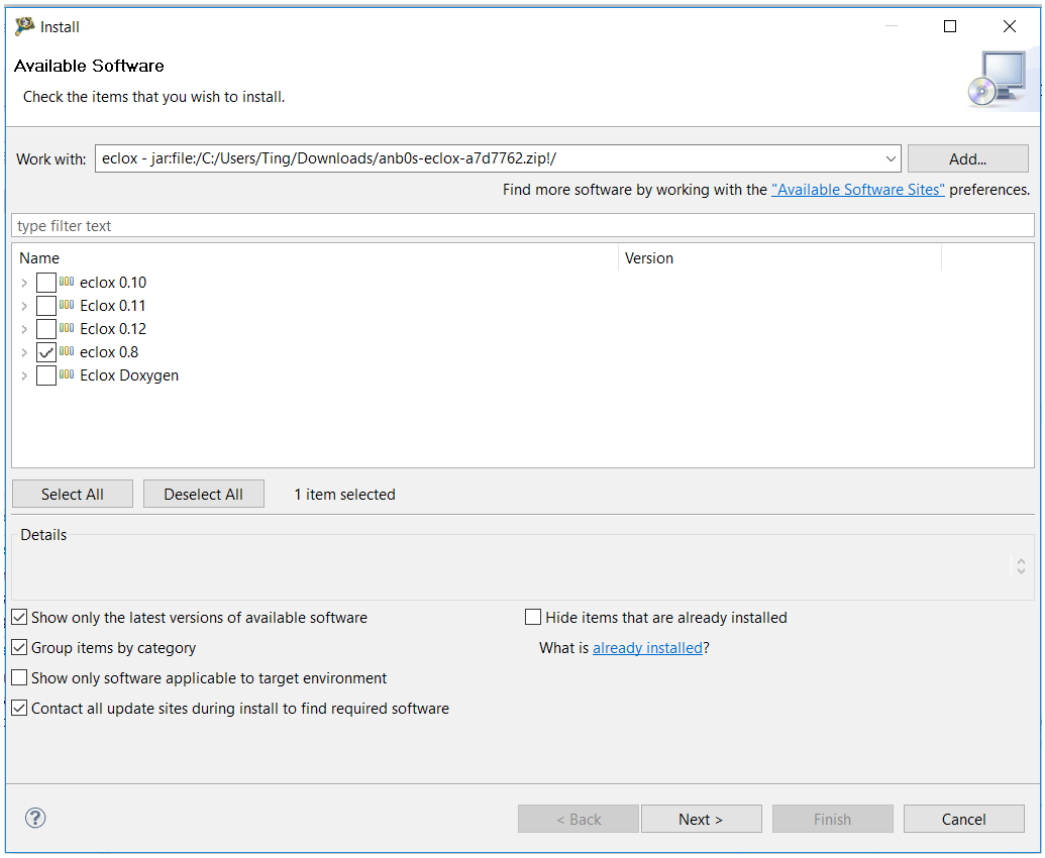
2.8.2 Doxygen

Doxygen+Graphviz+Mscgen são ferramentas de suporte à geração da documentação dos seus códigos diretamente a partir dos comentários que você inseriu no seu programa, como mostra Erich Styger na referência [9]. Eclox é um *plugin* que facilita a integração dessas ferramentas de documentação ao Eclipse, e consequentemente, ao *CodeWarrior* implementado sobre o Eclipse.

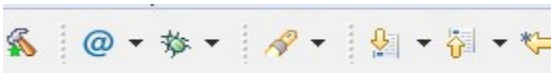
Para instalar o sistema de documentação, baixe os arquivos compactados indicados em [8], [14] e [17]. Instale os aplicativos graphviz e mscgen usando os instaladores [14] e [17], respectivamente. Para instalar eclox, abra a janela de instalação e clique em *"Add ..."* para adicionar novos locais/arquivos dos aplicativos de terceiros. Aparecerá uma nova janela com as duas alternativas. No nosso caso, já temos o instalador baixado, só é necessário inseri-lo como *"Archive"* e indicar o caminho onde se encontra o instalador. Clique em *"OK"*.



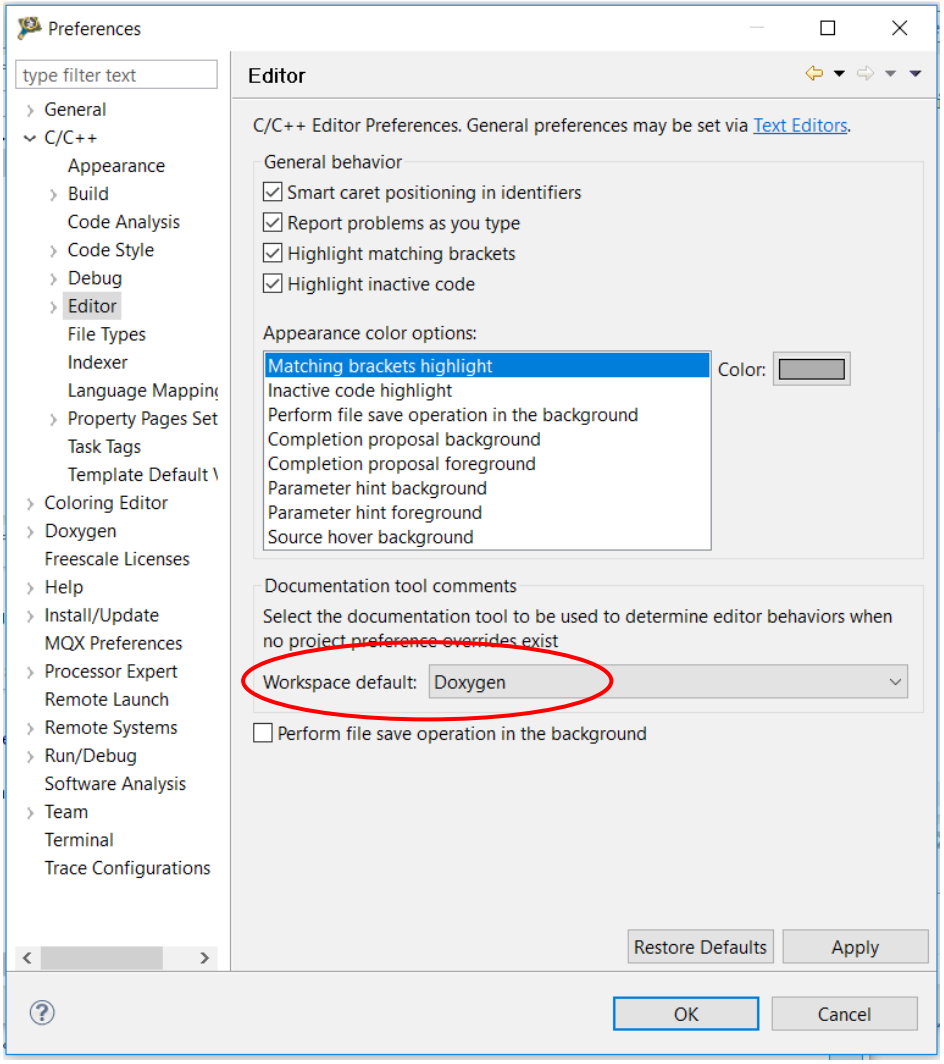
Aparecerá uma nova janela mostrando as diferentes versões de eclox disponíveis. Para o eclipse sobre o qual está implementado o IDE, a versão compatível é eclox 0.8.



Quando o Doxygen estiver corretamente integrado, automaticamente aparece na barra de ferramenta do *CodeWarrior* o botão com ícone @.



Para certificar se o Doxygen está configurado como a ferramenta de documentação do editor C/C++, linguagem que utilizamos no desenvolvimento dos nossos códigos, abra a janela **Preferences** com os passos *Window* > *Preferences* e veja se "*Doxygen*" está selecionado como "*Workspace default*".



Um teste rápido para verificar se o ambiente está corretamente configurado: digite por exemplo no seu código-fonte `/*!` e o editor deve complementar automaticamente o escopo de comentários como se segue:

```
/*!
 *
 * @return
 */
```

2.9 Documentação dos Códigos

Documentar o seu código não só é uma forma de assegurar a propriedade intelectual como também facilita a sua manutenção ou atualização pelos terceiros. Se incluirmos nos nossos códigos-fonte

blocos de comentários com delimitadores específicos e comandos especiais listados em [10], Doxygen organiza automaticamente os comentários, conforme a semântica do comando precedido aos textos de comentários, e gera uma documentação a nível profissional. Ele é um gerador de documentação de programas bastante flexível. Aceita códigos-fonte em diferentes linguagens de alto nível (C/C++/C#/*Objective*-C/PHP/Java) e gera documentos em distintos formatos populares como html, tex, rtf, man etc. Porisso e pela semelhança da sintaxe dos seus blocos de comentários com a da linguagem C, é a ferramenta de documentação preferida pela comunidade de desenvolvedores de *software* livre [18].

Por exemplo, ao adicionar o seguinte bloco de comentários num código

```
/*!
  * \brief Led piscante
  * \return 1 somente para satisfazer a sintaxe C
  */
```

Doxygen gera os seguintes textos na documentação:

Led piscante. Returns

1 somente para satisfazer a sintaxe C

Os delimitadores `/*!` `*/` indicam ao Doxygen quais blocos de comentário que ele precisa processar. E os comandos \brief e \return definem ações específicas do Doxygen. Neste caso, os dois comandos geram, respectivamente, uma breve descrição da função documentada e o valor que ela retorna com os textos inseridos no bloco de comentários. Vale ressaltar que Doxygen suporta também o prefixo @, ou seja, `\brief` ↔ `@brief` e `\return` ↔ `@return`.

Doxygen distingue três tipos de descrições para um elemento do código: breve (precedido pelo comando `\brief`), detalhada (o que segue de uma descrição breve ou precedido pelo comando `\details`) e embutido (em funções/rotinas). Em relação a uma função/rotina, os blocos de comentários das duas primeiras descrições ficam fora do escopo de definição da função. Para a terceira descrição, o suporte de Doxygen é precário. Ela requer que o desenvolvedor use os comandos disponíveis em Doxygen para construir de forma quase artesanal um texto no formato desejado. Nesta disciplina, só documentaremos a interface, ou protótipo, das nossas funções usando descrições breves. Limitaremos ainda à documentação básica dos membros de um arquivo, como as variáveis globais e estáticas e os tipos de dados derivados de `struct`, `union` e `enum`. Para mais informações, remetemos os leitores à referência [11].

2.9.1 Documentação de Arquivos

Mesmo sem blocos de comentários no estilo esperado pelo Doxygen, ele consegue varrer todos os arquivos das pastas especificadas e tentar extrair desses arquivos membros e funções para uma documentação mínima na forma como ele acha que deve ser. Para termos um melhor controle na documentação gerada, recomenda-se explicitar os arquivos que contém de fato os blocos de comentários com a notação de Doxygen. Tais arquivos devem conter nas primeiras linhas um bloco de comentários Doxygen contendo o comando especial `\file <nome do arquivo>`. Além

do comando \file, recomenda-se incluir no mínimo os seguintes comandos: \brief (uma breve descrição das funções codificadas no arquivo), \author ou \authors (autores do arquivo) e \date (data de implementação) e \version (versão do arquivo) se houver mais de uma versão. Para a função main.c do projeto Apostila foi inserido o seguinte bloco no início do arquivo para identificar o arquivo como documentado com a notação Doxygen:

```
/*!
 * \file main.c
 * \brief Projeto-modelo em C gerado automaticamente
 * \author Wu Shin-Ting
 * \date 31/08/2022
 */
```

2.9.2 Documentação da Interface das Funções

Um bloco de comentários que permite gerar a documentação de cada função implementada num arquivo deve conter no mínimo o comando especial \brief seguido de uma breve descrição da ação da função. O comando especial \fn que especifica o protótipo da função, como mostra no bloco de comentários da função main, é opcional se o bloco de funções estiver imediatamente antes ou depois da declaração/definição da função:

```
/*!
 * \fn int main (void)
 * \brief Esta documenta&ccedil;&atilde;o &eacute; um teste
 * \return somente para satisfazer a sintaxe C
 */
```

Observe que há ainda um comando especial \return. Esse comando é usado para documentar o valor retornado pela função caso o tipo da função não seja void. Neste caso, a função main é declarada como do tipo int e ela retorna incondicionalmente o valor 0 quando o fluxo de controle chega na instrução return 0.

Se na chamada de uma função passa-se uma lista de argumentos, pode-se incluir na documentação a descrição de cada um dos argumentos usando o comando especial \param com os atributos opcionais [in], [out] ou [in.out]. Esses atributos indicam, respectivamente, o sentido de entrada (para a função), saída (da função) e entrada e saída do fluxo dos dados dos argumentos. O seguinte bloco de comentários para a função divide, inserida no arquivo main.c do projeto Apostila, ilustra o uso desse comando especial na documentação dos 4 argumentos da função:

```
/*!
 * \brief Computa o quociente e o resto da divis&atilde; de dois n&uacute;meros
inteiros
 * \param[in] a dividendo
 * \param[in] b divisor
 * \param[out] c quociente
 * \param[out] d resto
 * \return quociente
 */
int divide (int a, int b, int *c, int *d) {
    *c = a / b;
    *d = a%b;
    return *c;
}
```

Segue-se a documentação da função gerada. Mesmo sem o comando explícito de \fn, gerou-se o protótipo da função e a descrição de cada argumento. Além disso, vale ressaltar a codificação dos

caracteres acentuados por sequências de escape que suportam ISO-8859-1 no código-fonte (ã = `&atilde;`, ú = `&uacute;` etc) para gerar corretamente os caracteres acentuados no nosso navegador segundo a recomendação em [19].

Function Documentation

◆ divide()

```
int divide ( int  a,
             int  b,
             int * c,
             int * d
           )
```

Computa o quociente e o resto da divisão de dois números inteiros.

Parameters

[in] **a** dividendo
[in] **b** divisor
[out] **c** quociente
[out] **d** resto

Returns

quociente

2.9.3 Documentação dos Membros de um Arquivo

São considerados como membros de um arquivo C, os tipos de dados derivados de `struct`, `union`, ou `enum` ou variáveis globais. Usamos o comando especial `\var` para descrever uma variáve global, `\struct`, `\union` e `\enum` para um novo tipo de dados derivado, `\typedef` para redefinição de um novo tipo de dados e `\def` para definição de uma macro.

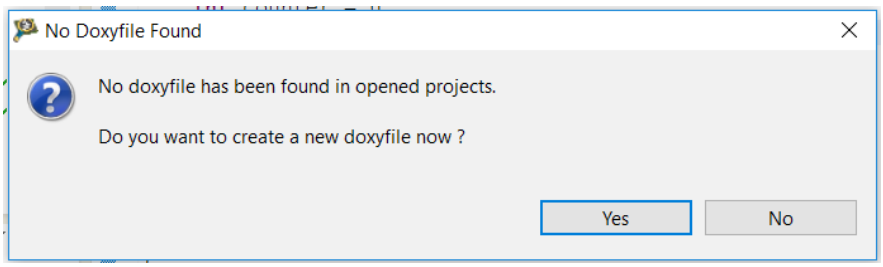
```
/*!
 * \def GPIO_PIN(x)
 * \brief Construir uma m&aacute;scara com 1 no bit x.
 */
#define GPIO_PIN(x)  ((1)<<(x))

/*!
 * \enum boolean_tag
 * \brief Deriva&ccedil;&atilde;o de um tipo enum para valores booleanos
 */
enum boolean_tag {
      OFF,     /*!< falso/apaga/desativa/liga */
      ON      /*!< verdadeiro/acende/ativa/fecha */
};

/*!
 * \typedef booleano_type
 * \brief redefini&ccedil;&atilde;o do nome do tipo de dado derivado enum
boolean_tag.
 */
typedef enum boolean_tag booleano_type;

/*!
 * \var int a
 * \brief Cont&eacute;m o valor do dividendo
 */
```

```
int a;
```

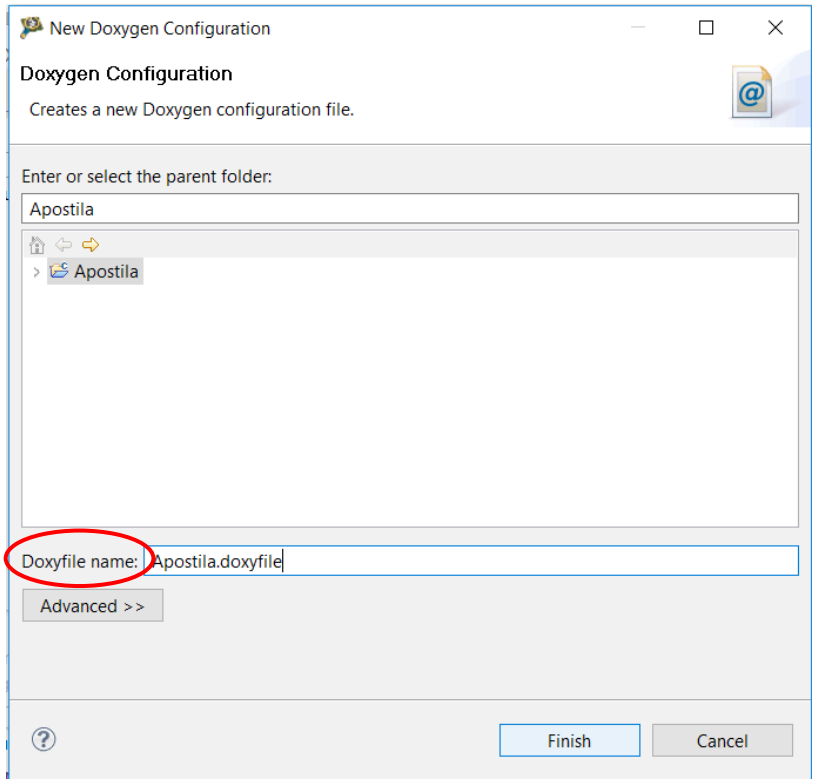
Ao inserirmos esses trechos de códigos de definição dos membros em main.c, foi possível gerar a seguinte documentação sobre a macro, o novo tipo de dado, o tipo de dado derivado e a variável global. Vale comentar que se pode omitir a primeira linha em todos os blocos de comentários acima, de forma análoga à omissão de \fn no bloco de comentários da função divide.

2.9.4 Geração de Documentação

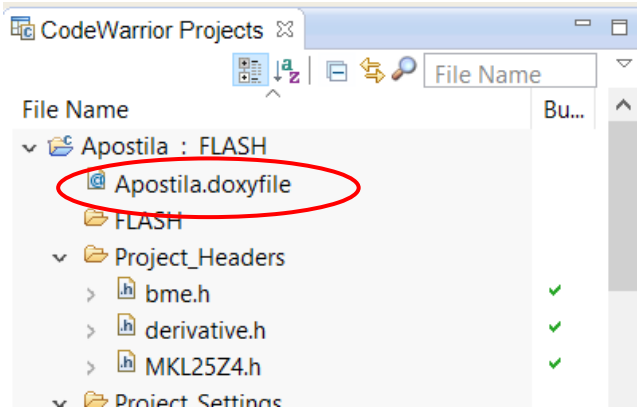
Doxygen gera a documentação de um código com base nos parâmetros especificados num arquivo-script de configuração. Se não existir este arquivo no projeto aberto, o Eclipse automaticamente ajuda o usuário a criar um quando se clica no botão "@" na barra de ferramenta



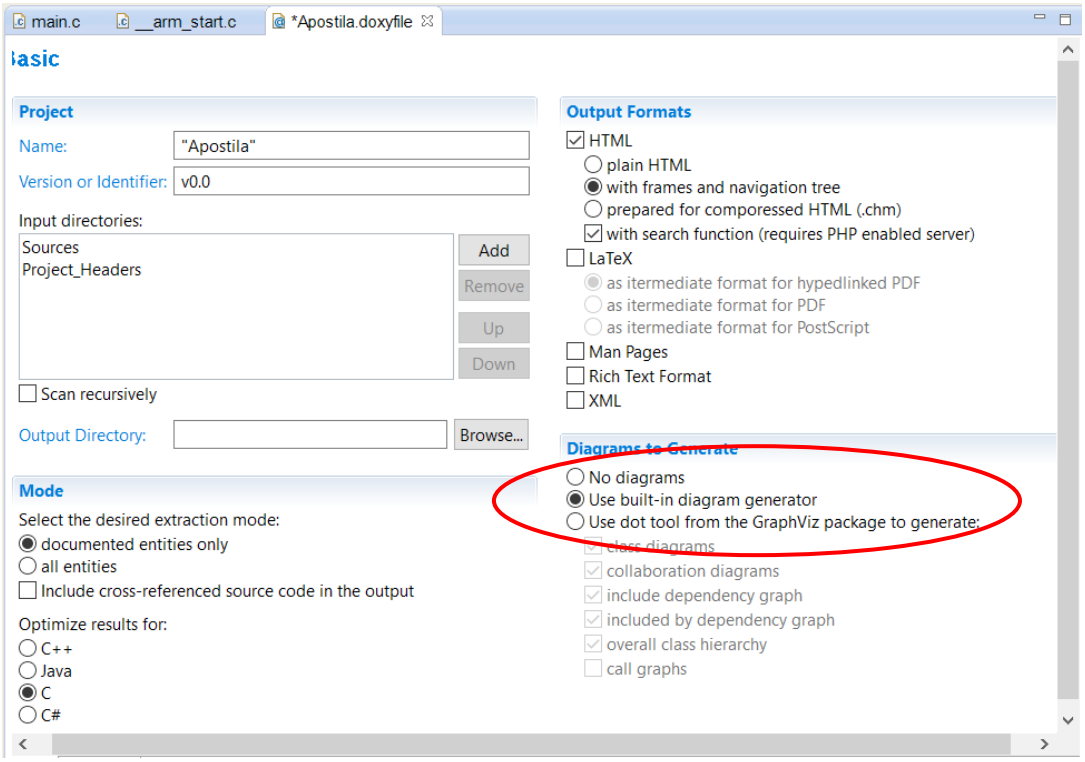
Se clicar em "Yes", aparece um menu *pop-up* solicitando o nome do arquivo de configuração que pode ser um novo ou um já existente. Na imagem abaixo, adicionou-se no campo *"Doxyfile name"* o nome de um novo script `Apostila.doxyfile` para o projeto ativado.



O novo arquivo criado aparece na árvore da janela *CodeWarrior Projects*

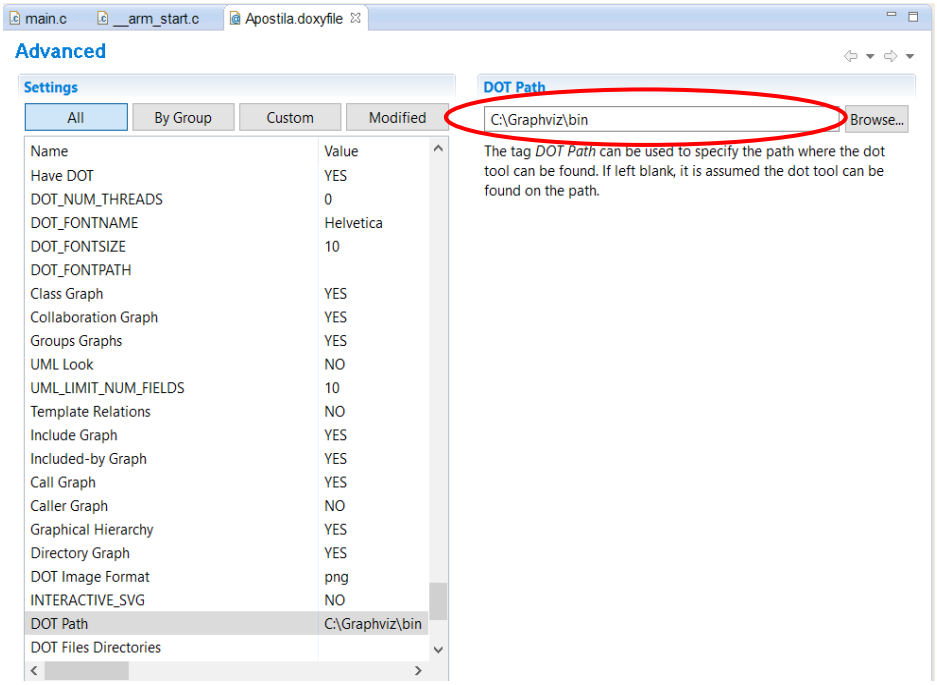


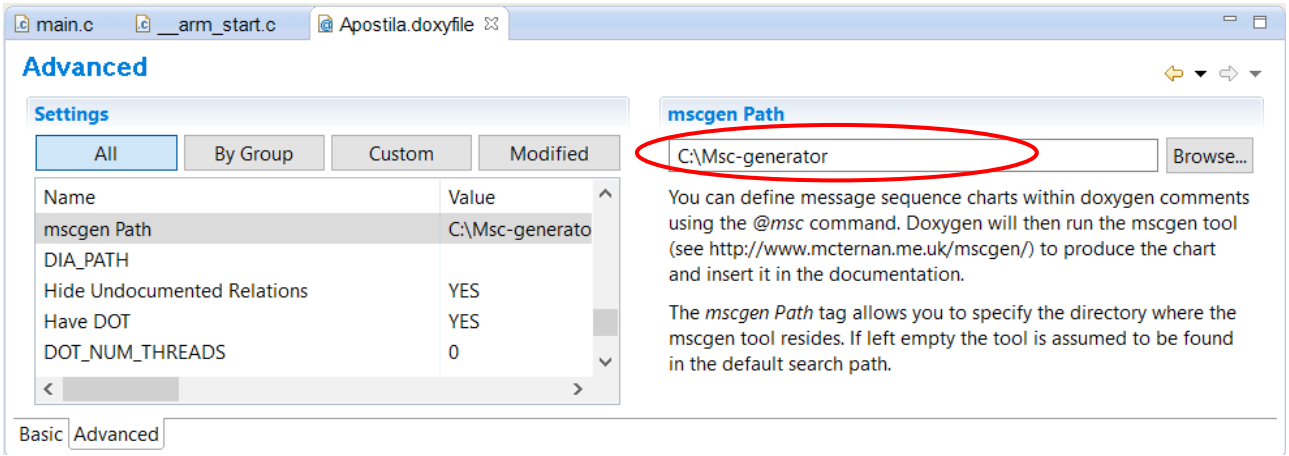
Ao clicar no nome do arquivo "*Apostila.doxyfile*", Eclox mostra uma janela de editor permitindo configuração personalizada deste arquivo. Através do painel "*Basic*", pode-se especificar o nome (campo "*Name*") e a versão (campo "*Version or Identifier*") do projeto, os diretórios que devem ser processados (área "*Input directories*") para gerar a documentação, processamento recursivo dos diretórios adicionados na área (caixa de seleção "*Scan recursively*") e o diretório de saída dos arquivos de documentação gerados (campo "*Output Directory*"), o modo de extração dos comentários adicionados nos arquivos processados (caixas de seleção "*Mode*"), a linguagem do código-fonte (caixas de seleção "*Optimize results for*"), o formato de saída da documentação (caixas de seleção em "*Output Formats*") e a forma de geração de diagramas de classes e de dependência entre elas (caixas de selação "*Diagrams to Generate*"). A janela abaixo mostra uma configuração para gerar uma documentação em html. Note que foi configurado que a geração dos diagramas seja com uso das próprias ferramentas de Doxygen. Como essas ferramentas apresentam poucos recursos, o próprio autor de Doxygen recomenda o uso de Graphviz e Mscgen. Caso esses estiverem instalados, marque *"Use the dot tool from the GraphViz to generate"*.



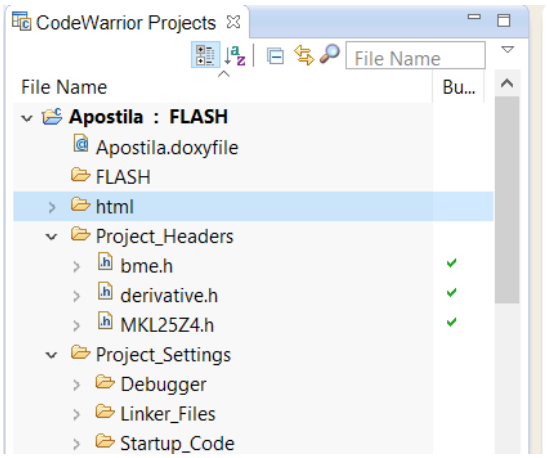
Observe que para os projetos desta disciplina é suficiente selecionar “C”, fazer extração dos arquivos com blocos de comentários de notação Doxygen, gerar uma documentação no formato html, e incluir três pastas na área *“Input Directories*”: *Project_Headers* e *Sources*. Se quisermos ver as dependências das funções implementadas com as funções disponíveis na pasta *Project_Settings/Startup_Code*, devemos incluir esta pasta. Como não foi especificado o diretório de saída dos arquivos de documentação, Doxygen usará o diretório corrente como o diretório de saída.

Há um segundo painel, uma versão "*Advanced*" do editor, através do qual devemos especificar os diretórios onde se encontram os aplicativos Graphviz e Mscgen através dos campos “*DOT Path*” e “*mscgen Path*”. O aplicativo Graphviz é utilizado pelo Doxygen para gerar os grafos de dependência entre as funções e o aplicativo Mscgen é usado para gerar diagramas de interação entre os eventos. Você pode ainda gerar um painel com a hierarquia dos diretórios da documentação ativando “*Generate Tree View*”. Nas imagens seguintes são mostrados os caminho dos aplicativos Graphviz e MscGen nas bancadas do LE-30.

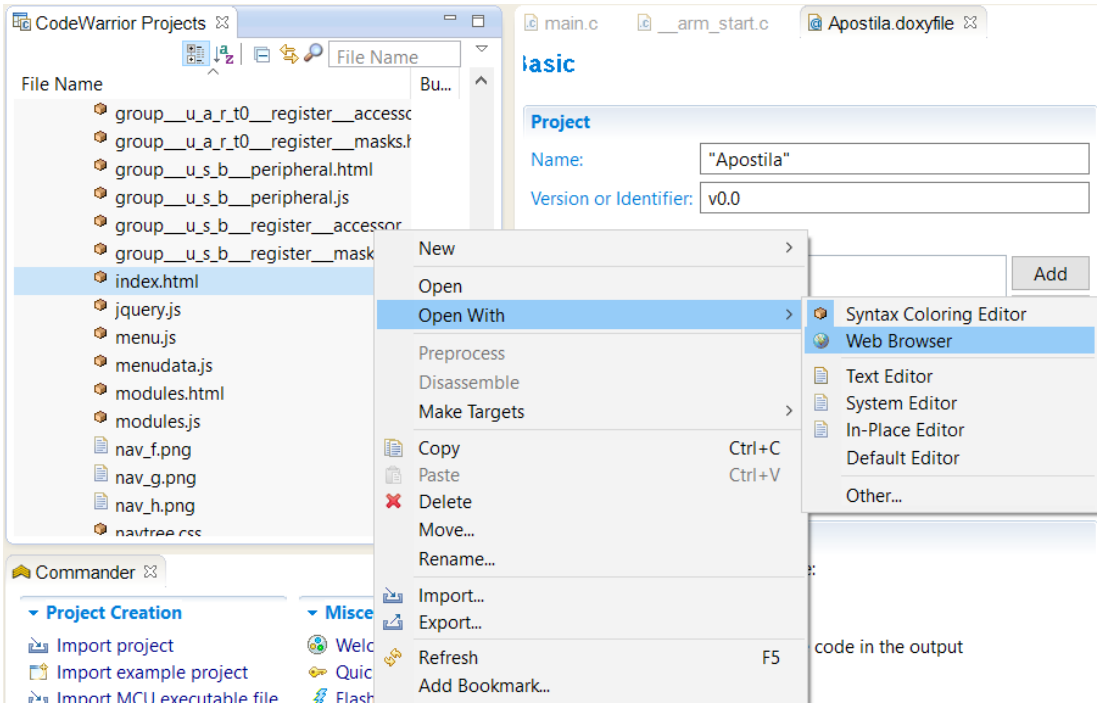




Para finalizar a configuração, basta salvar o *script* e, para gerar uma documentação, clicar novamente no botão "@". As documentações nos formatos selecionados serão geradas. Para a configuração que setamos, uma documentação em html será gerada e colocada numa pasta html.



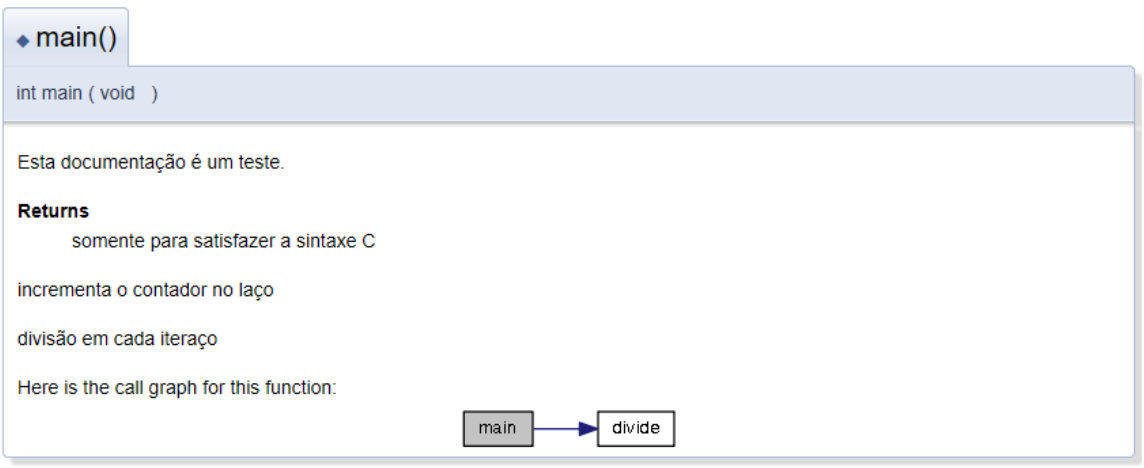
E para renderizar o conteúdo desta pasta (html) é necessário carregar o arquivo `index.html` dentro desta pasta num navegador.



Para ilustrar a geração de um grafo de dependência, vamos incluir uma chamada da função `divide` dentro de `main`:

```
int main(void)
{
	int counter = 0;
	static int b=6;
	int c, d;

	for(;;) {
		counter++;	/*! incrementa o contador no la&ccedil;o */

		divide (a,b,&c,&d); /*! divis&atilde;o em cada itera&ccedil;o */
	}

	return 0;
}
```

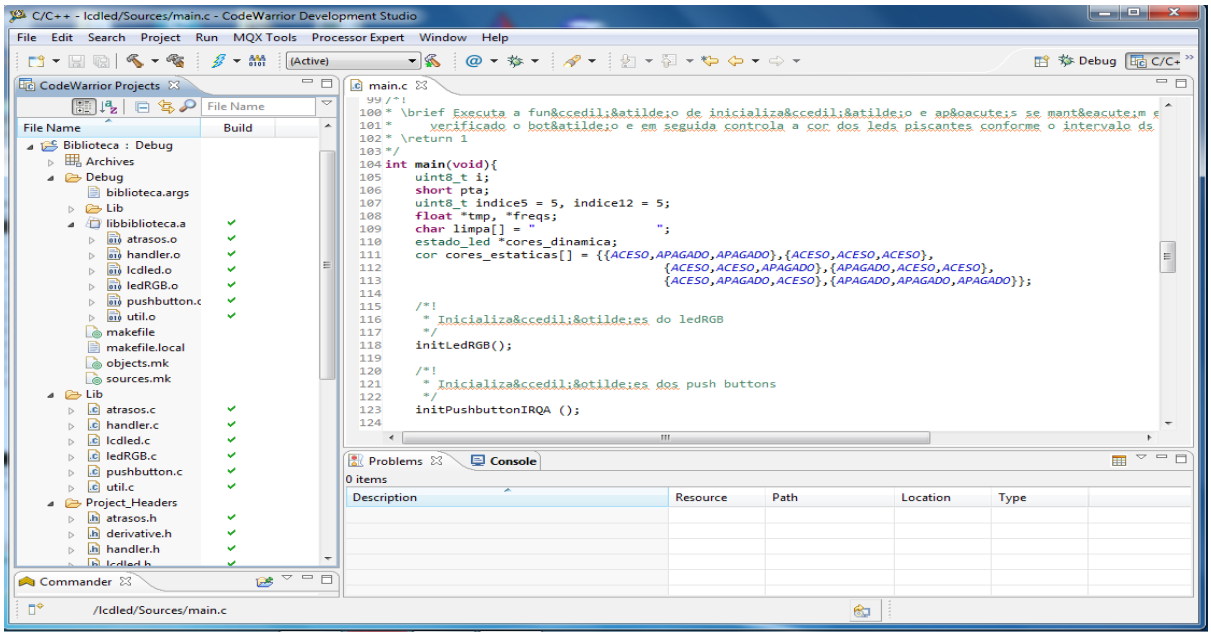
Segue-se a documentação gerada para `main`.



2.10 Criação de uma Biblioteca de Rotinas

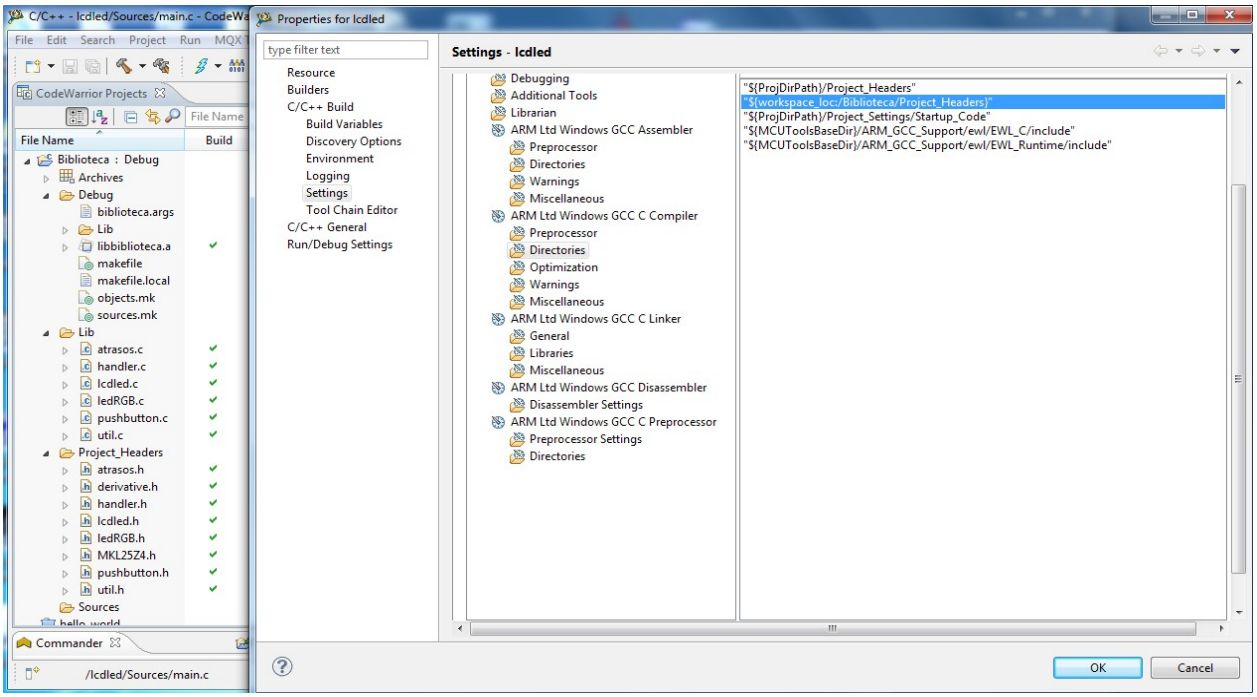
Para evitar duplicação de códigos e facilitar a sua manutenção, é recomendável pré-compilar rotinas de uso frequente nos diferente projetos, agrupá-las numa biblioteca e ligá-las com os programas específicos de um projeto somente na etapa de linkagem.

No ambiente IDE CodeWarrior a criação de uma biblioteca de arquivos pré-compilados é também um projeto. Seguem-se os mesmos passos iniciais mostrados na Seção 2.1. Somente, ao invés de configurar "*Application*" para *Project Type/Output* no passo 7, seleciona-se "*Library*" como o tipo de projeto desejado. No final do processo é gerado um novo projeto com as seguintes pastas: *Debug, Lib*, *Project Headers*, *Sources*. Os arquivos-fonte de rotinas (*.c) que devem fazer parte da biblioteca são colocados na pasta *Lib* e os respectivos arquivos-cabeçalho *.h são usualmente organizados na pasta *Project Headers*. Na figura abaixo ilustra o projeto de biblioteca de nome "Biblioteca" criado. Foram inseridas vários arquivos *.c e *.h nas pastas *Lib* e *Project Headers*, respectivamente. Com o comando "*Project* > *Bulid*" foram gerados os correspondentes arquivos-objeto *.o e organizados automaticamente no arquivo-biblioteca "libbiblioteca.a" na pasta *Debug*, prontos para uso.

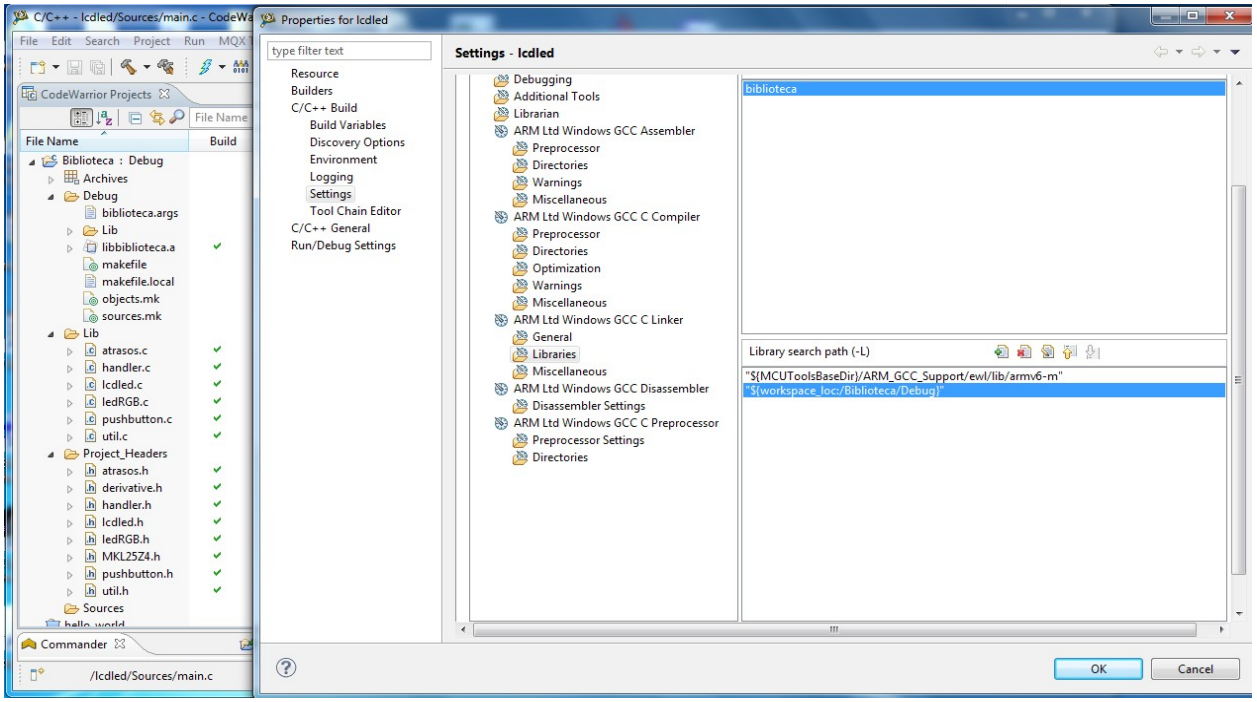


Como usar esta biblioteca num projeto-aplicação? Para isso é necessário que o projeto-aplicação possa resolver os protótipos das rotinas chamadas durante a fase de compilação e tenha acesso às instruções das rotinas invocadas durante a fase de ligação. Podemos configurar o ambiente de compilação e de ligação dos projeto-aplicação, incluindo os caminhos das pastas que contém os protótipos das rotinas e o arquivo-biblioteca em si através de "*Project* > *Properties* > *C/C++* > *Settings*", como detalhado em [13].

A figura que se segue ilustra a adição do caminho da pasta (*Directories*) que contém os protótipos das rotinas necessários para a compilação. Observe que neste caso consideramos que tanto o projeto-biblioteca *libbiblioteca.a* quanto o projeto-aplicação *lcdled* se encontram num mesmo espaço de trabalho (*workspace*).



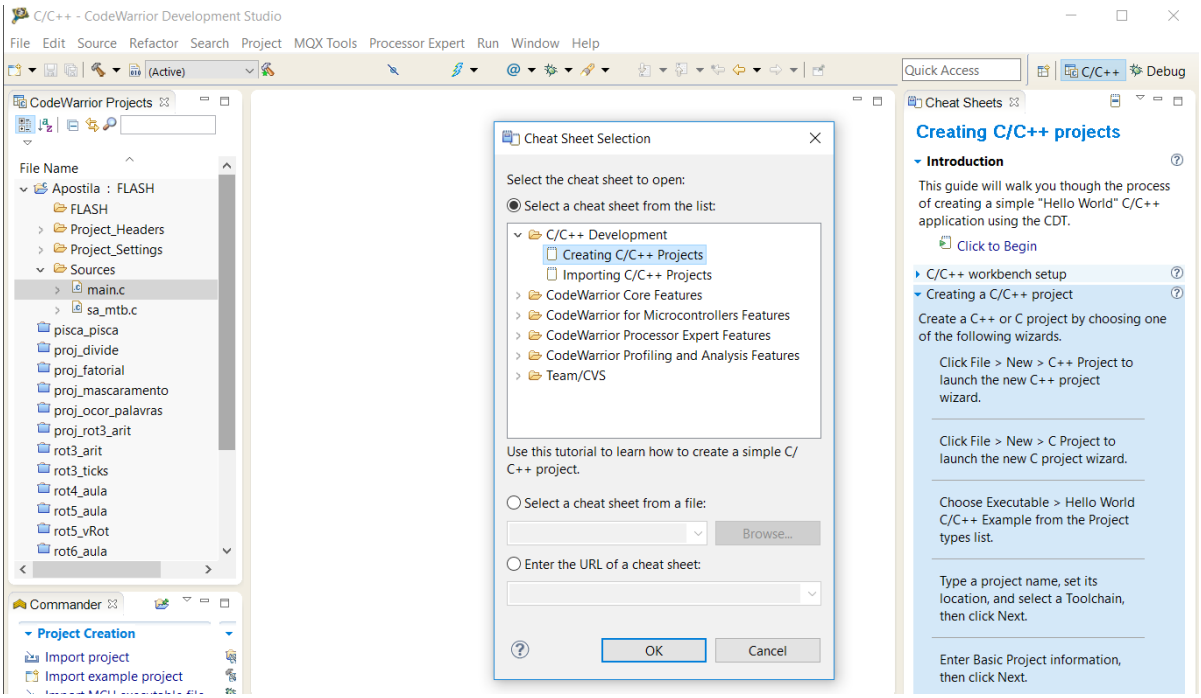
E a última figura mostra a adição do arquivo-biblioteca lib**biblioteca**.a a ser utilizado pelo ligador na construção do projeto *lcdled* e do caminho da pasta onde se encontra esta biblioteca.



2.11 Folhas de Dicas

O IDE 10.6/11.1 dispõe de uma coletânea de folhas de dicas (*cheat sheet*) acessível pelo caminho *Help* > *Cheat Sheet*. Essas folhas de dicas são muito úteis para consultas rápidas dos detalhes de

um procedimento específico. A seguinte figura ilustra a folha de dicas de "*Creating C/C++ Projects*" na subpasta *C/C++ Development*.



Referências

[1] Freescale. Kinetis Peripheral Module Quick Reference: A Compilation of Demonstration Software for Kinetis L Series Modules
https://www.dca.fee.unicamp.br/cursos/EA871/references/ARM/KLQRUG.pdf
[2] KL25 Sub-Family Reference Manual – Freescale Semiconductors (doc. Number KL25P80M48SF0RM), Setembro 2012.
https://www.dca.fee.unicamp.br/cursos/EA871/references/ARM/KL25P80M48SF0RM.pdf
[3] Freescale. CodeWarrior Development Suite – Eclipse Quick Reference Windows
https://www.dca.fee.unicamp.br/cursos/EA871/references/complementos/CWMCUQRCARD.pdf
[4] ARM. ARMv6-M Architecture Reference Manual
https://www.dca.fee.unicamp.br/cursos/EA871/references/ARM/ARMv6-M.pdf
[5] Eric Youngdale. The ELF Object File Format: Introduction.
http://www.linuxjournal.com/article/1059
[6] Eric Youngdale. The ELF Object File Format by Dissection.
http://www.linuxjournal.com/article/1060
[7] NXP. CodeWarrior Development Studio Common Features Guide.
https://www.dca.fee.unicamp.br/cursos/EA871/references/manuais/CWCFUG.pdf
[8] Arquivo de instalação do Graphviz
https://www.dca.fee.unicamp.br/cursos/EA871/references/instaladores/graphviz-2.38.msi
[9] Erich Styger. 5 Best Eclipse Plugins: #1 (Eclox with Doxygen, Graphviz andMscgen)
http://mcuoneclipse.com/2012/06/25/5-best-eclipse-plugins-1-eclox-with-doxygen-graphviz-and-mscgen/
[10] Dimitri van Heesch. Doxygen; Special Commands.
https://doxygen.nl/manual/commands.html
[11] Dimitri van Heesch. Doxygen Manual.

https://doxygen.nl/manual/index.html
[12] ISO-8859-1 and ISO-8859-15 Characters in Oct, Hex and HTML http://www.pjb.com.au/comp/diacritics.html
[13] Erich Styger. Creating and using Libraries with ARM gcc and Eclipse https://mcuoneclipse.com/2013/02/12/creating-and-using-libraries-with-arm-gcc-and-eclipse/
[14] Arquivo de instalação de Eclox. https://www.dca.fee.unicamp.br/cursos/EA871/references/instaladores/anb0s-eclox-a7d7762.zip
[15] Eclipse CDT 8.0 Cheat Sheet https://www.dca.fee.unicamp.br/cursos/EA871/references/complementos_ea871/eclipseCDT8.0-cheatsheet.pdf
[16] Erich Styger. text, data and bss: Code and Data Size Explained https://mcuoneclipse.com/2013/04/14/text-data-and-bss-code-and-data-size-explained/
[17] Arquivo de instalação de Mscgen. https://www.dca.fee.unicamp.br/cursos/EA871/references/instaladores/mscgen_0.20.exe
[18] Projects using Doxygen https://doxygen.nl/projects.html
[19] Doxygen - Internationalization https://doxygen.nl/manual/langhowto.html

Agosto de 2016
Revisado em Fevereiro de 2017
Revisado em Agosto de 2022