Capítulo 2

Ambiente de Desenvolvimento – *Software*

Autores: Wu Shin-Ting e Antônio Augusto Fasolo Quevedo

O *Integrated Development Environment* (IDE) CodeWarrior 10 (CW10) 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, a versão 10 foi desenvolvida 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 dupla de alunos, os quais podem 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. 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 10.4, executando em Windows 7. 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 Estrutura Básica de um Programa C no IDE

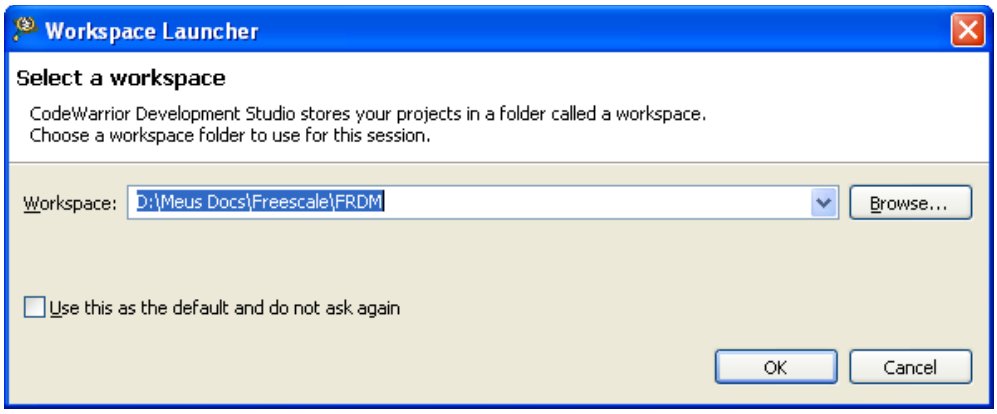
O capítulo 1 em [1] apresenta, sob o ponto de vista de programação, um procedimento típico para inicializar o microcontrolador MKL25Z. Como este procedimento é comum para uma grande maioria dos projetos, o IDE CodeWarrior dispõe de um mecanismo que gera automaticamente uma estrutura básica de programas 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.

2.1.1 Criação de um Projeto

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.

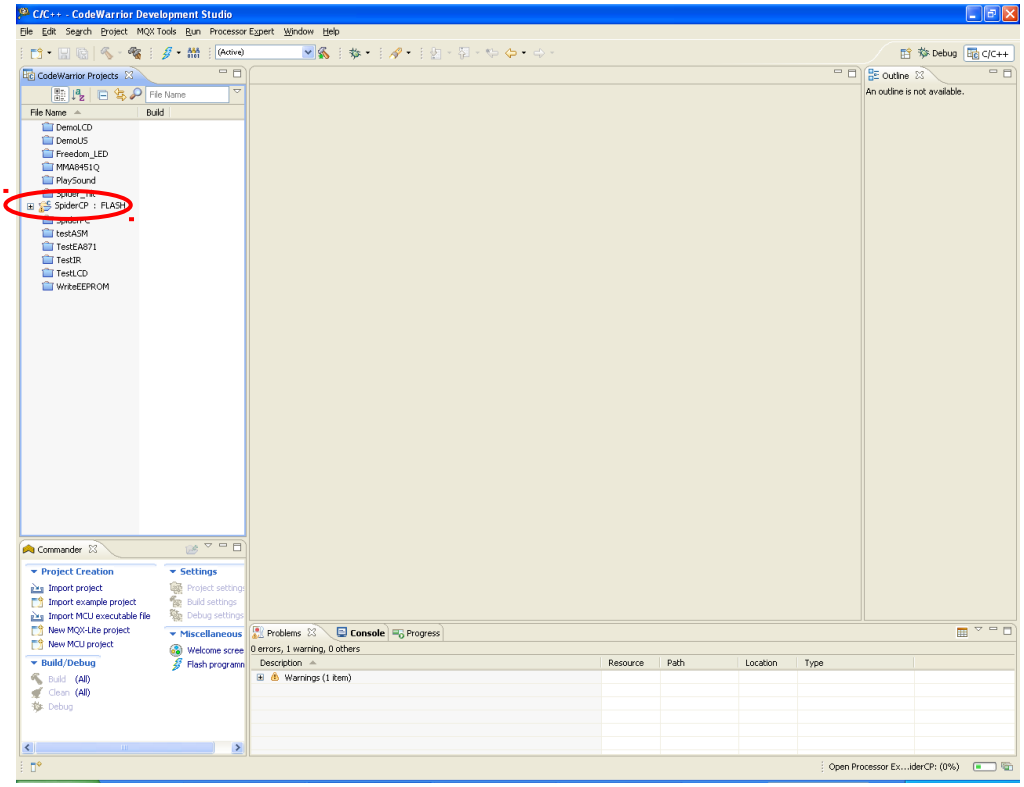
1. No Windows, vá ao menu "*Iniciar* > *Programas* > *Freescale CodeWarrior* > *CW for MCU v10.6* > *CodeWarrior*", ou chame o programa através do ícone do CodeWarrior 10.6 na área de trabalho.

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*.

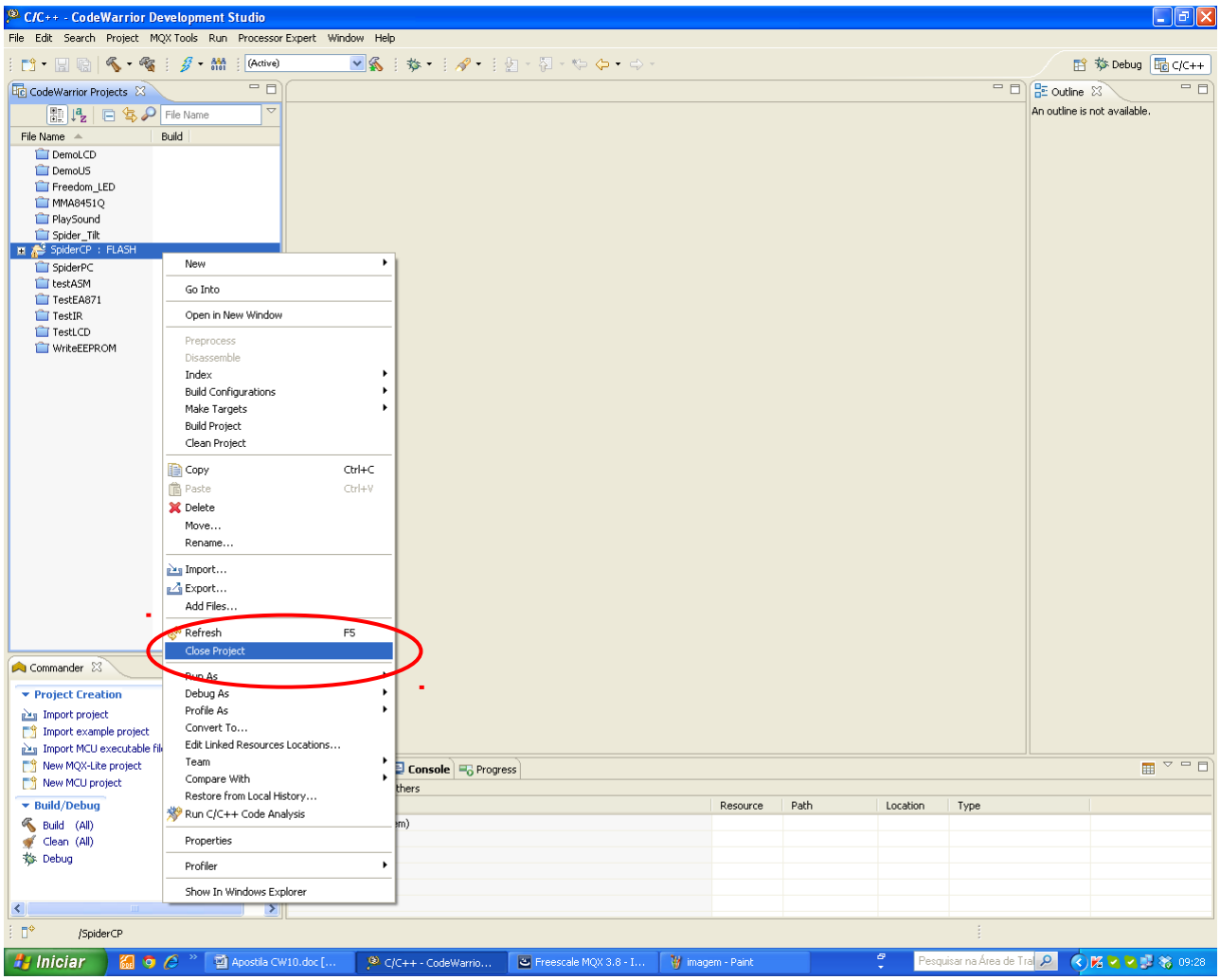


3. O CodeWarrior 10 vai abrir na perspectiva de programação. Pode-se ver à esquerda um painel com todos os projetos do *workspace*. Uma pasta aberta significa que aquele projeto está aberto.

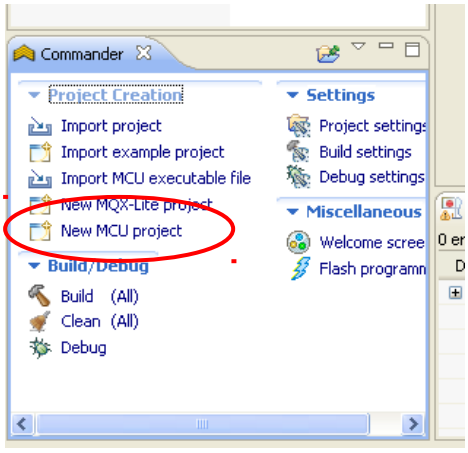
Na primeira utilização do *workspace*, ao invés da perspectiva de programação, pode aparecer um menu geral. Neste caso, basta escolher a opção "*Go to Workspace*".



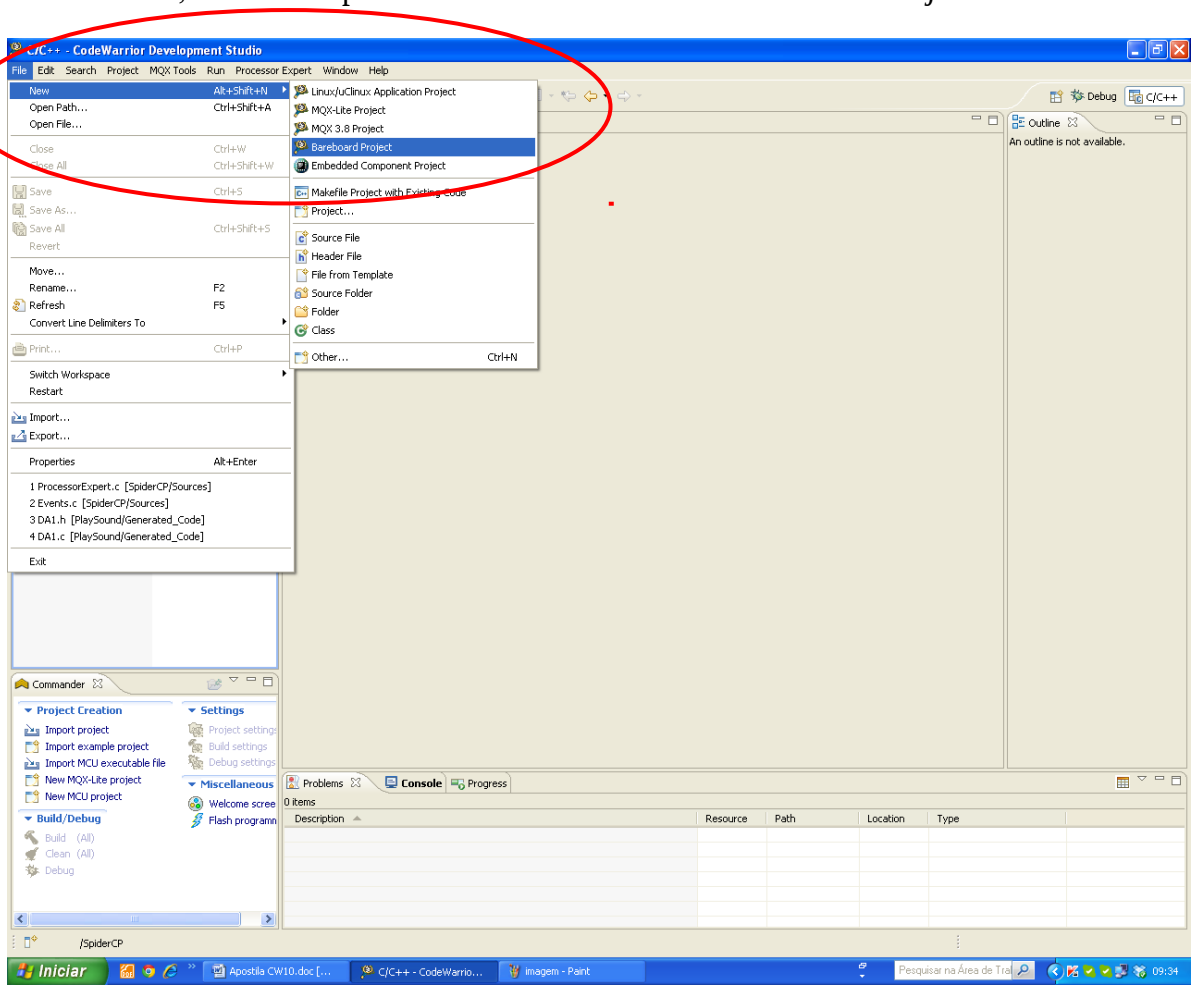
4. Para evitar problemas, evite ter mais de um projeto aberto de cada vez. Se houver diversos projetos abertos, feche os desnecessários clicando com o botão direito sobre a pasta do projeto aberto e escolhendo a opção "*Close Project*".



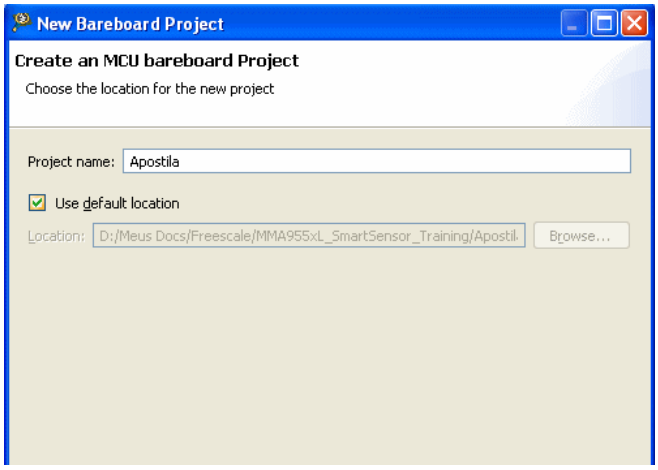
5. A janela *Commander* no canto inferior esquerdo da perspectiva agrupa os comandos mais comuns. Para criar um novo projeto, basta clicar em *New MCU project* nesta janela.



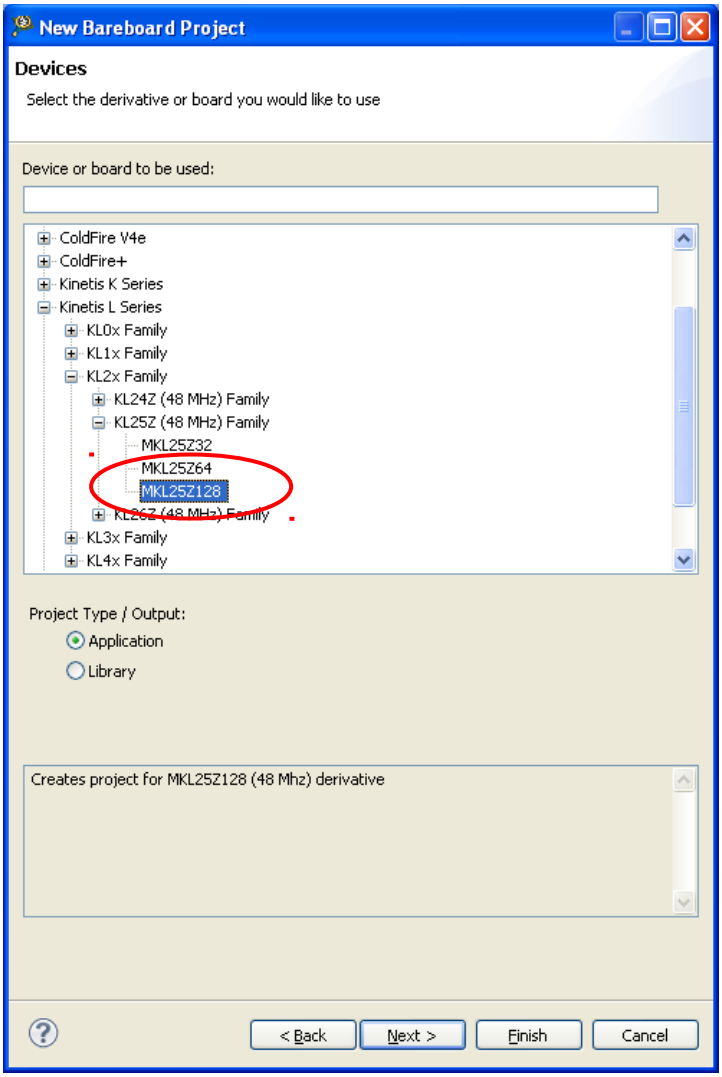
Alternativamente, no menu superior selecione *File* > *New* > *Bareboard Project*.



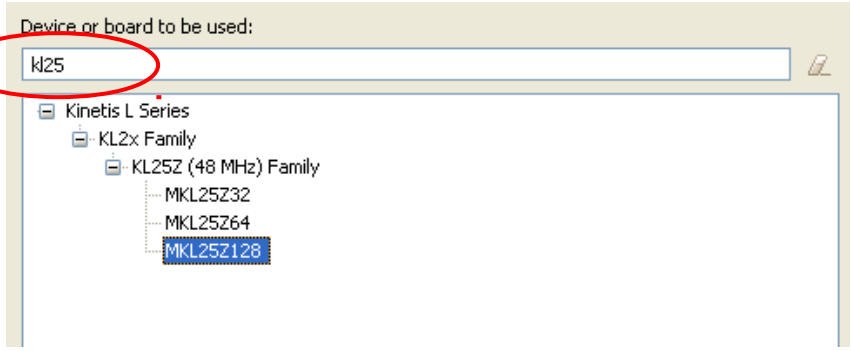
6. Na janela que se abre, escolha um nome para o projeto. No exemplo, foi escolhido o nome "Apostila". Se a caixa de seleção "*Use default location*" estiver selecionada, os arquivos serão salvos na pasta do *workspace*, que pode ser vista logo abaixo da caixa. Desmarcando a caixa, pode-se escolher outra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



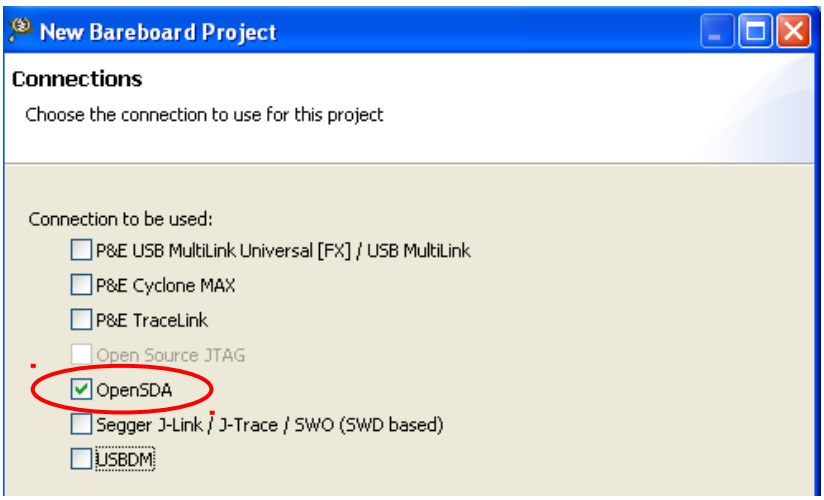
7. Agora deve-se escolher o microcontrolador ou a placa de desenvolvimento a ser utilizada. Neste curso usamos o controlador MKL25Z128, na placa FRDM-KL25. Na árvore de opções, siga o caminho "*Kinetis L Series* > *KL2x Family* > *KL25Z (48MHz) Family* > *MKL25Z128*". Clique "*Next*" em seguida.



Alternativamente, pode-se digitar “KL25” 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 KL25, e assim pode-se selecionar mais facilmente o processador-alvo.

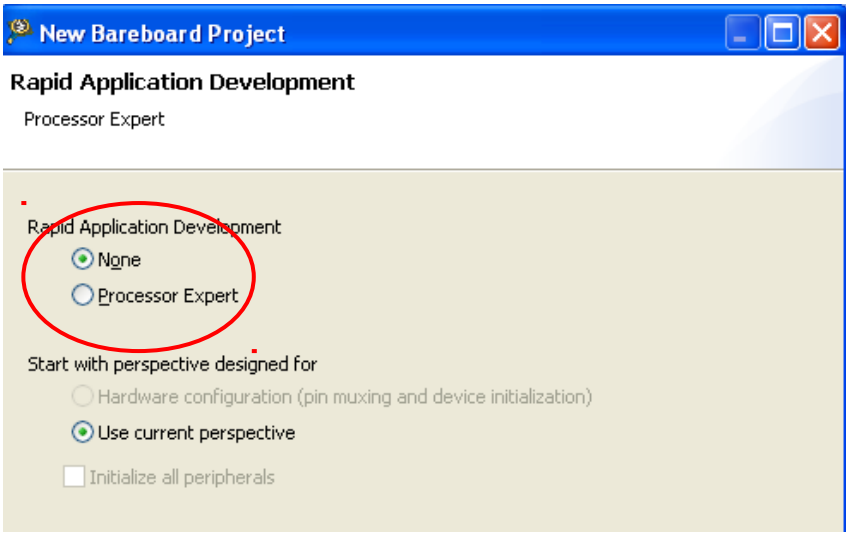


8. Agora, aparece uma lista de possíveis conexões. A conexão estabelecida entre o computador e a placa permite transferir os programas executáveis para a placa, bem como executar a depuração dos programas carregados em MCUs em tempo real. Neste caso, selecione a opção "*OpenSDA*”, que é o *hardware* 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*. Por exemplo, na opção "I/O Support", a opção *default* (UART) e a opção *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 se evitar inicializações desnecessárias. Sugerimos usar a terceira opção, a menos que o experimento use as funções acima citadas.

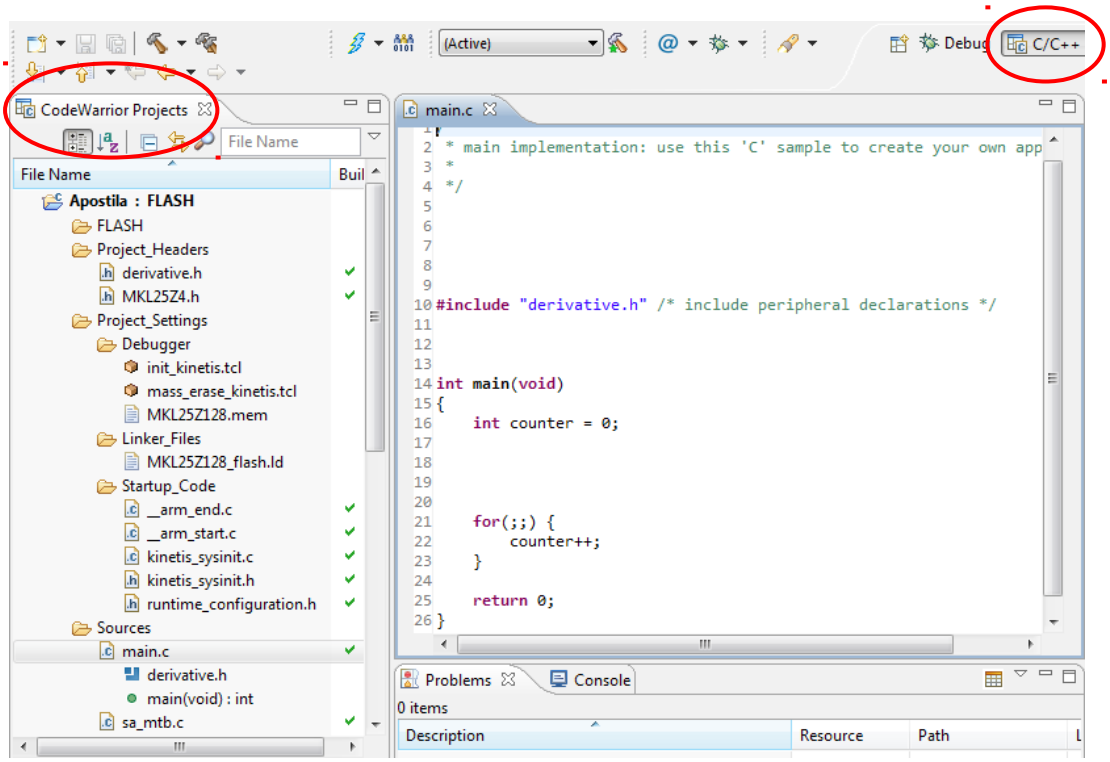
10. Nesta próxima janela, definimos se o CodeWarrior usará ou não uma ferramenta que facilita a geração de códigos para a inicialização dos módulos do MCU e funções em alto nível para controlar estes módulos (*Processor Expert*). Neste curso, esta ferramenta não será utilizada, e a mesma só está disponível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



2.1.2 Estrutura dos Arquivos Gerados

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. 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.



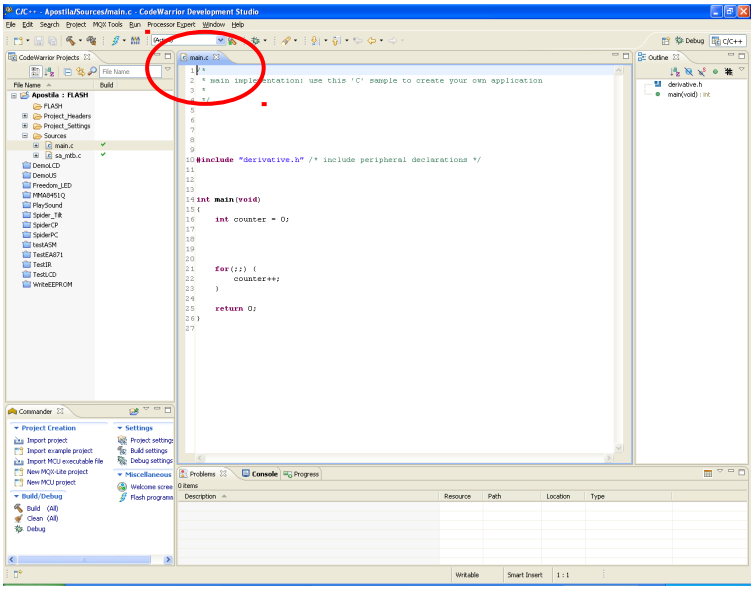
Na pasta *Project Headers* temos os arquivos de cabeçalho onde estão incluídas as declarações de tipos de dados e variáveis, os protótipos de funções e as macros. Dois arquivos contendo as declarações relacionadas especificamente com o microcontrolador MKL25Z4 foram adicionadas automaticamente: derivative.h e MKL25Z4.h. A pasta *Sources* é reservada para todos os códigos-fonte do projeto em C. 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 endereço 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. 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 que mencionamos anteriormente.

Observe que no arquivo *kinetis_sysinit.c*, o endereço da rotina *__thumb_startup(void)* é armazenado no vetor 1 da tabela de vetores 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 configurados com o procedimento dado na Seção 2.1.1 inicializam a sua execução pela rotina *__thumb_startup*. Só depois da inicialização é chamada, 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.

2.1.3 Edição de Códigos Específicos

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. O programa inclui o arquivo de cabeçalho *derivative.h* e implementa a rotina *main* chamada na rotina *__thumb_startup*. 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. Algumas facilidades nesta área de edição podem ser consultada em [3]. Note que o CodeWarrior IDE é desenvolvido em cima do ambiente de desenvolvimento integrado de C e C++ do Eclipse, Eclipse CDT [14]; portanto, vale usar todas as teclas de atalho listadas em [15].

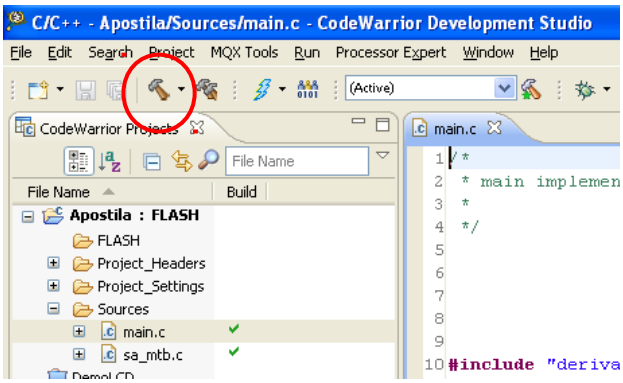


2.1.4 Adição de Novos Arquivos

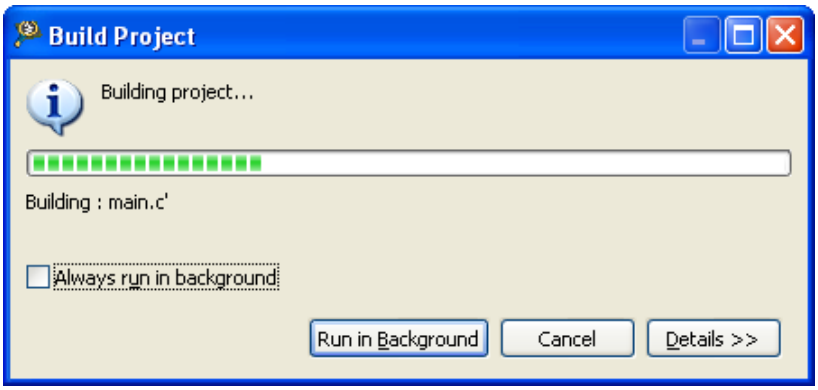
É muito comum separar as funções de um programa em vários arquivos pequenos. Para adicionar um novo arquivo numa pasta, por exemplo *Project_Headers* ou *Sources*, basta selecionarmos a pasta em que queremos adicionar um novo arquivo e entrarmos o nome do arquivo seguindo o caminho de comandos "*File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

2.2 Compilação, Linkagem e Geração do Arquivo elf

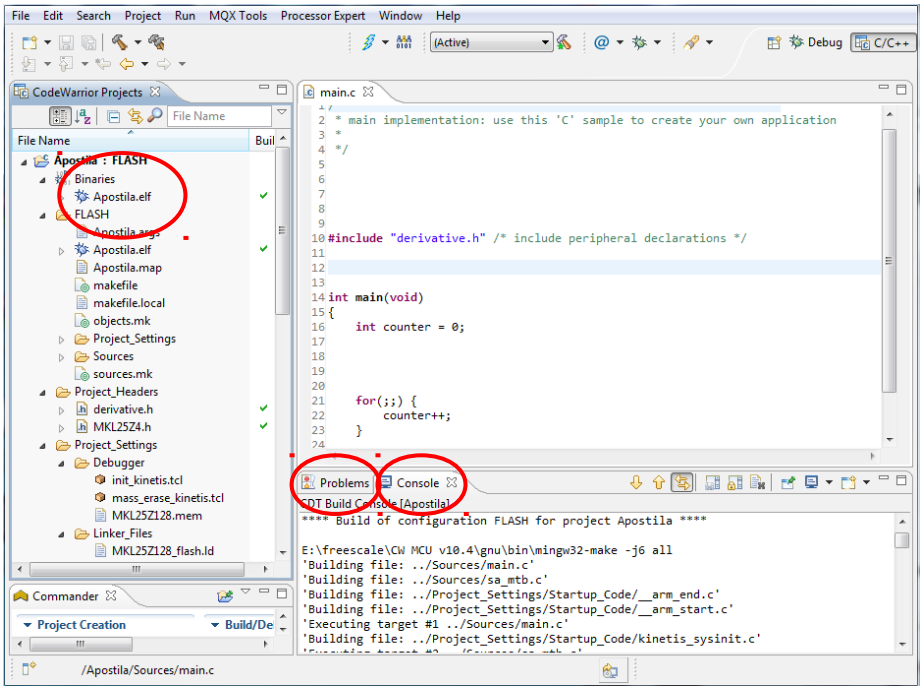
Quando finaliza a edição dos programas de um projeto, precisamos compilar todos os arquivos do projeto e linká-los para termos um código executável no formato elf (*Extensible Linking Formt*) [5.6]. Este arquivo contém as informações necessárias para linkagem, relocação de acordo com os dados contidos no arquivo *MKL25Z128_flash.ld*, e a execução do programa do projeto. Para isto, podemos clicar no botão da barra superior com o ícone do martelo, ou selecionar "*Project* > *Build Project*" na barra de ferramenta superior.



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 ocorra 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.

2.2.1 Diretivas de Compilação

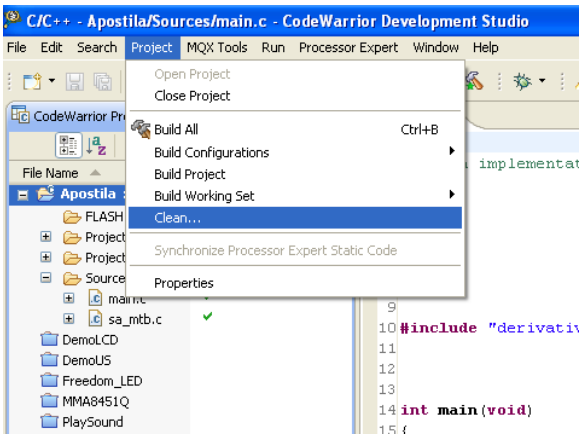
Antes de compilar o código-fonte, há um programa, conhecido como o pré-processador, que modifica o conteúdo deste conforme as diretivas de compilação. Estas diretivas são diferenciadas por terem o símbolo # no início da linha de um código-fonte. Dentre as diretivas mais utilizadas temos:

- #**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.

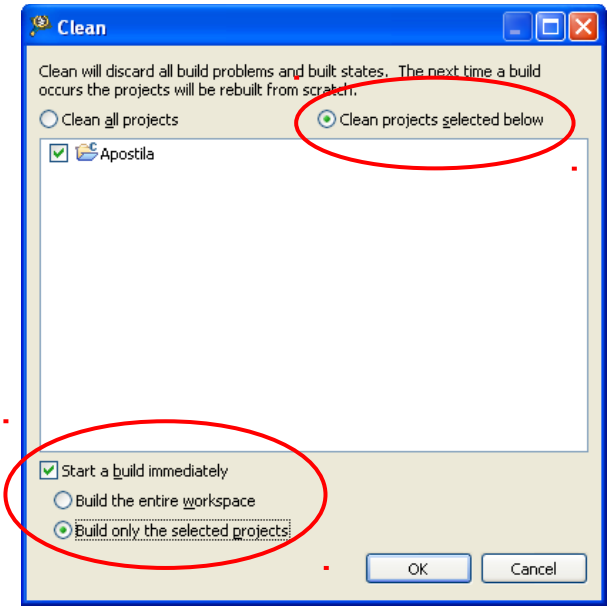
O código gerado pelo IDE utiliza a diretiva "include" para que o pré-processador insira todos os códigos do arquivo de cabeçalho "derivative.h" no arquivo main.c antes da compilação desta. Observe que esta inclusão é feita de forma recursiva até que todas as diretivas na hierarquia de inclusão sejam resolvidas.

2.2.2 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*"



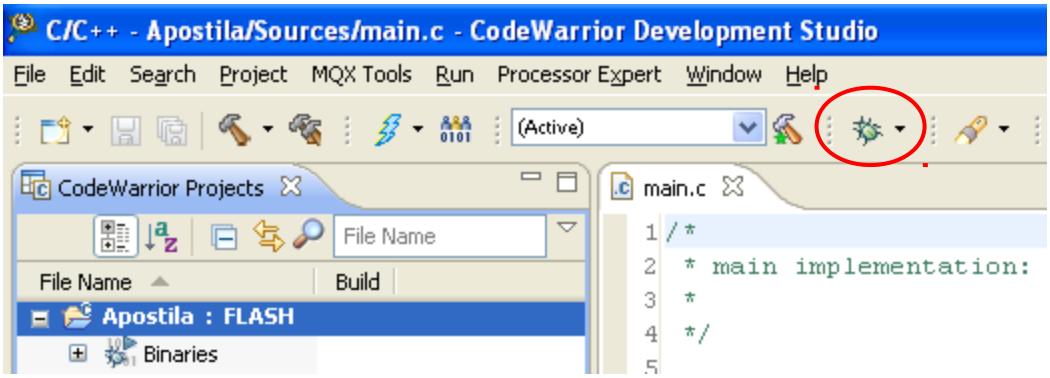
Na janela que se abre, selecione a opção *Clean projects selected below* e verifique se seu projeto está selecionado na lista. Marque a caixa *Start a build immediately* e selecione a opção *Build only the selected projects*. Desta forma, o projeto ativo será limpo e logo após será iniciado o *build* do mesmo. Teremos uma janela mostrando as etapas do *clean*, que costumam ser rápidas.



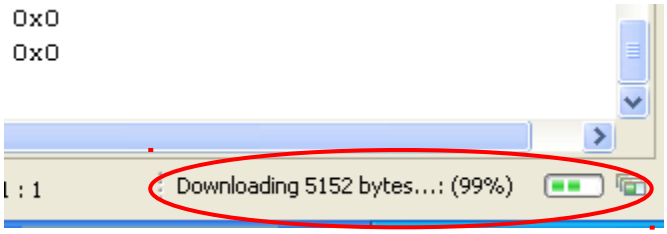
2.3 Transferência do Código para Microcontrolador

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 onde está executando o aplicativo

CodeWarrior e clicar no botão da barra superior com o ícone do besouro, ou seguir no menu principal o caminho "*Run* > *Debug*", ou ainda pressionar a tecla F11.



Certifique-se que o projeto "Apostila" esteja selecionado 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 *Control Panel* > *System* > *Device Manager* > *Ports (COM &LPT)*. O progresso de transferência ou da carga do código no microcontrolador é indicado pela barra de progresso no canto direito inferior do IDE.

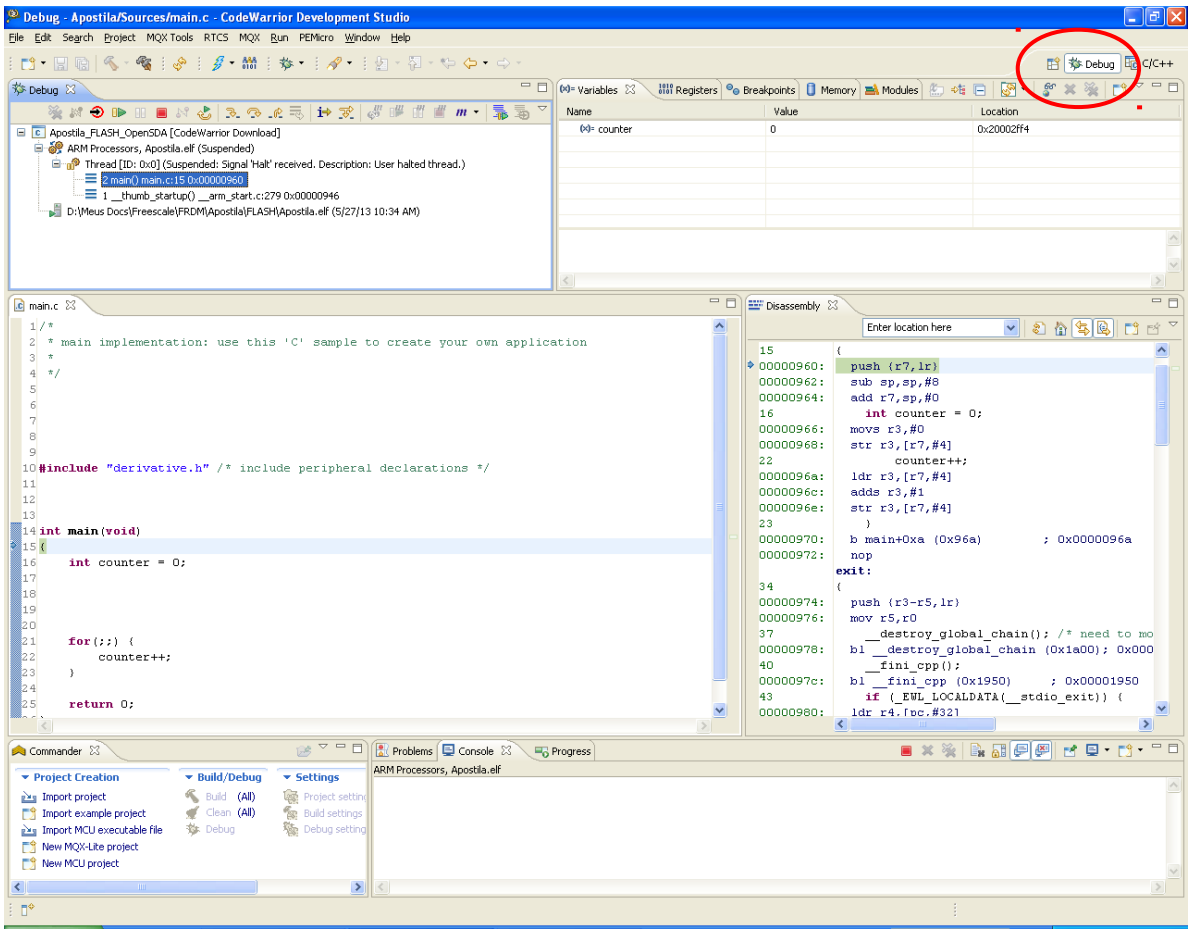


2.4 Depuração

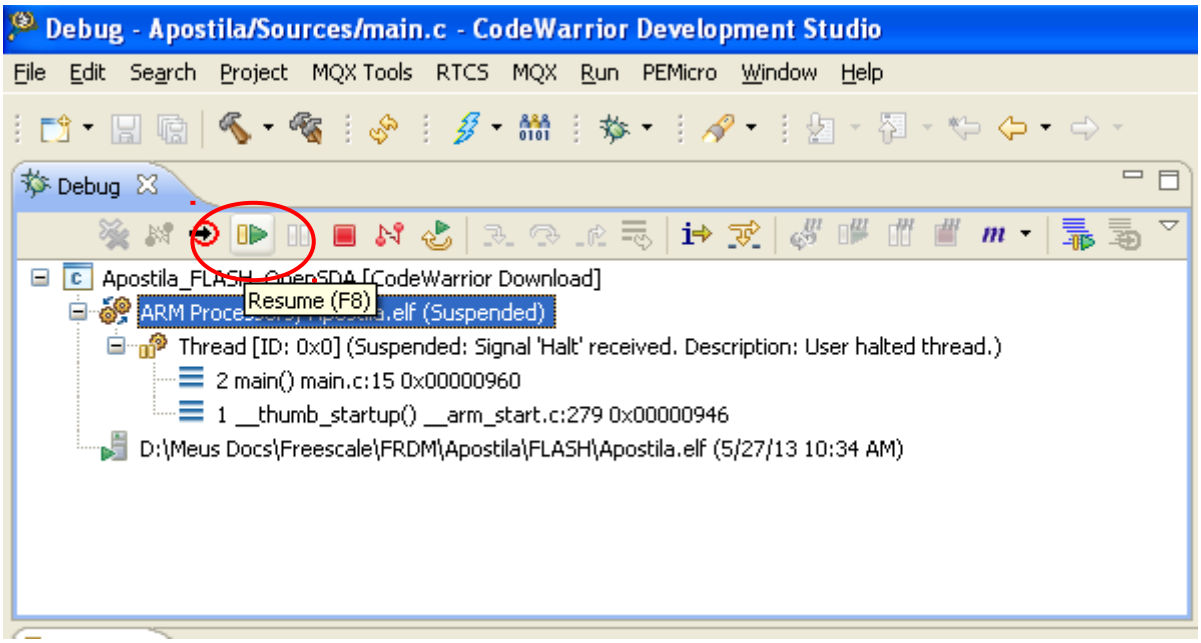
Assim que finalizar a transferência do código executável, a perspectiva é mudada automaticamente para a de **Depuração**. Depuradores são aplicativos que permitem ao programador monitorar a execução de um programa passo a passo, pará-lo, reiniciá-lo, ativar os *breakpoints* (pontos de parada), alterar 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: aba *Debug* 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 0x00000960 (linha destacada na aba *Disassembly*) que corresponde ao início da rotina *main* (linha destacada 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*. 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 projetista.

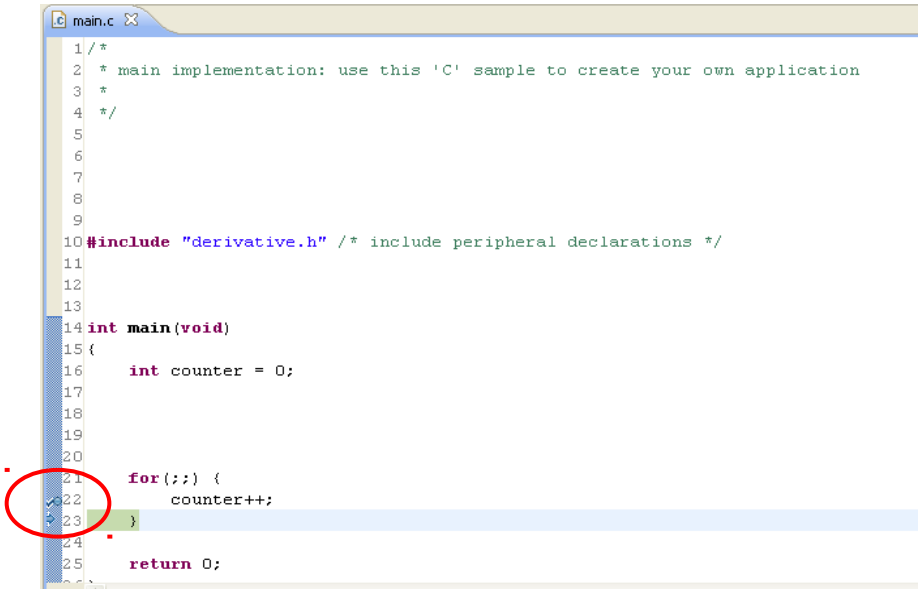


Selecionando o triângulo verde (*Resume*) na barra de ferramenta da aba *Debug*, ou pressionando F8, a execução do programa será retomada.

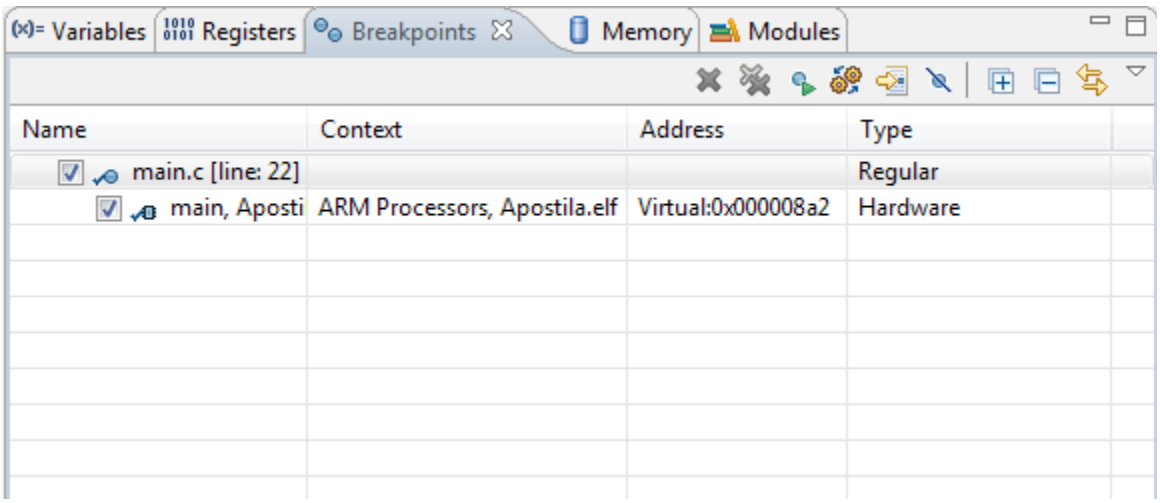


2.4.1 Aba *Breakpoints*

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.



A aba *Breakpoints* nos permite gerenciar os pontos de parada. No nosso caso, com um *breakpoint* setado, temos na janela somente uma entrada. Uma outra alternativa é através do item *Run* que fica na barra de ferramenta da janela principal. Ao clicar sobre este item, aparece um menu *pop-up* que contém várias ações relacionadas aos *breakpoints*.

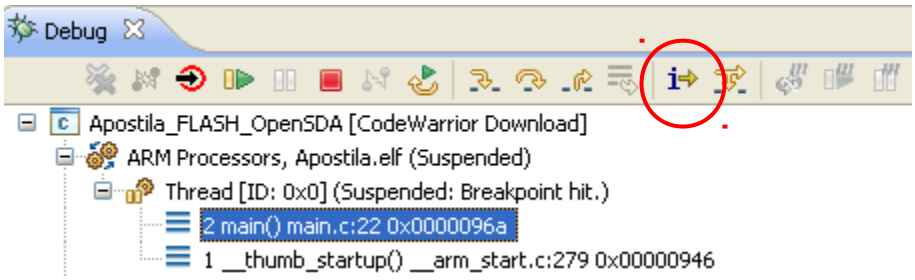


Vale ressaltar que a quantidade total de *breakpoints* suportada no IDE é 2 (dois).

2.4.2 Aba *Debug*

A barra de ferramenta da aba *Debug* é composta pelos seguintes ícones, da esquerda para direita, que facilitam o controle do fluxo de execução de um programa:

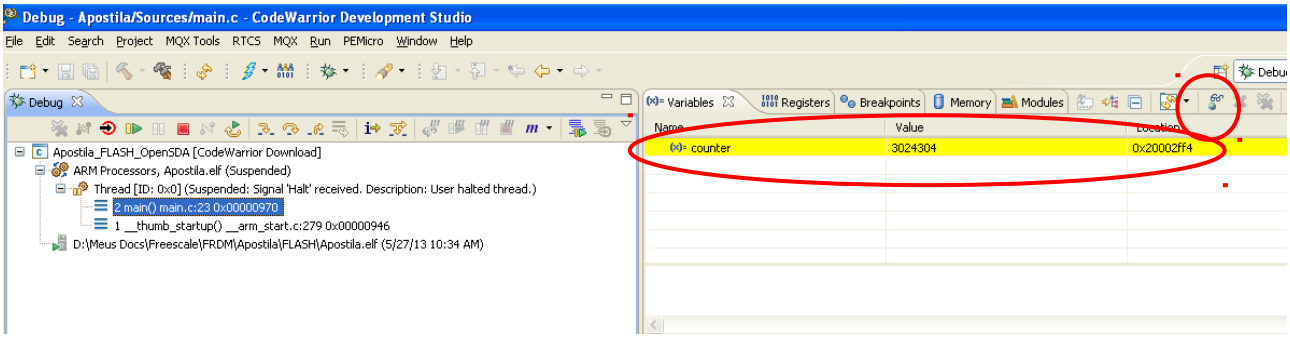
- *Reset* (círculo vermelho com seta preta): reinicializa o microcontrolador e o contador de programa (PC) é carregado com o endereço inicial da rotina *__thumb_startup*.
- *Resume*: retoma a execução contínua do programa pausado (por *breakpoints* ou por *Suspend*) com o endereço corrente do PC.
- *Suspend* (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.
- *Restart*: retoma a execução do programa a partir da rotina *main* do programa de projeto.
- *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 se o modo *Instruction Stepping* estiver desabilitado. Se o modo *Instruction Stepping* estiver habilitado, o avanço será no código *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 aba *Disassembly*).



Note que os comandos de avanço manual no fluxo de controle de instruções só valem para operações que não sejam sobre pontos flutuantes. E, 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 durante o processo de depuração.

2.4.3 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*". Podemos também incluir nesta aba as variáveis globais do programa de projeto com uso do comando "*Add Globals*" (ícone óculos).

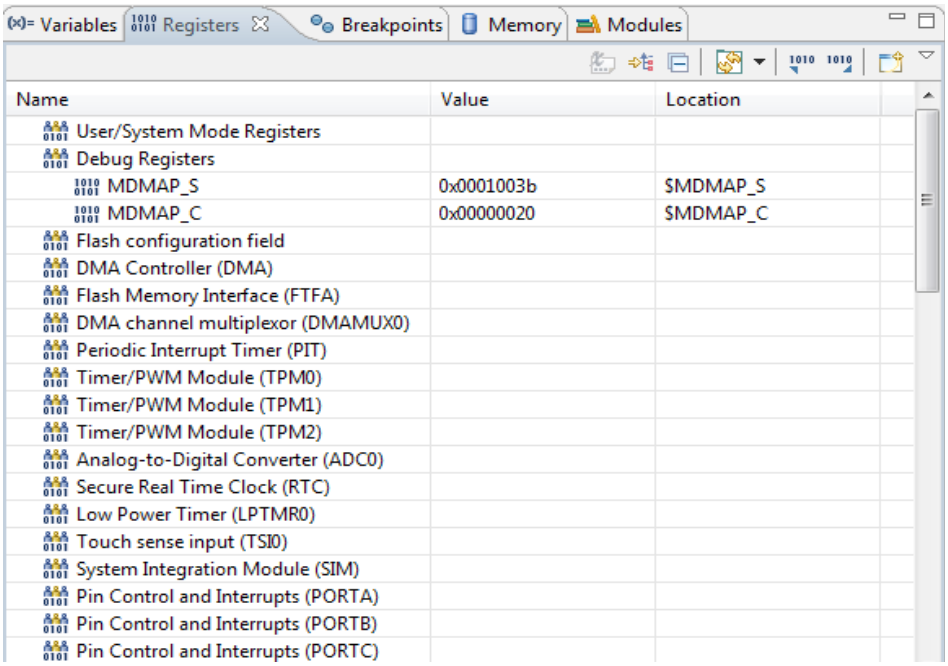


O valor da variável é sempre atualizado conforme o fluxo de execução do programa pausado. E ele pode ser modificado editando o novo valor diretamente no campo "Value". Clicando o botão direito na área da aba *Variables*, aparece um menu *pop-up* através do qual pode-se escolher com a opção *Format* o formato que desejamos visualizar a variável selecionada: decimal, hexadecimal, binário, etc. Por este menu, podemos também alterar o seu valor (*Change Value* …) e adicionar novas variáveis globais à lista (*Add Global Variables* …).

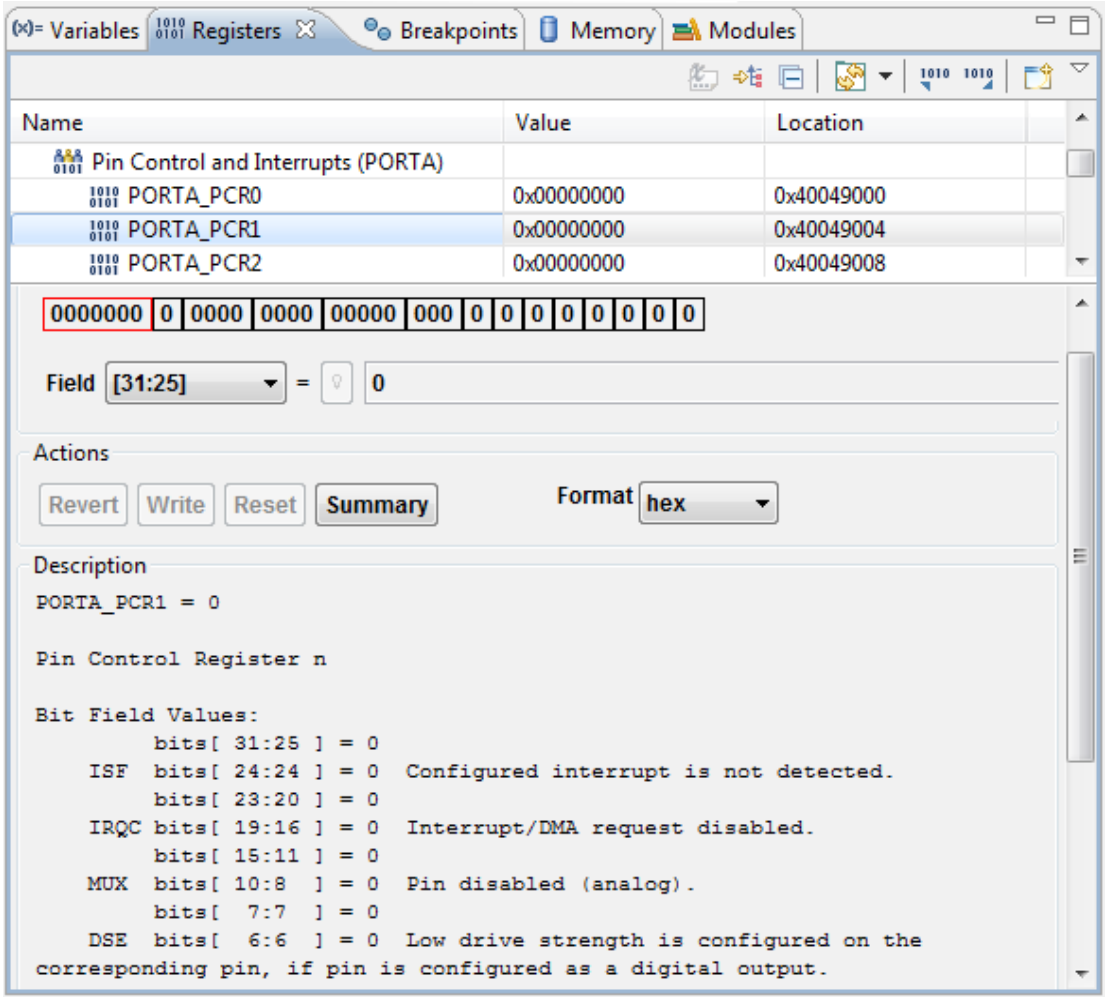
O valor das variáveis pode ser também conferido na Aba *Memory* com uso do endereço correspondente que aparece na coluna *Location*.

2.4.4 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 sob a nossa responsabilidade.

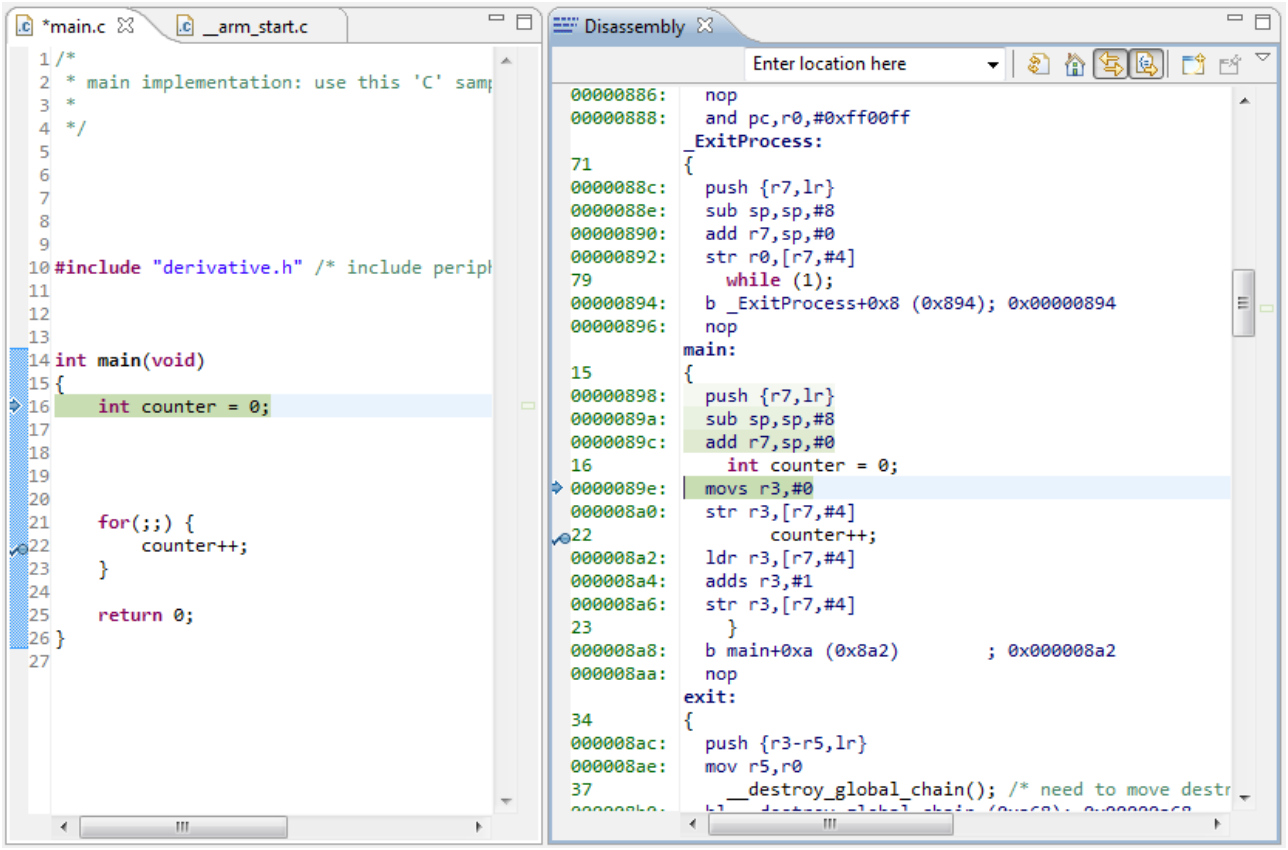


Ao selecionarmos um registrador é exibido na parte inferior da aba o conteúdo do registrador por função associada aos seus *bits*. Podemos através desta interface checar e modificar as configurações.



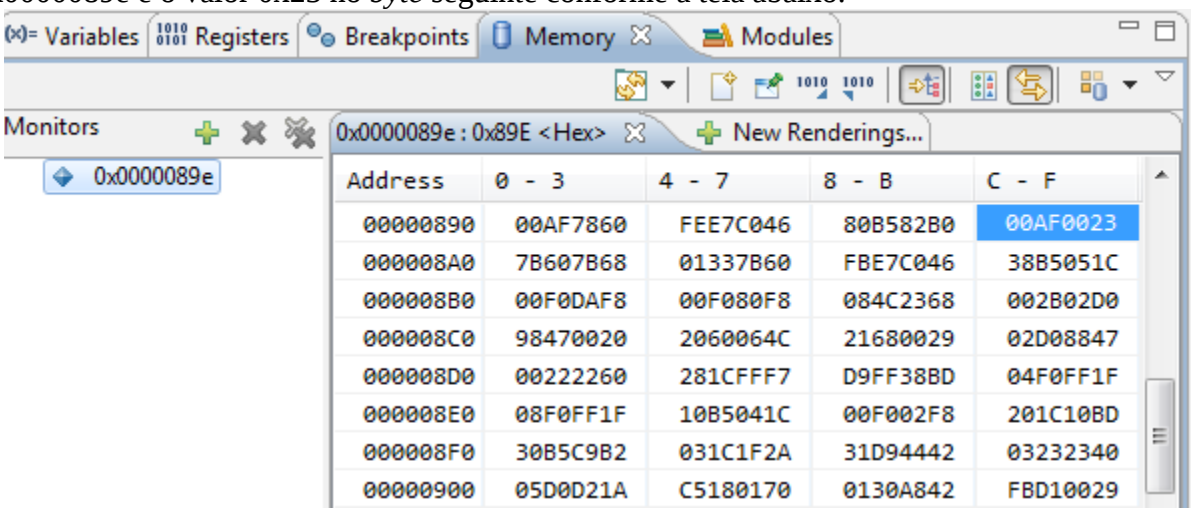
2.4.5 Aba *Disassembly*

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.



2.4.6 Aba *Memory*

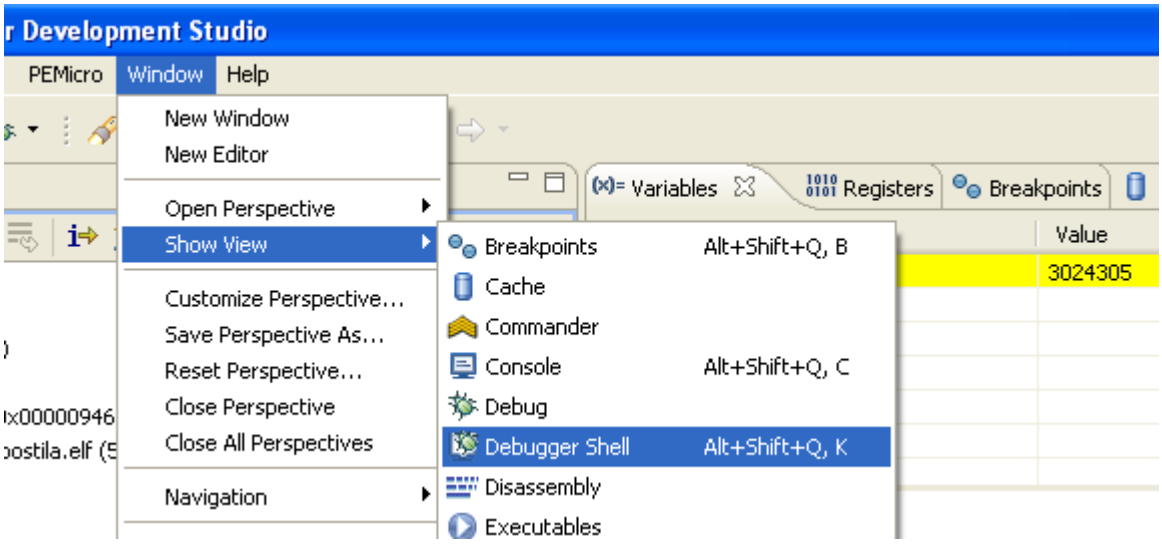
A aba *Memory* nos permite monitorar o conteúdo do espaço de endereços. Por exemplo, podemos monitorar o conteúdo de um bloco de endereços a partir do endereço 0x0000089e onde está armazenada a instrução “movs r3,#0”, clicando em + verde. Na caixa de diálogo que aparecerá será inserido o endereço de interesse. 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. Como a ordenação dos *bytes* na memória é *little endian*, teremos o valor 0x00 no endereço 0x0000089e e o valor 0x23 no *byte* seguinte conforme a tela abaixo.



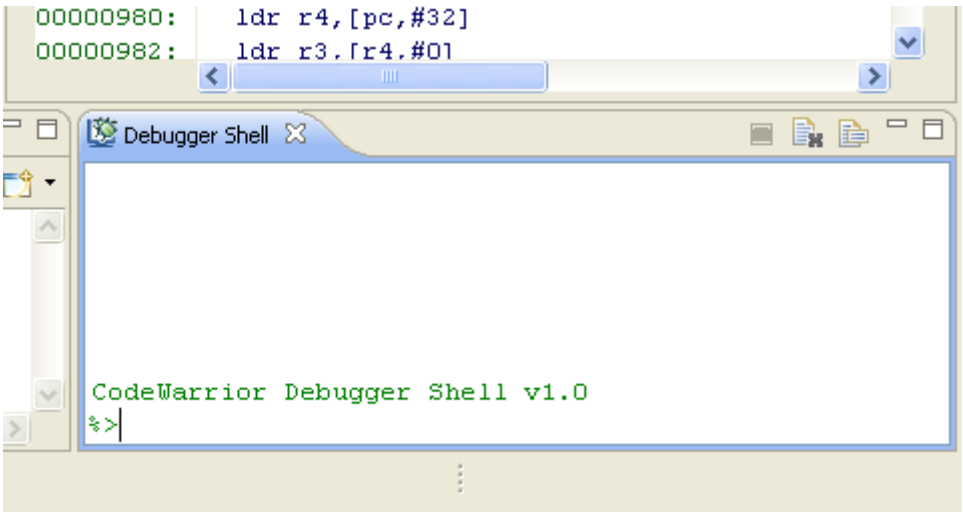
Uma característica interessante da aba *Memory* é a possibilidade de “renderizar”, ou visualizar, o conteúdo do espaço de endereços em diferentes formas.

2.4.7 Aba *Debugger Shell*

A aba *Debugger Shell* é uma janela de texto, que permite usar linha de comando para modificar posições de memória e registradores, além de outras funções de depuração. Para que esta janela apareça, na perspectiva *Debug*, vá ao menu superior e clique em “*Window* > *Show View* > *Debugger Shell*”.



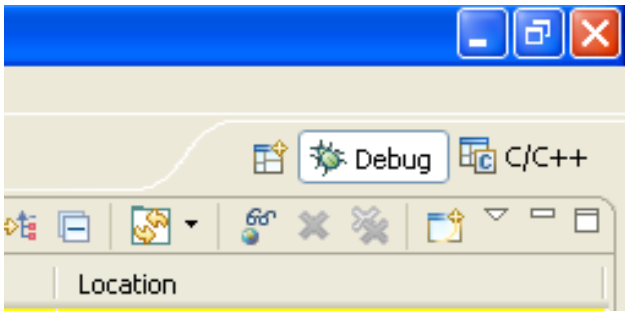
No canto inferior direito, a aba *Debugger Shell* irá aparecer.



Digitando “*help*”, pode-se ver a lista de comandos possíveis. Digitando “*help <comando>*”, pode-se ver a forma de utilização para cada comando.

2.5 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.

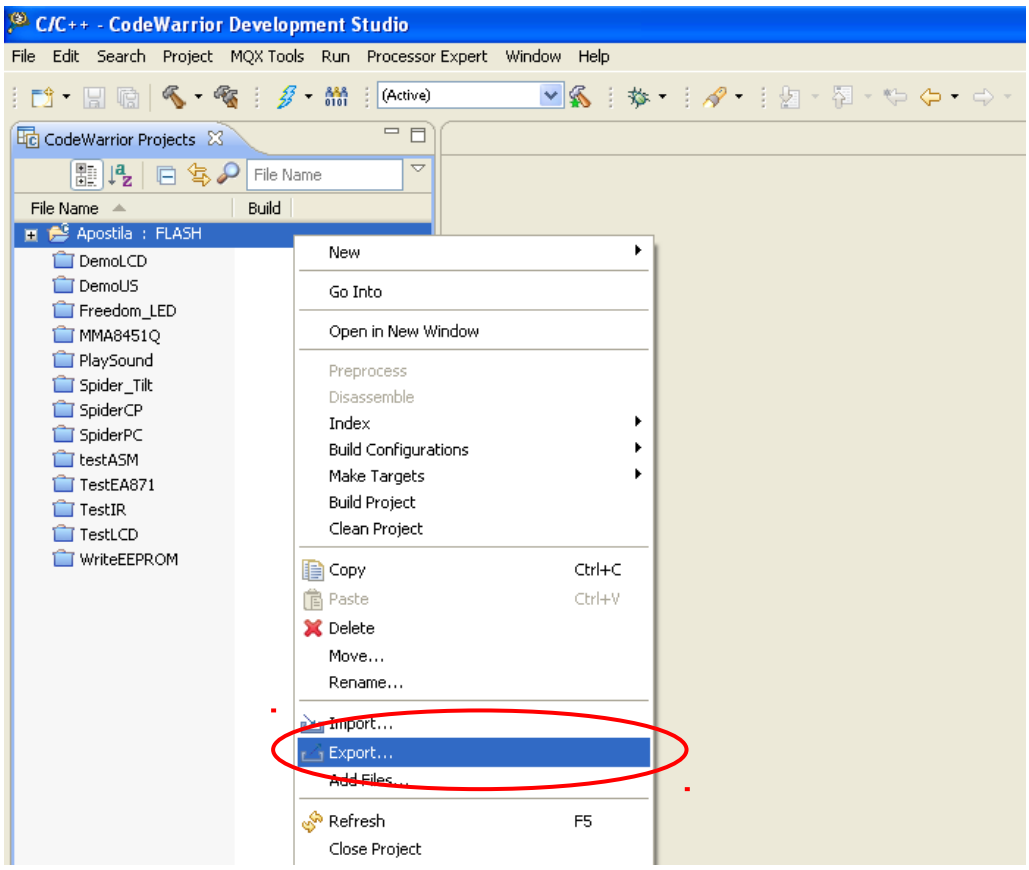


2.6 Exportação e Importação de um Projeto

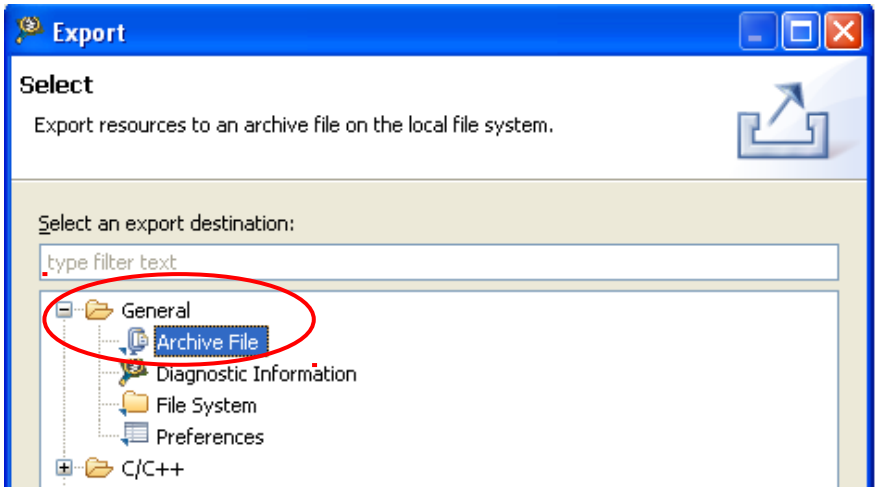
Segue-se um procedimento direto para exporta e importar um projeto no IDE CodeWarrior:

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*".

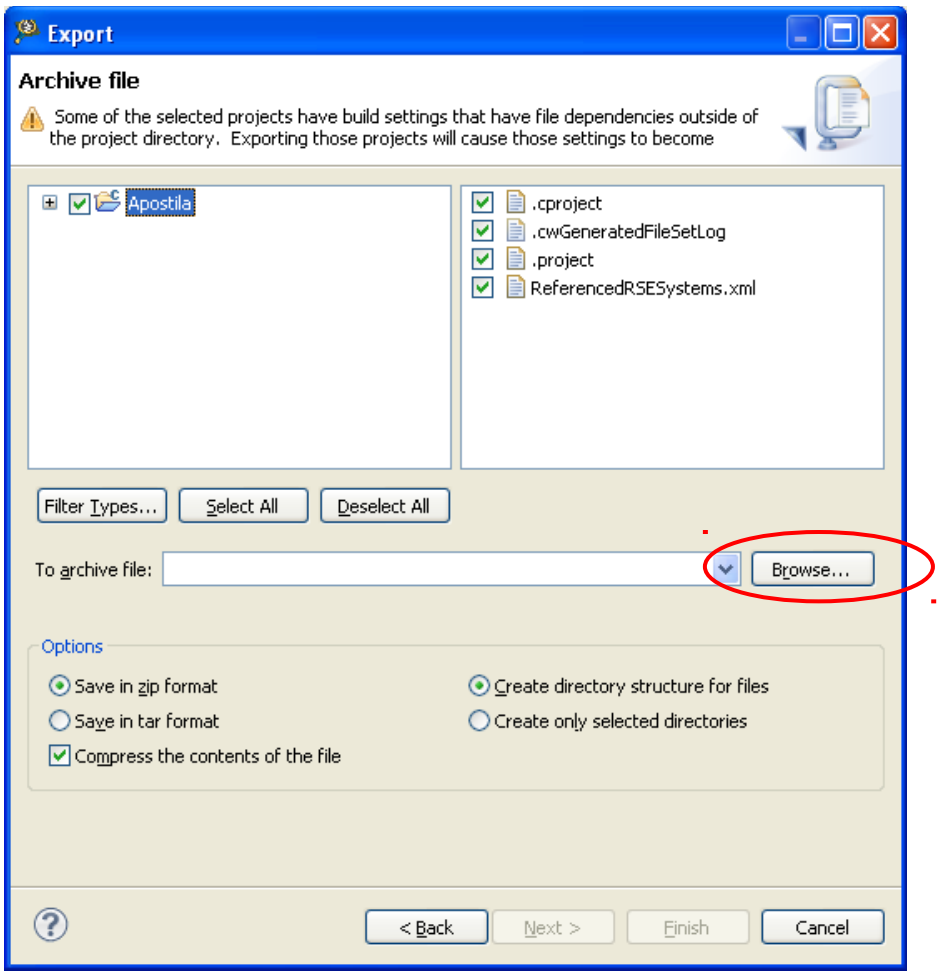
2. Clique novamente na pasta do projeto e selecione "*Export...*". Alternativamente, pode-se usar o menu "*File*", opção "*Export...*".



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

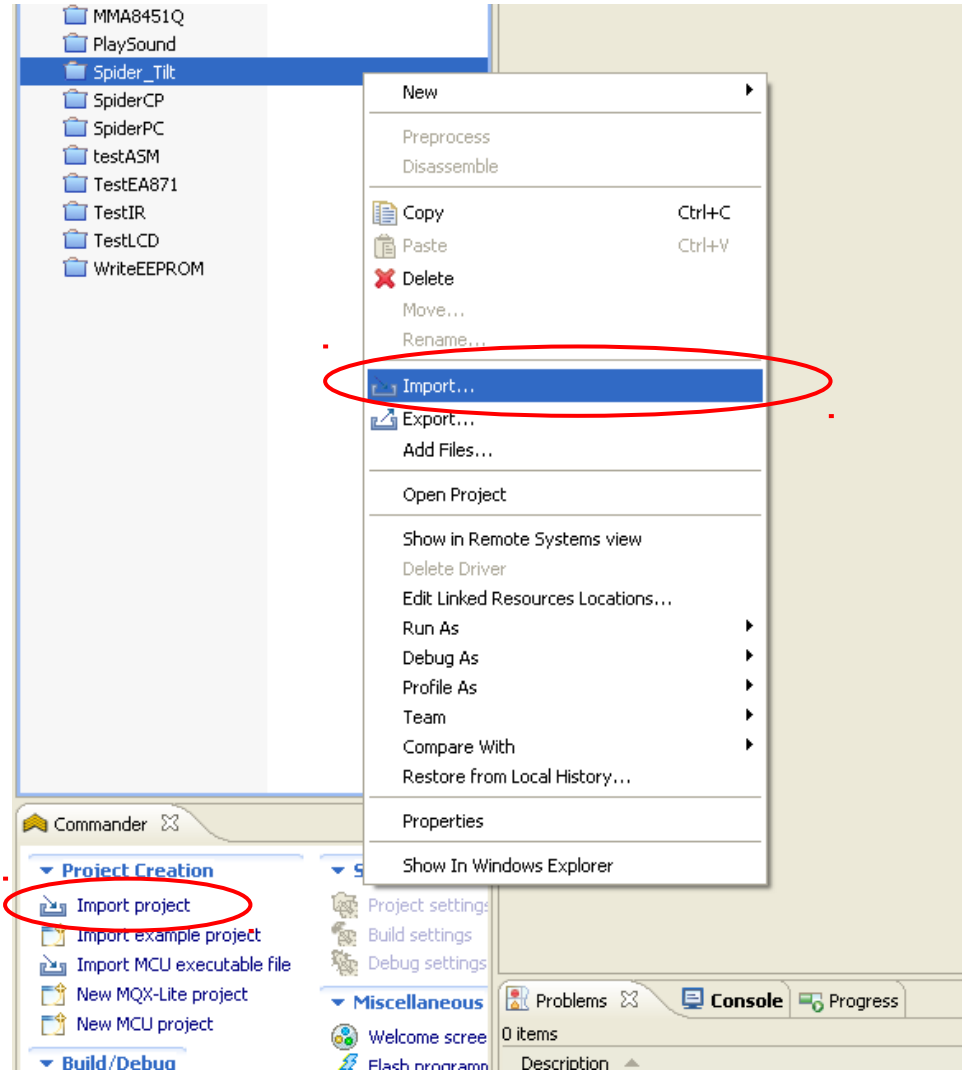


4. Nesta janela, mantenha as opções padrão, e clique no botão "*Browse...*". Selecione uma pasta para guardar o arquivo de exportação, e o nome do arquivo. Clique em "*Finish*".

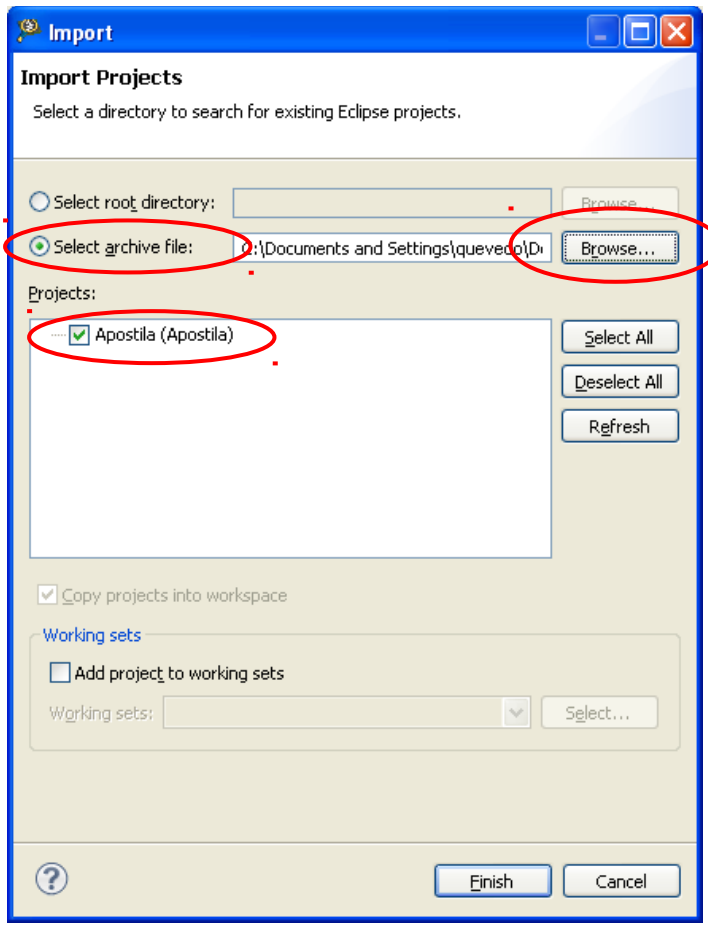


5. Um arquivo ZIP com o nome escolhido será gravado no local selecionado. Para fazer a importação, apague totalmente o projeto, inclusive os arquivos originais. Se um projeto com o mesmo nome de outro projeto no mesmo *workspace* tentar ser importado, aparecerá uma mensagem de erro.

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...*"



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*".

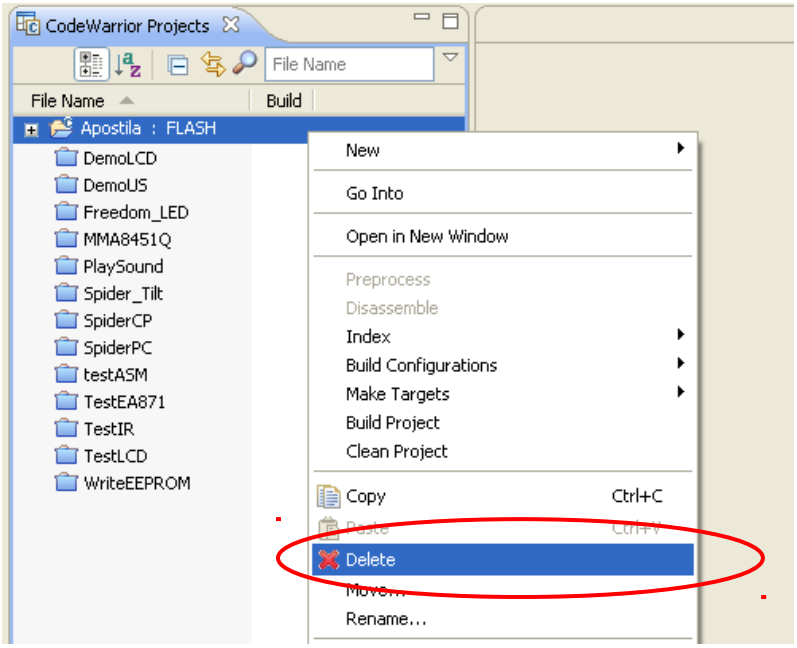


8. Pronto, seu projeto aparece agora no *workspace*.

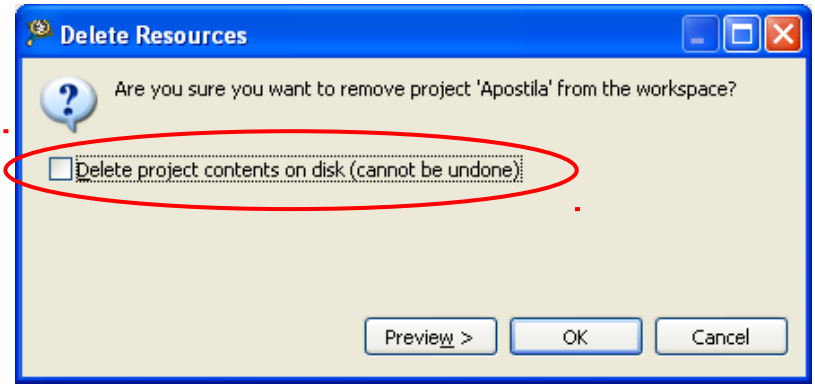
2.7 Remoção de um Projeto

Uma possível sequência de ações para remover completamente um projeto:

1. Clique com o botão direito no projeto e selecione a opção "*Delete*". Alternativamente, selecione o projeto com um clique do botão esquerdo, e pressione a tecla “*Delete*”.



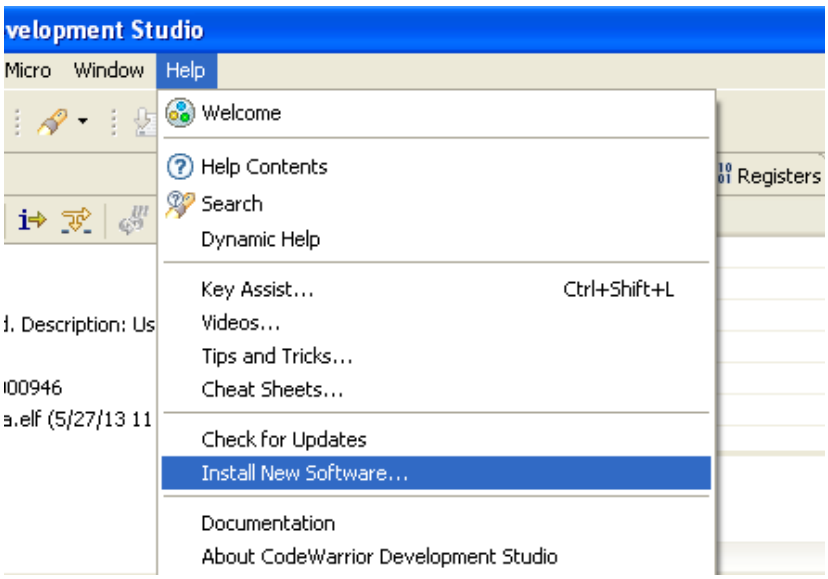
3. Uma nova janela se abrirá para confirmar o apagamento ou cancelá-lo. Por *default*, os arquivos não serão apagados. O projeto apenas sairá do *workspace*, mas continuará fisicamente na mesma pasta. Se quiser realmente apagar os arquivos do projeto, marque a caixa "*Delete project contents on disk (cannot be undone)*". Use esta opção com cuidado.



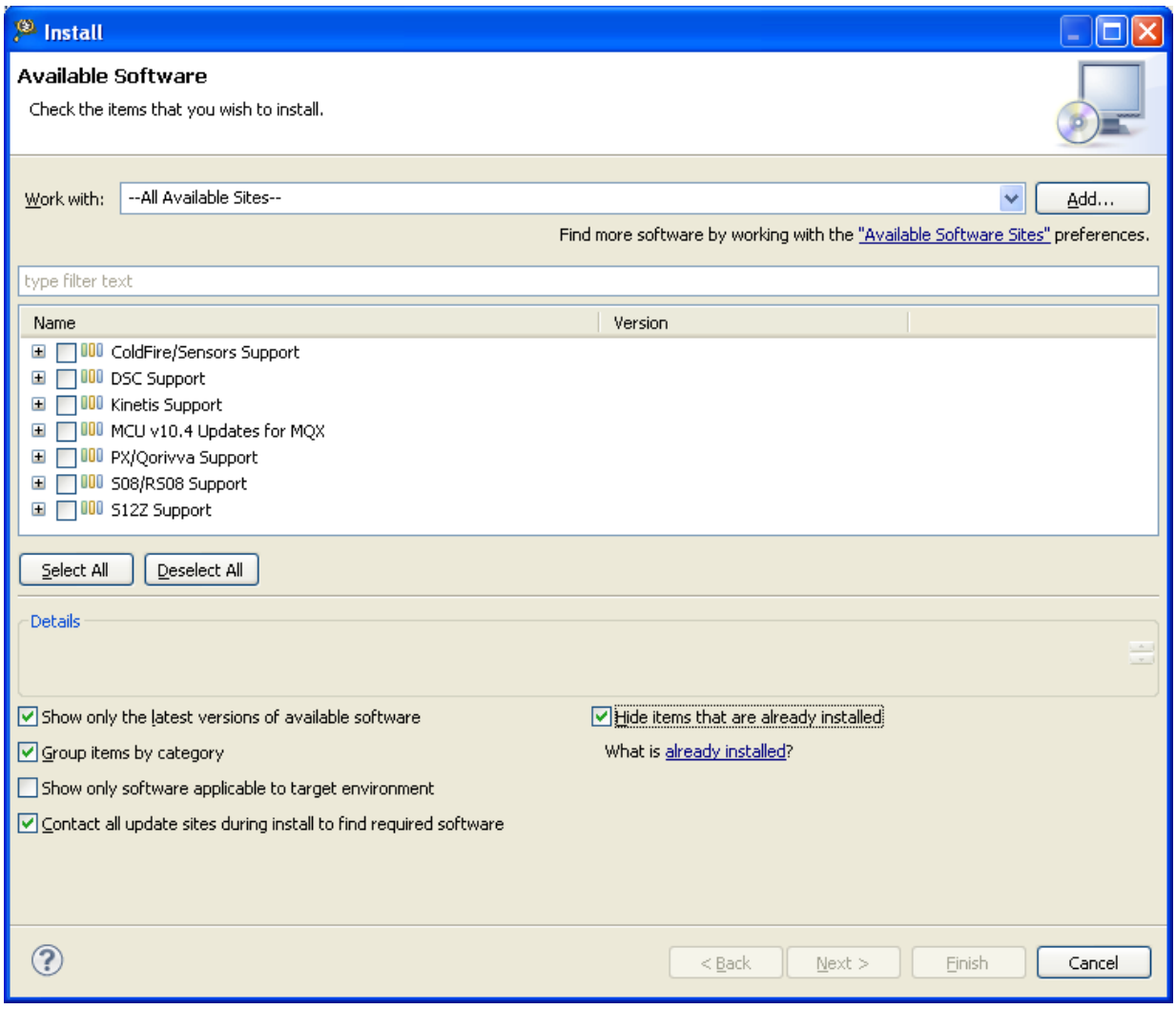
4. Clique em OK. Seu projeto foi apagado completamente.

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*.



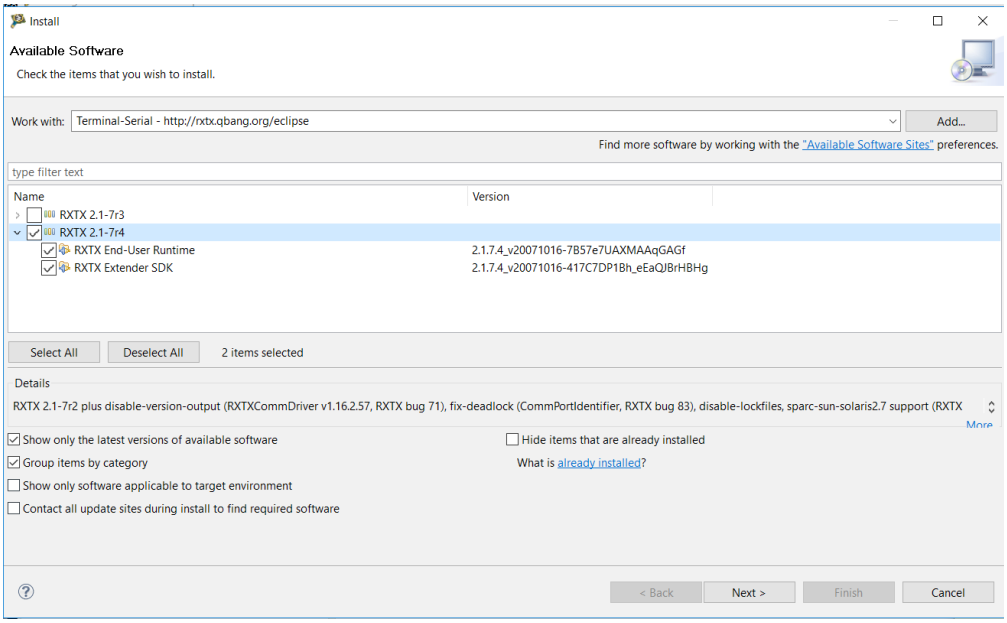
Na janela que se abre, podemos selecionar em "*Work With*" a opção "*All Available Sites*". O botão "*Add*" permite que se adicione pastas de arquivos ou websites que contenham outros *plugins*. No CW10, o padrão é a página de atualizações da *Freescale*. Pode-se marcar as opções desejadas. É interessante marcar a opção "*Hide items that are already installed*". Após esta seleção, clicando-se em "*Next*" segue-se 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*.



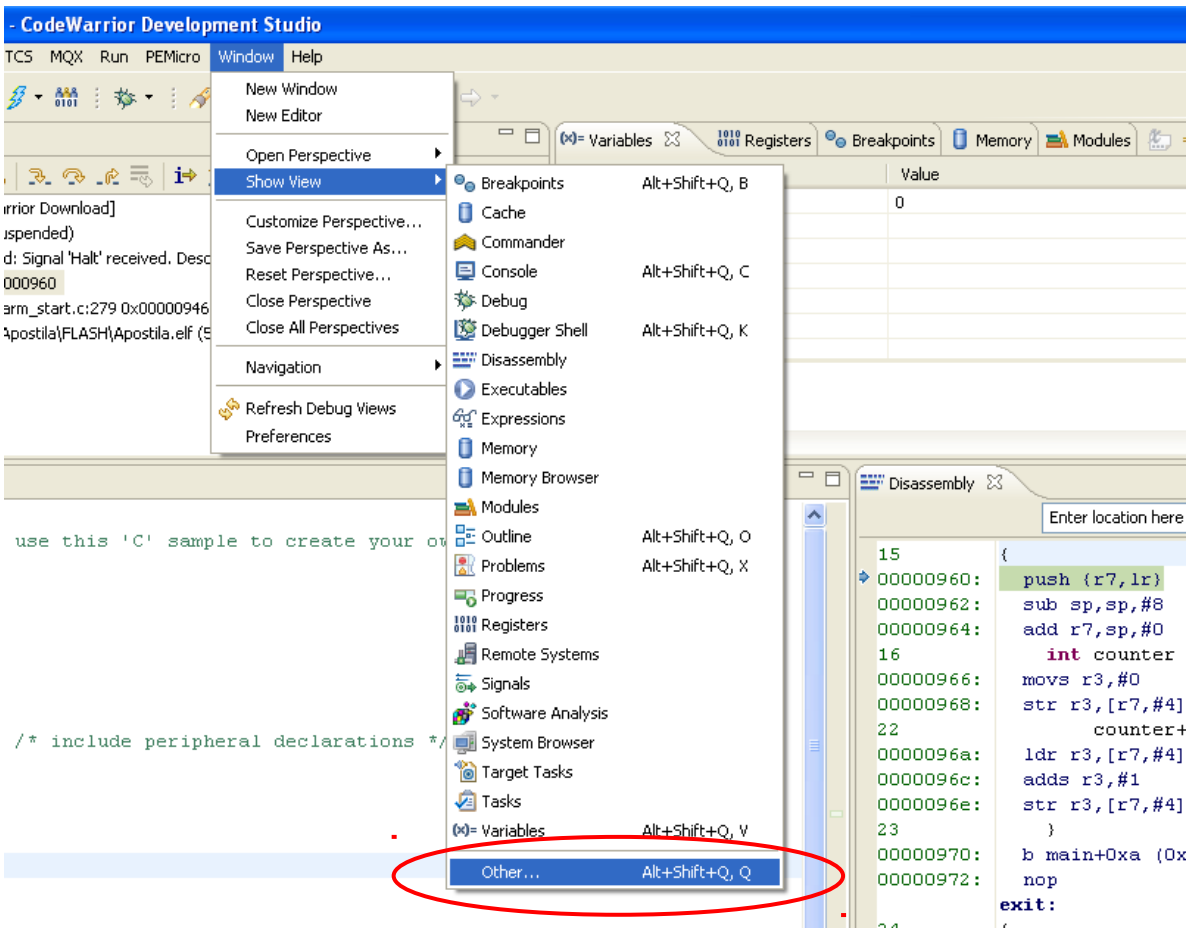
2.8.1 Terminal Serial

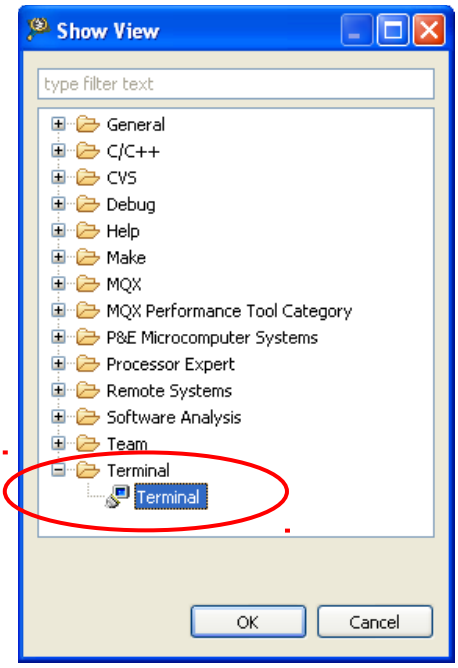
O aplicativo OpenSDA permite que um programa residente no microcontrolador se comunique com um computador via uma porta serial. E o aplicativo Terminal Serial é um *plugin* que provê uma interface de comunicação direta, via porta serial, entre o computador onde está instalado o CodeWarrior e o microcontrolador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

Uma alternativa é descompactar o arquivo indicado na referência [8] e seguir os passos de instalação de um novo *plugin* mostrado anteriormente. Na janela "*Install*", clique no botão "*Add"*. Selecione "*Local*" como a fonte do arquivo de instalação. Navegue até a pasta onde o arquivo de instalação foi desempacotada. Abra a árvore do TXRX, selecione APENAS o runtime (desmarque o SDK) e prossiga com a instalação padrão. Outra alternativa é utilizar o endereço da internet para instalação como mostra a seguinte figura: http://rxtx.qbang.org/eclipse.



Certifique se o terminal foi instalado corretamente, percorrendo o caminho "*Window > Show View > Other ...*". O item "*Terminal*" deve aparecer no sub-menu. Selecione o item e ao abrir uma janela *pop-up*, escolha o tipo de Terminal *"Serial"*. Uma nova aba *"Teminal"* deve ser aberta.





2.8.2 Doxygen

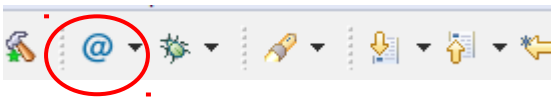
Documentar o seu código não só é uma forma de assegurar a propriedade intelectual como também facilita a sua manutenção ou atualização pelos terceiros. Doxygen+Graphviz+Mscgen são ferramentas de suporte à geração da documentação dos seus códigos diretamente a partir dos

comentários que você inseriu no seu programa. Eclox é um *plugin* que facilita a integração dessas ferramentas de documentação ao Eclipse, e consequentemente, ao ambiente IDE (*Integrated Development Environment*) CodeWarrior implementado sobre o Eclipse.

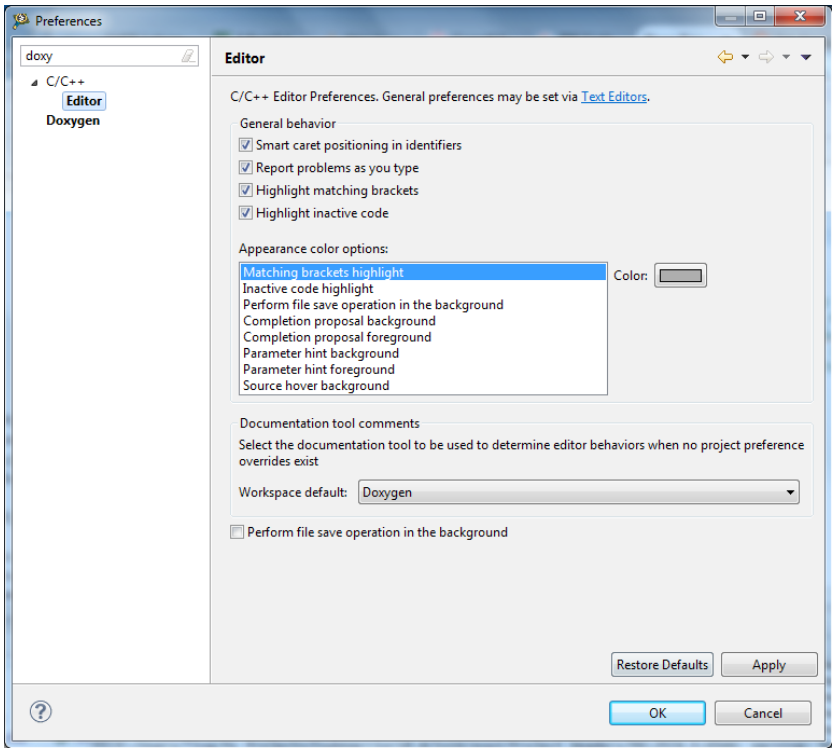
2.8.2.1 Instalação como *Plugin*

Erich Styger mostra detalhadamente na referência [9] como se instala estas ferramentas no Eclipse e como configurar o Eclipse após a instalação. Preste atenção de que somente Eclox-0.8 é compatível com a versão **Eclipse** em que o CodeWarrior foi desenvolvido. Outra observação é que não é necessário instalar Eclox Doxygen se você já tiver o Doxygen instalado.

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
 */
```

2.8.2.2 Documentação dos Códigos

Quando os comentários dos códigos forem precedidos pelos comandos especiais listados em [10], Doxygen organiza automaticamente os comentários conforme a semântica do comando precedido. Por exemplo, ao adicionar os seguintes comentários no escopo

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

pode ser gerado na documentação os seguintes textos na documentação:

Led piscante.

Returns

1 somente para satisfazer a sintaxe C

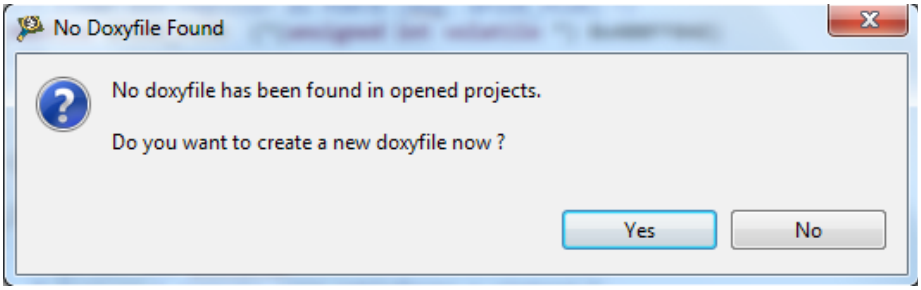
pois os comandos @brief e @return sinalizam Doxygen a gerar, respectivamente, uma breve descrição da função *main* e o valor retornado pela mesma função. Um exemplo de código comentado com os comandos especiais de Doxygen é mostrado na tela abaixo.

*main.c | __arm_start.c | __arm_eabi_init.c | exp2.doxyfile

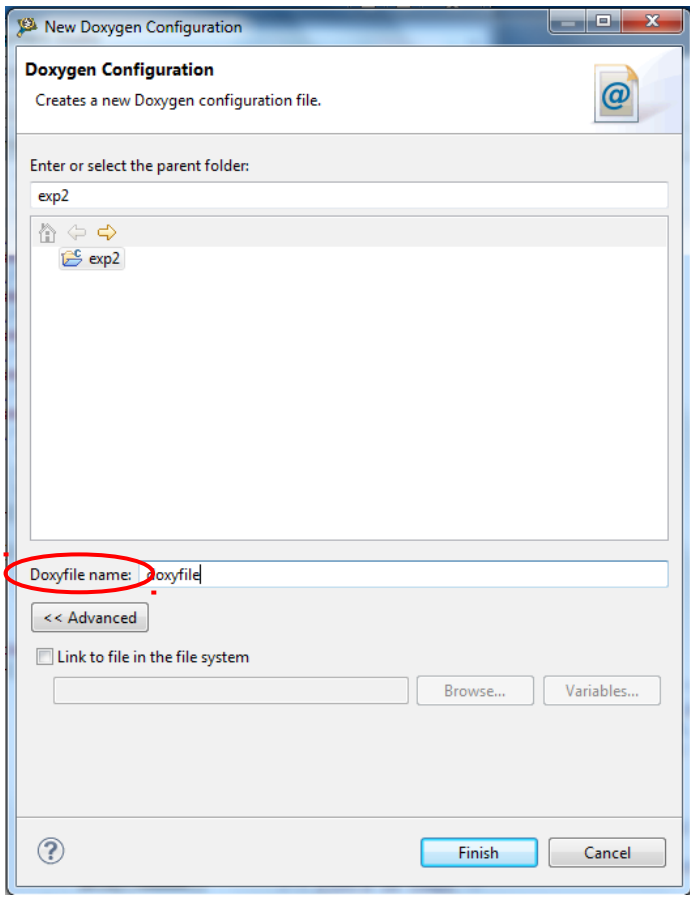
```
/*!
 * @file exp2.c
 * @brief Pisca-pisca de um led vermelho
 * @mainpage Projeto de Pisca-Pisca
 * Este &eacute; o primeiro projeto da disciplina EA871.
 * @authors Wu Shin-Ting and Eduardo Gavioli
 * @date 29/02/2016
 */
/*! Habilita as portas do GPIO (Reg. SIM_SCGC5) */
#define SIM_SCGC5    (*(unsigned int volatile *) 0x40048038)
/*! MUX de PTB18 (Reg. PORTB_PCR18) */
#define PORTB_PCR18 (*(unsigned int volatile *) 0x4004A048)
/*!  Data direction do PORTB (Reg. GPIOB_PDDR) */
#define GPIOB_PDDR  (*(unsigned int volatile *) 0x400FF054)
/*! Set bit register do PORTB (Reg. GPIOB_PSOR) */
#define GPIOB_PSOR  (*(unsigned int volatile *) 0x400FF044)
/*! Clear bit register do PORTB (Reg. GPIOB_PCOR) */
#define GPIOB_PCOR  (*(unsigned int volatile *) 0x400FF048)
/**
 * @brief gera um atraso correspondente a i iteracoes
 * @param i numero de iteracoes
 */void delay( unsigned int i)
{
    while (i) i--;
}
/*!
 * @brief Led piscante
 * @return 1 somente para satisfazer a sintaxe C
 */
int main( void)
{
    SIM_SCGC5 = SIM_SCGC5 | (1<<10);    /*!  Habilita clock GPIO do PORTB */
    PORTB_PCR18 = PORTB_PCR18 & 0xFFFFF8FF; /*! Zera bits 10, 9 e 8 (MUX) de PTB18 */
    PORTB_PCR18 = PORTB_PCR18 | 0x00000100; /*! Seta bit 8 do MUX de PTB18, assim os 3 bits de MUX ser?o 001 */
    GPIOB_PDDR  = GPIOB_PDDR  | (1<<18);    /*! Seta pino 18 do PORTB como sa?da */

    for(;;)
    {
```

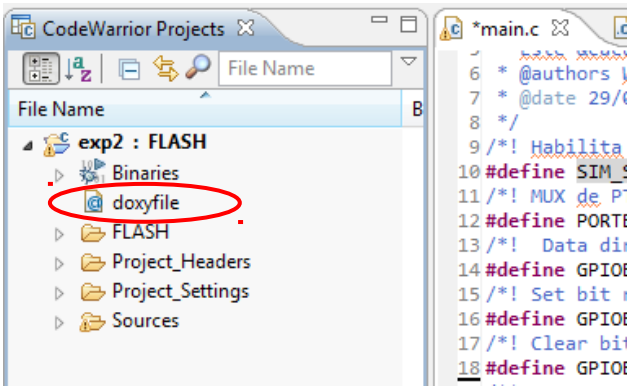
Doxygen é um gerador de documentação de programas bastante flexível. Aceita códigos-fonte em diferentes linguagens de alto nível e gera documentos em distintos formatos, como html, latex, rtf e man. Definimos o formato de uma documentação através de um arquivo de configuração. Se não existir este arquivo no projeto aberto, o Eclipse automaticamente ajuda o usuário a criar um quando se clica no botão "@"



Se clicar em “*Yes*”, aparece um novo *pop-up* menu solicitando o nome do arquivo de configuração que pode ser um novo ou um já existente. Na imagem abaixo, depois de ativado a pasta do projeto “exp2”, adicionou-se no campo “Doxyfile name” o nome de um novo arquivo “doxyfile” para o projeto ativado.



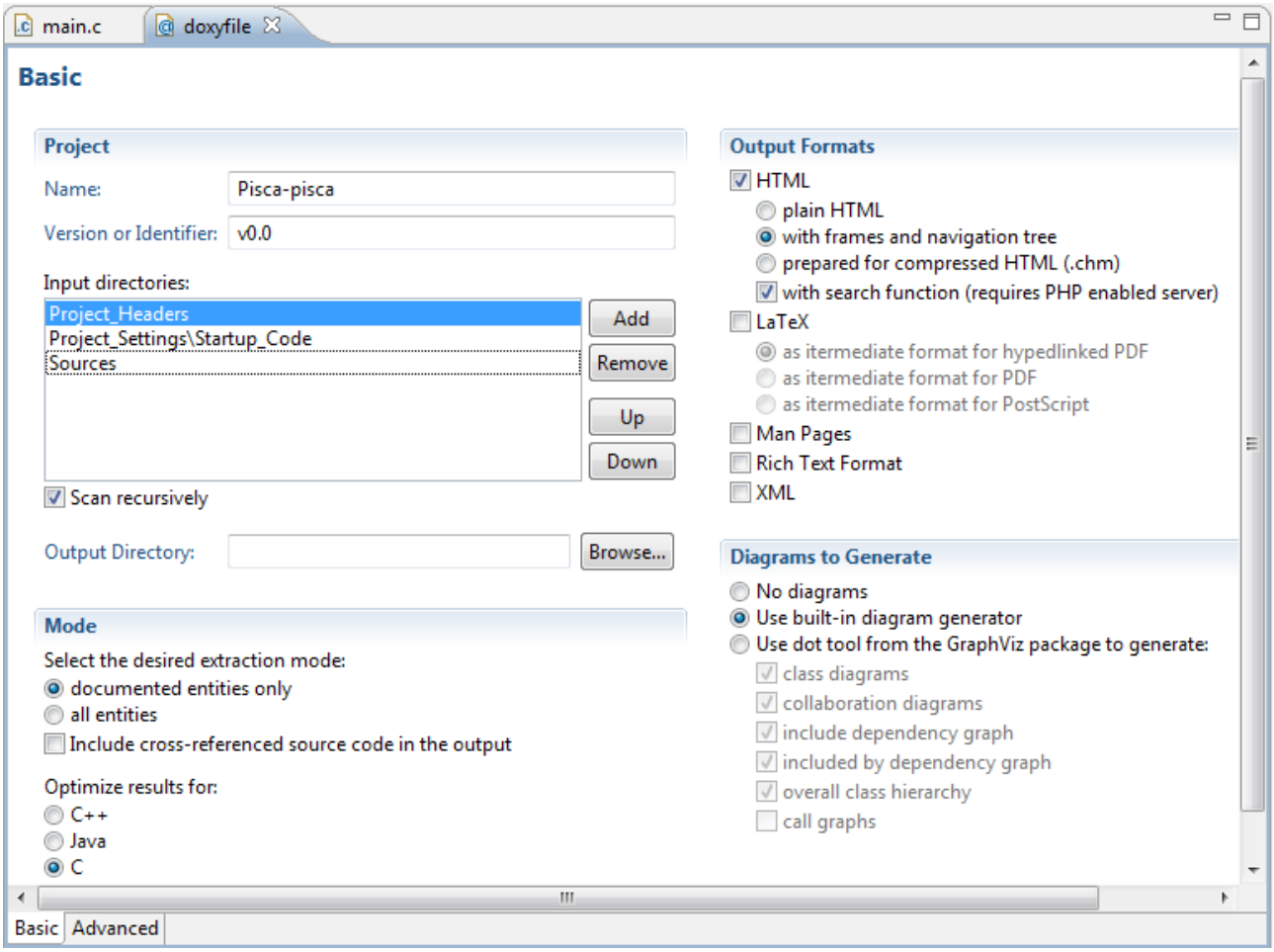
O novo arquivo criado aparece na árvore da janela **CodeWarrior Project** conforme a imagem abaixo



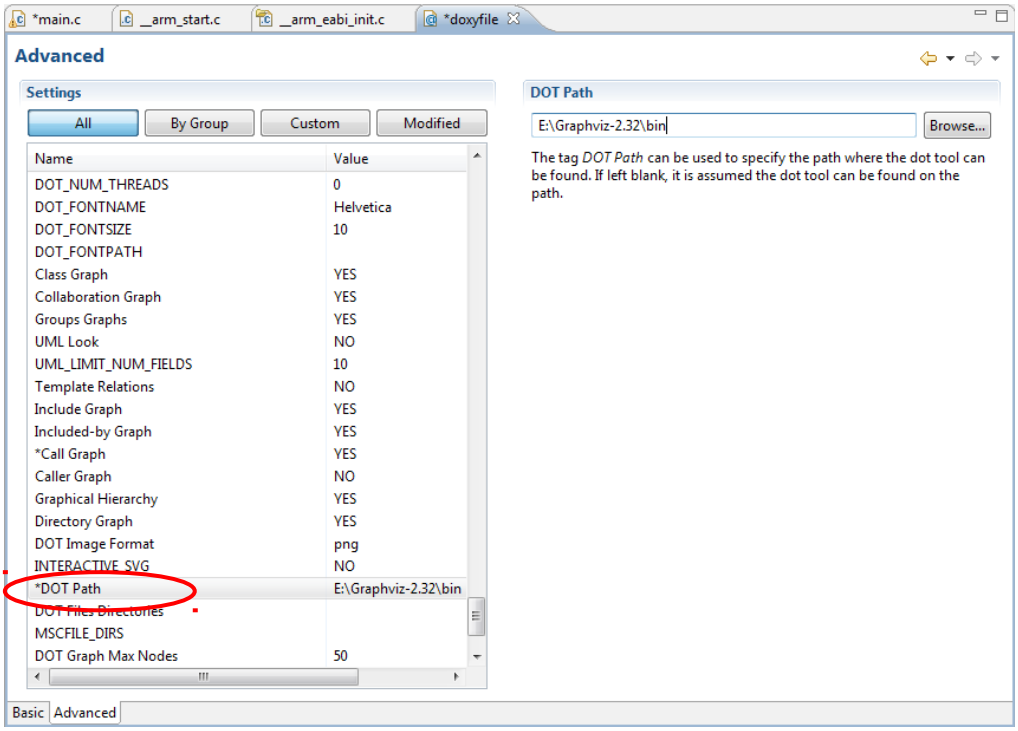
Ao clicar no nome do arquivo "*doxyfile*", Eclox mostra uma janela de editor permitindo configuração personalizada deste arquivo. Através desta janela, da versão "*Basic*", pode-se especificar o nome (campo "*Name*") e a versão (campo "*Version or Identifier*") do projeto, os diretórios que devem ser 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

*Advanced*de*Advanced* dependência e de classes (caixas de selação "*Diagrams to Generate*"). A janela abaixo mostra uma configuração para gerar documentação em html.

Observe que para os projetos desta disciplina é suficiente selecionar "C", fazer extração dos arquivos com comentários, gerar a documentação no formato html, incluir três pastas na área "*Input Directories*": *Project_Headers*, *Project_Settings\Startup_Code* e *Sources*. Como não foi especificado o diretório de saída dos arquivos de documentação, Doxygen usará o diretório corrente como o diretório de saída. Ou seja, as pastas dos arquivos da documentação geradas serão colocadas na pasta do projeto.

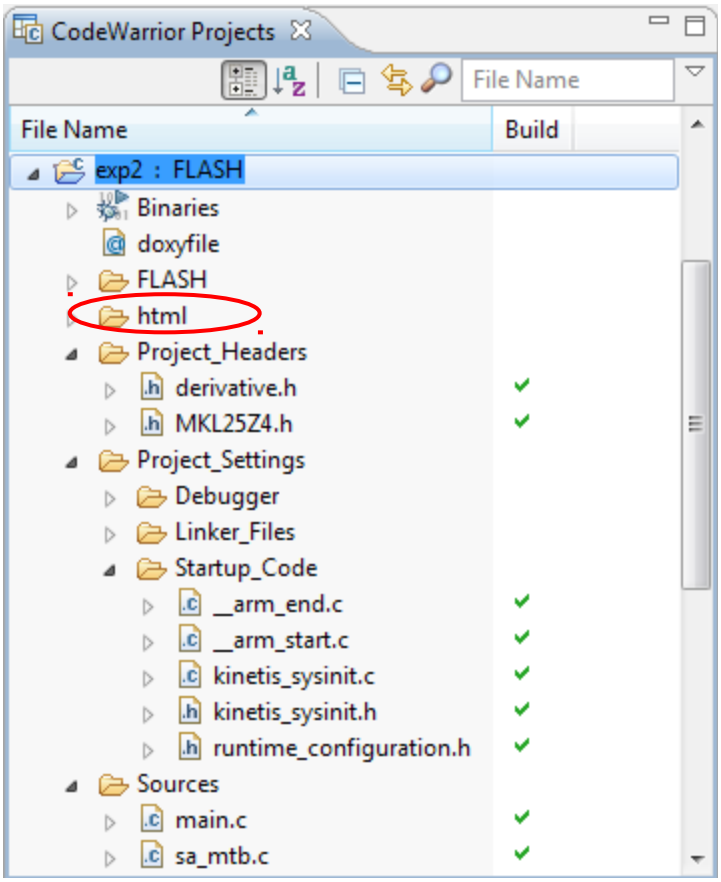


Observe que há uma versão "*Advanced*" do editor através do qual podemos, por exemplo, especificar o diretório onde se encontram os aplicativos Graphviz e Mscgen através dos campos "*DOT Path*" e "*mscgen Path*". O aplicativo Graphviz [16] é utilizado pelo Doxygen para gerar os grafos de dependência entre as funções e o aplicativo Mscgen [17] 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*". Na imagem abaixo é mostrada a configuração do caminho do aplicativo de Graphviz. É importante que o item "*Have DOT*" do *script* esteja marcado com "*YES*". Outra observação importante é que Doxygen não reconhece espaços brancos nos nomes das pastas. Instale os dois aplicativos em pastas com nomes que não tenham espaços brancos.

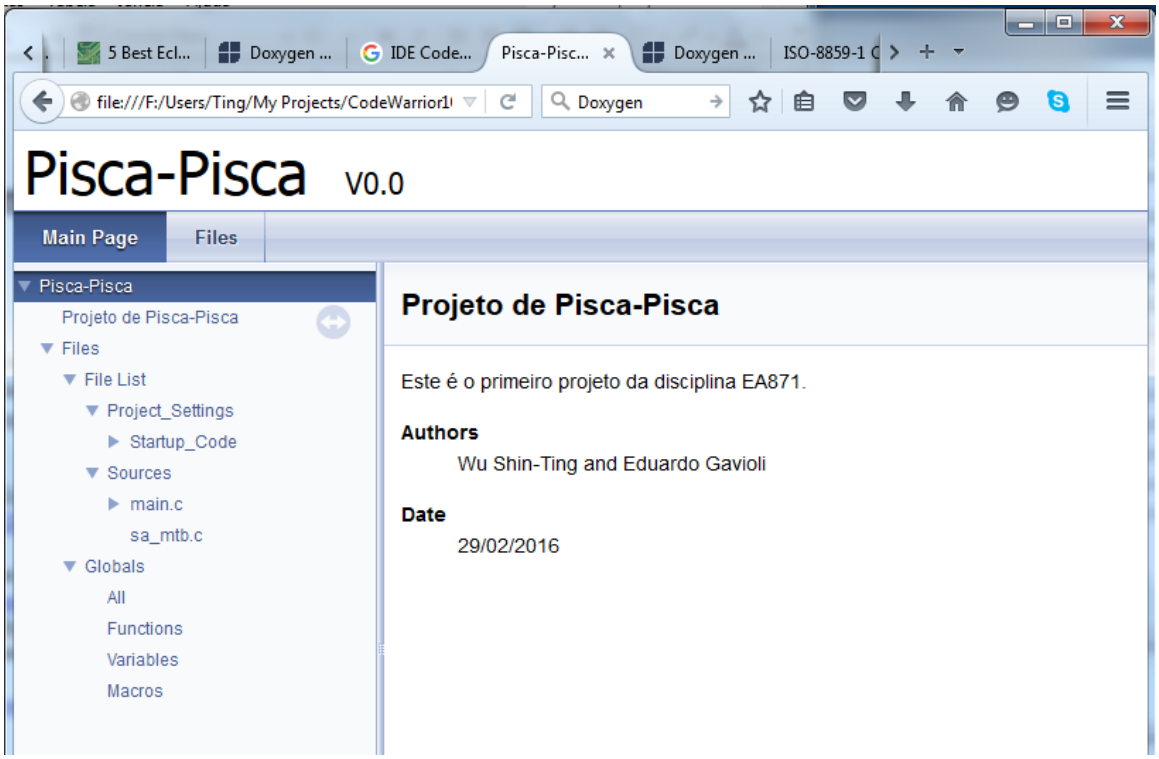


Para finalizar a configuração, basta clicar em “@” novamente. O arquivo doxyfile é automaticamente atualizado.

Uma vez configurado o arquivo, basta clicar novamente o 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 é necessário carregar o arquivo index.html dentro desta pasta num *browser* de html.

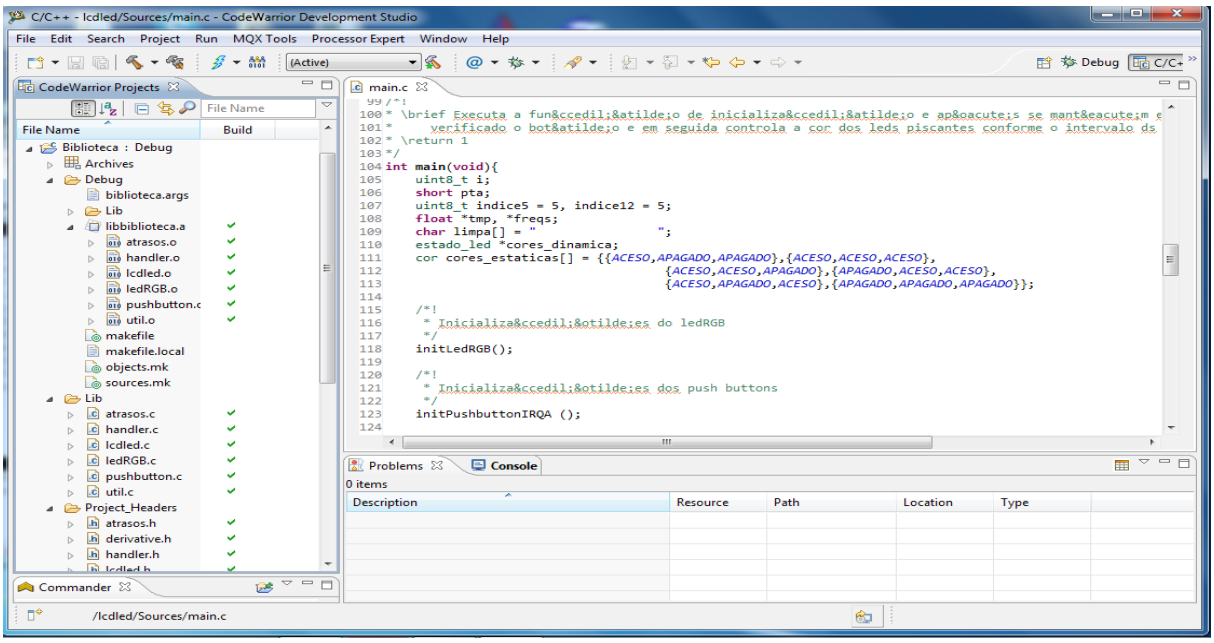


Vale comentar que Doxygen só trabalha com a codificação UTF-8. Se o *browser* de html que não a reconhece, o documenta não seria renderizado corretamente. Para contornar isso, os caracteres ISO-8859-1 foram digitados diretamente no formato html, p.ex. "é" no lugar de "é", nos códigos que distribuirei nesta disciplina [12]

2.9 Criação de uma Biblioteca de Rotinas

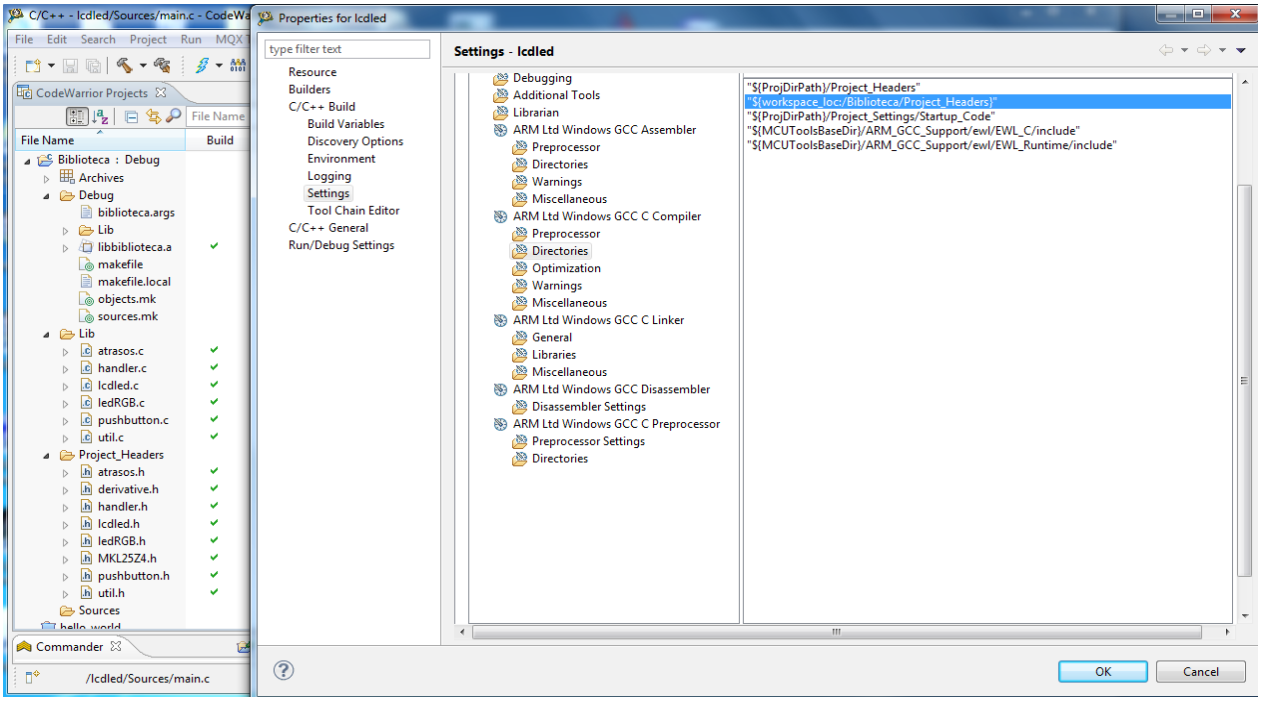
Para evitar duplicação de códigos e facilitar a sua manutenção, é recomendável pré-compilar rotinas de uso frequente nos diferente projetos, agrupá-las numa biblioteca e ligá-las com os programas específicos de um projeto somente na etapa de linkagem.

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.

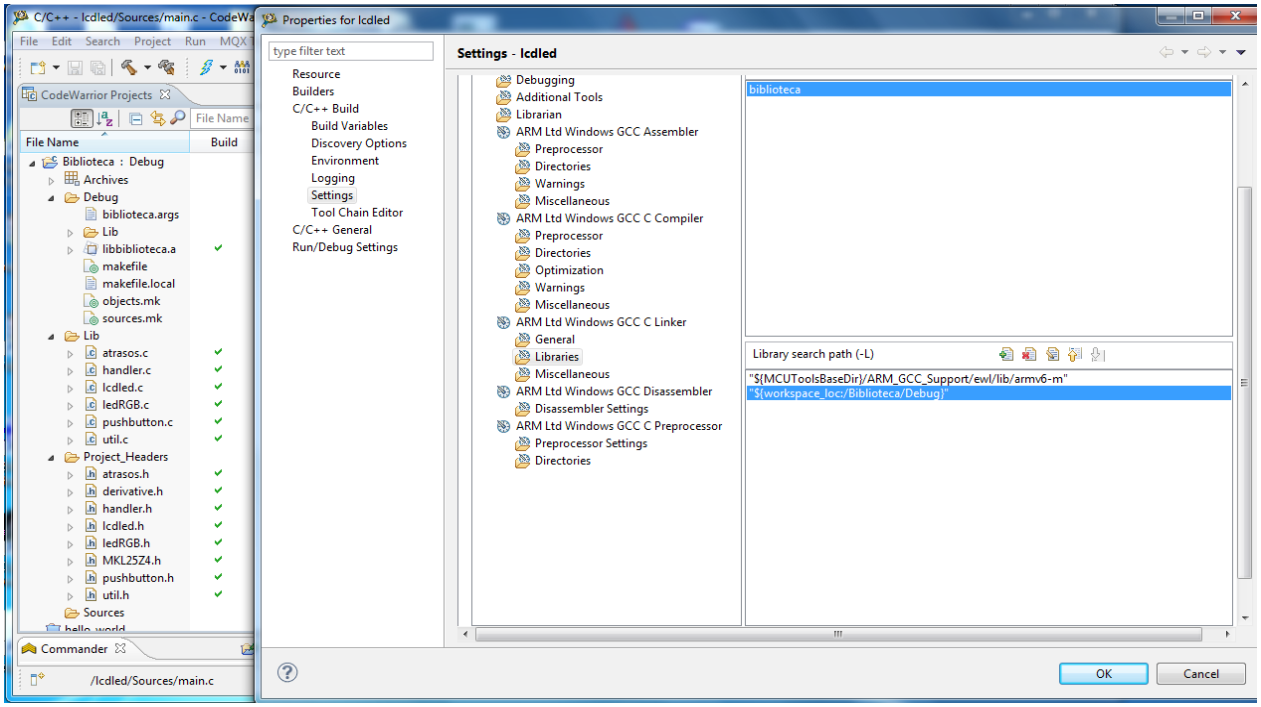


Como usar esta biblioteca num projeto-aplicação? Para isso é necessário que o projeto-aplicação possa resolver os protótipos das rotinas chamadas durante a fase de compilação e tenha acesso às instruções das rotinas invocadas durante a fase de ligação. Podemos configurar o ambiente de compilação e de 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.



Referências

[1] Freescale. Kinetis Peripheral Module Quick Reference

ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KLQRUG.pdf
[2] KL25 Sub-Family Reference Manual – Freescale Semiconductors (doc. Number KL25P80M48SF0RM), Setembro 2012.
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KL25P80M48SF0RM.pdf
[3] Freescale. CodeWarrior Development Suit – Eclipse Quick Reference Windows
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea076/complementos/CWMCUQRCARD.pdf
[4] ARM. ARMv6-M Architecture Reference Manual
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/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.
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/manuais/CWCFUG.pdf
[8] Arquivo de instalação do Terminal
http://rxtx.qbang.org/eclipse/downloads/RXTX-SDK-I20071016-1945.zip
[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://www.stack.nl/~dimitri/doxygen/manual/commands.html
[11] Dimitri van Heesch. Doxygen Manual.
https://www.stack.nl/~dimitri/doxygen/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] Eclipse CDT
http://www.eclipse.org/cdt/
[15] Eclipse CDT 8.8 Cheat Sheet
http://www.triada.si/download/eclipseCDT8.0-cheatsheet.pdf
[16] Graphviz – Graph Visualization Software
https://graphviz.gitlab.io/download/
[17] Msc-generator
https://sites.google.com/site/mscgen2393/

Agosto de 2016
Revisado em Fevereiro de 2017
Revisado em Fevereiro de 2018