# Tutorial

# Criação de projeto e setup no System Workbench for STM32

## 1. Introdução

No mundo dos sistemas embarcados, há muitas *IDE's*, cadeias de ferramentas (*toolchains*) e bibliotecas que estão prontas para uso com os microcontroladores da família STM32.

A *toolchain System Workbench for STM32*, também chamada de *SW4STM32*, é um ambiente de desenvolvimento integrado (*IDE*) *multi-OS* gratuito baseado na plataforma *Eclipse*, que suporta toda a gama de microcontroladores STM32 da *ST Microelectronics*, além de suas placas comerciais de desenvolvimento ou placas personalizadas criadas por qualquer usuário. A cadeia de ferramentas *SW4STM32* pode ser obtida no site *www.openstm32.org*, que inclui fóruns, blogs e treinamentos para suporte técnico.

A *toolchain System Workbench* e seu site colaborativo foram construídos pela *Ac6 Tools*, uma empresa de serviços francesa que fornece treinamento e consultoria em sistemas embarcados.

Uma vez gratuitamente cadastrados neste site, os usuários receberão instruções de instalação na página *Documentation>System Workbench* para continuar com o download gratuito da *toolchain* que recebe suporte oficial da *ST Microelectronics*.

As principais caracteríticas dessa *IDE* são:

- Suporte abrangente para microcontroladores STM32, placas Nucleo STM32, kits Discovery e placas de avaliação, além de firmware STM32 (*Standard Peripheral Library* ou STM32Cube HAL)
- Compilador GCC C/C++
- Debugger
- IDE Eclipse
- Compatível com plugins do Eclipse
- Suporte ao gravador/debugger ST-LINK
- Nenhum limite de tamanho de código
- Suporte a diferentes sistemas operacionais: Windows®, Linux e OS X®

Nos sistemas embarcados, normalmente, o software é projetado e desenvolvido para uma placa personalizada em um projeto ou produto específico, e não para uma placa de avaliação ou desenvolvimento, disponibilizada comercialmente. Nesse caso, é necessário saber como inicializar o hardware e personalizar um projeto para uso com uma placa customizada.

Este tutorial mostra como criar e usar um projeto no *System Workbench* para uma placa personalizada dotada de um microcontrolador específico da família STM32 da *ST Microelectronics*. O hardware do projeto não será pré-definido, isto é, não haverá arquivos de configuração com as definições de hardware da placa customizada, sendo necessário que o usuário implemente suas próprias definições.

As etapas a seguir mostram como criar um novo projeto no *System Workbench* para uma placa personalizada que utiliza o microcontrolador STM32F407VET6.

## 2. Criando um novo projeto no System Workbench

Para criar um novo projeto no *System Workbench,* selecione **File → New → C Project** no menu superior, conforme mostrado na figura 1, na página seguinte.

Na janela C Project, mostrada na figura 2, defina o nome do projeto no campo **Project name** (não utilize espaços em branco), além de selecionar opções tais como **Project Type** e **Toolchains**. Escolha a opção "Empty Project" para **Project Type**, de modo a criar um projeto vazio, e selecione a opção "Ac6 STM32 MCU GCC" para **Toolchains**, para fazer uso da toolchain da Ac6. Em seguida. clique em **Next** para continuar com a criação do projeto.

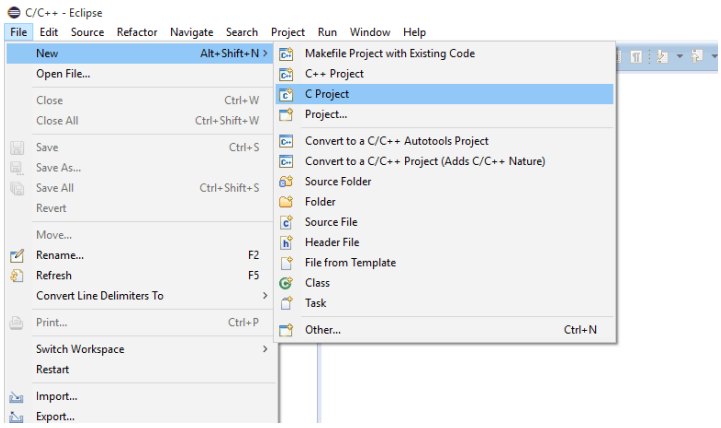


*Figura 1 – Criando um novo projeto no System Workbench.*

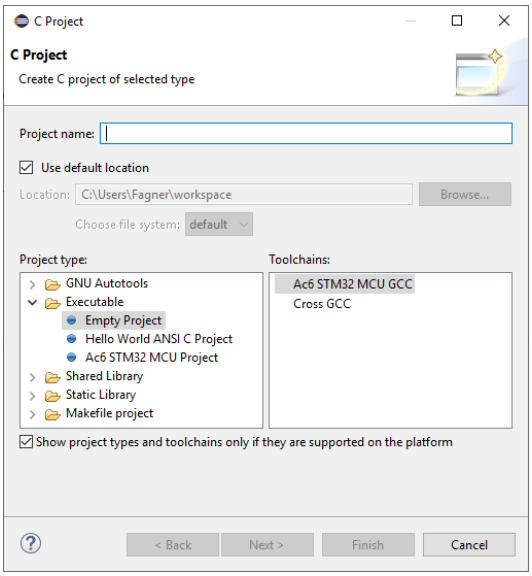


*Figura 2 – Janela de opções de projeto.*

Na janela seguinte, são marcadas as opções de configuração para o projeto. Normalmente, ambas as opções **Debug** e **Release** estarão marcadas, como visto na figura 3. Basta clicar em **Next** para continuar com a criação do projeto.

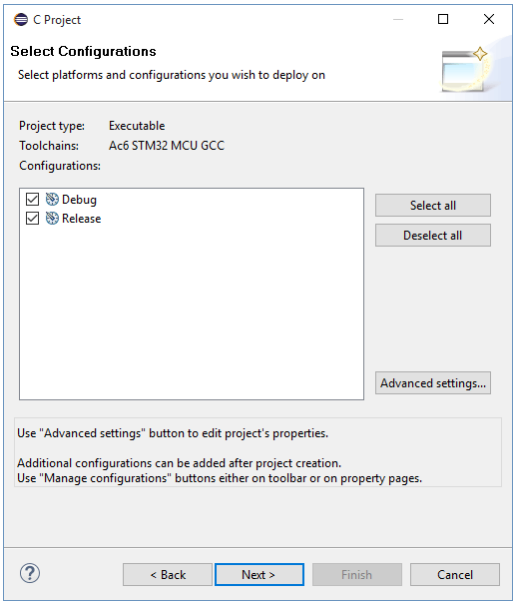


*Figura 3 – Janela de configurações do projeto.*

Na janela **Target Configuration**, mostrada na figura 4, é fornecida toda uma gama de opções que facilitam a seleção de um microcontrolador voltado para uma placa específica, seja NUCLEO, Discovery, EVAL ou até mesmo uma placa customizada. Para este caso, clique na aba Mcu, na opção *Series* selecione **STM32F4** e na opção *Mcu* selecione **STM32F407VETx**, destacando assim o microcontrolador empregado como base para o projeto a ser desenvolvido. Clique em **Next** para continuar com a criação do projeto.

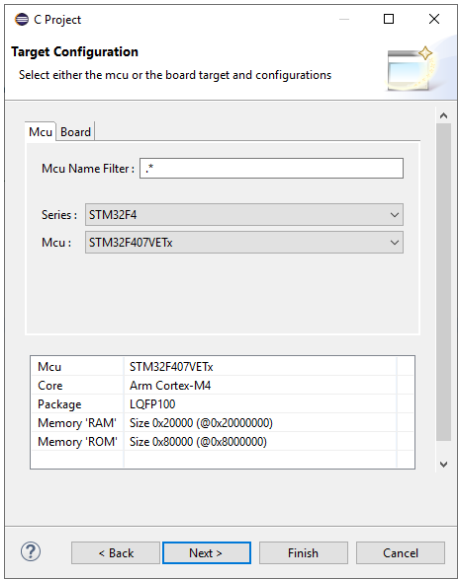


*Figura 4 – Janela de configurações do dispositivo.*

A próxima janela compreende o **Project Firmware configuration**, que é o ponto onde será escolhida uma base de código para o seu projeto, podendo ser o *Standard Peripheral Library*, que se fundamenta no CMSIS (*Cortex Microcontroller Software Interface Standard*), ou *Hardware Abstraction Layer* (*Cube HAL*), que é uma nova base para uso de código ARM desenvolvido pela *ST Microelectronics*. Todavia, caso não queiramos usar nenhuma das estruturas, também podemos marcar a opção **No firmware**.

Para este caso, marque a opção **Standard Peripheral Library**. Observe pela imagem mostrada na figura 5 que há um aviso em vermelho, no meio da janela, dizendo "*Target firmware has not been found, please download it*". Isso significa que o firmware do microcontrolador desejado não foi encontrado, e pede para você baixá-lo, o que pode ser facilmente feito ao clicar no botão **Download target firmware**. Clique neste botão para iniciar o download do **firmware**. Aguarde o término do download para clicar em **Finish**.

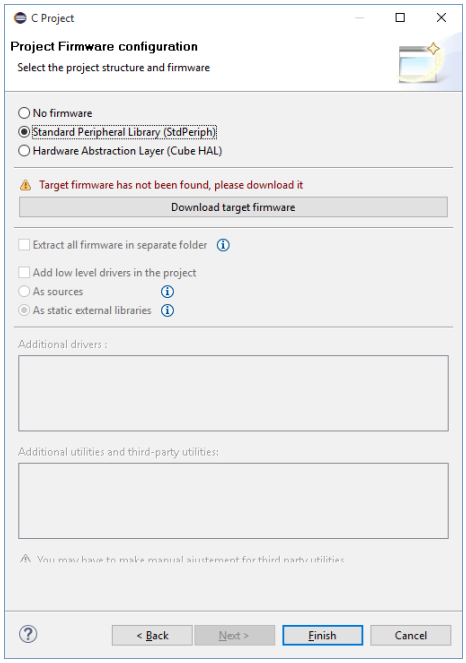


*Figura 5 – Janela de configurações do firmware.*

Terminado o download, agora irá aparecer um aviso de " *'STM32F4xx DSP StdPeriph Lib V1.8.0' has been found* ", mostrando que a ferramenta agora é capaz de localizar o firmware da opção selecionada, conforme mostrado na figura 6.

Agora, observe também que mais opções estão disponíveis na mesma janela. Deixe marcada a caixa de seleção **Add low level drivers in the project**, que adicionará os drivers para controle de GPIO, Clock, ADC, dentre outros periféricos, e marcada esta caixa ainda é possível escolher se os drivers serão adicionados como arquivos fontes (**As sources**) ou como bibliotecas estáticas externas (**As static external libraries**). Deixe marcada a opção **As sources**.

Finalizado todo o processo, clique em **Finish** para encerrar a configuração e gerar o projeto.

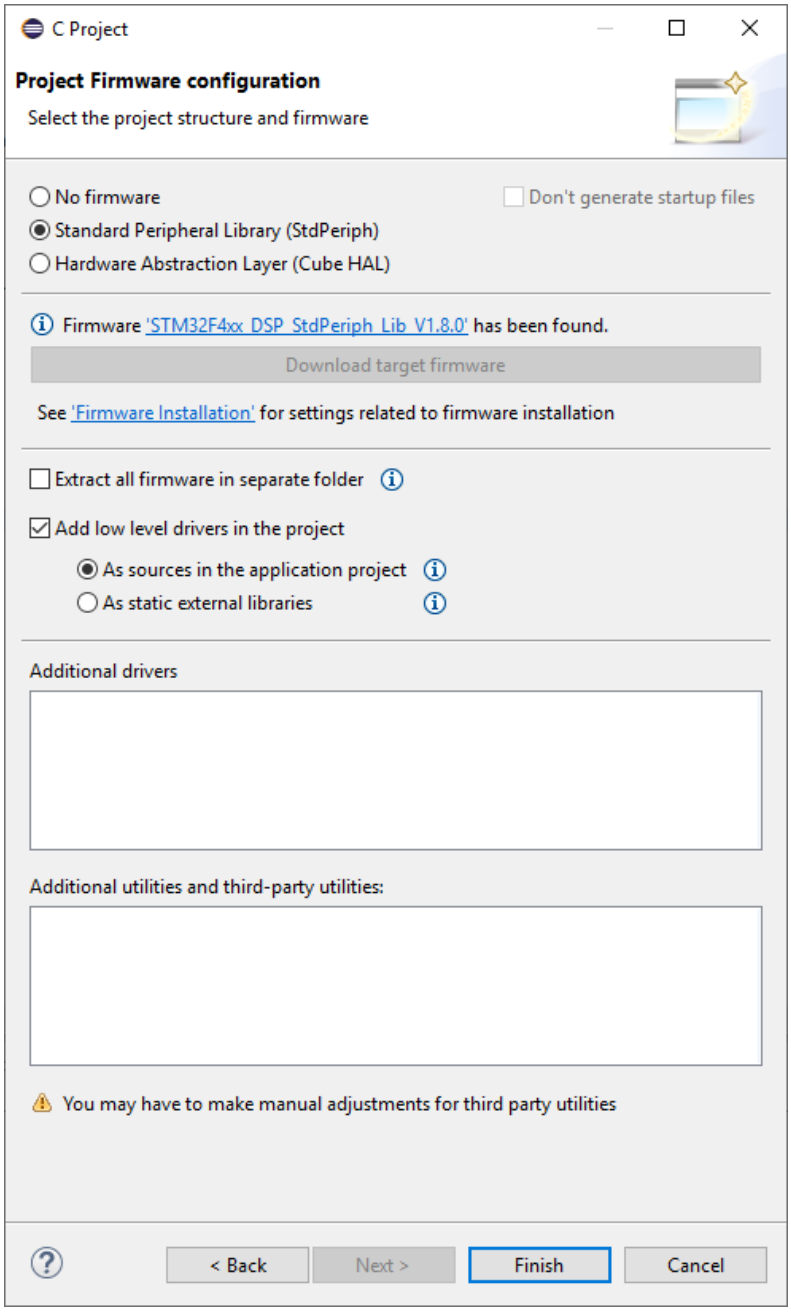


*Figura 6 – Janela de configurações do firmware com novas opções.*

Após clicar em Finish, a ferramenta irá carregar a estrutura do projeto e então passará a exibir uma perspectiva tal como mostrada na Figura 7. Caso isso não ocorra, clique na opção C/C++, localizada no canto superior direito da *IDE* para levar a visualização para perspectiva de edição de código C/C++.

Dentre toda a estrutura base criada para o novo projeto, o principal código é o famoso **main.c**, localizado no diretório **src** da raiz do projeto. Os demais diretórios correspondem a bibliotecas e drivers de suporte, conforme as seleções que foram feitas durante as etapaa de criação do projeto.

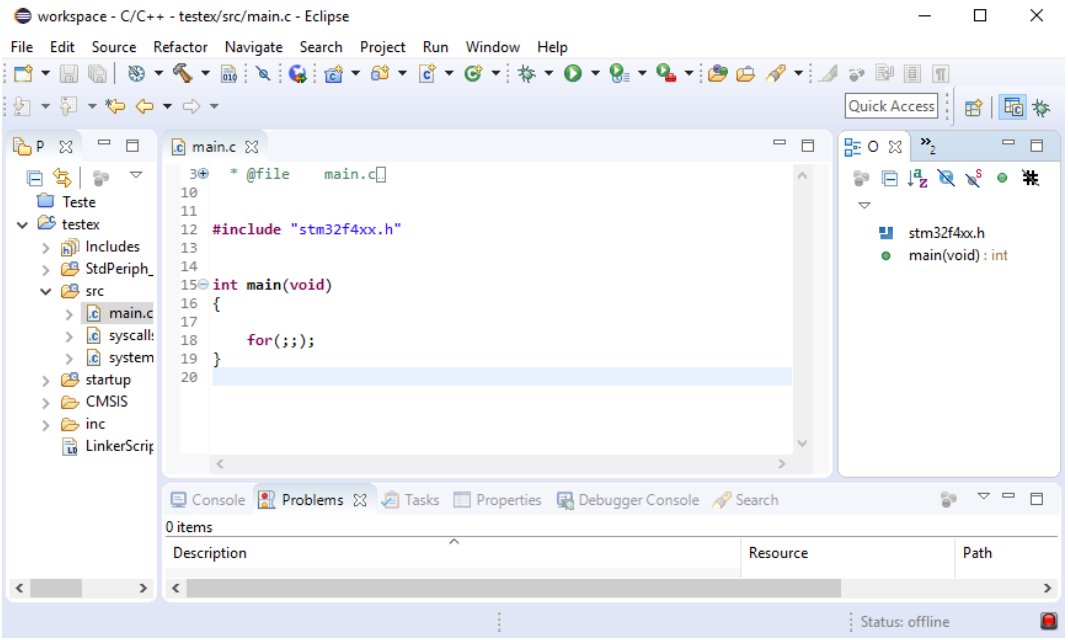


*Figura 7 – IDE com perspectiva C/C++ após a criação do projeto com código main.c aberto.*

## 3. Sequência de inicialização do STM32

Quando um microcontrolador STM32 é energizado, a função main não é executada imediatamente. Após o reset, uma sequência de inicialização é ativada e, em seguida, ocorre a execução de um bloco de código de startup. No final da seqüência de inicialização, a função main é finalmente chamada e executada.

Abaixo, são mostrados os 7 passos da sequência de inicialização que ocorrem antes da execução da função main:

1. Após a liberação do sinal de reset, os bytes de pré-configuração de hardware (option bytes) são carregados da flash para os registradores de alguns periféricos (como o memory protection unit (MPU), que indica se a memória pode ou não ser lida ou gravada por comandos de software enviados pelo dubugger por exemplo, reset voltage level, watchdog, reset em low power modes...);
2. Leitura dos pinos de boot do microcontrolador para selecionar a área de execuçao de código (flash, RAM ou system memory, onde está contido um bootloader);
3. Carregamento do endereço inicial da pilha no registrador R13 (Initial Stack Pointer Value) e carregamento do endereço inicial do código Reset_Handler no registrador R15 (Program Counter);
4. Execução do código Reset_Handler;
5. Execução do código SystemInit, dentro de Reset_Handler;
6. Inicialização na RAM das variáveis declaradas no programa do usuário;
7. Chamada da função main.

Os três primeiros passos são executados automaticamente pelo hardware, não havendo qualquer execução de código.

Após o reset, o microcontrolador STM32 roda na configuração padrão, isto é:

- Core configurado em single stack;
- Acesso completo a todos os registradores;
- Clock do sistema oriundo do oscilador interno de alta velocidade (16 MHz para o STM32F407);
- Clock conectado apenas à memória flash, RAM e o controlador de interrupções NVIC;
- Flash configurada para 0 wait states e maioria dos pinos configurados como entrada flutuante.

O bloco de código que é inicialmente executado ao ligar o MCU é colocado em um arquivo assembly de inicialização chamado **startup_stm32.s**, localizado no diretório **startup** da raiz do projeto. Este arquivo contém um código que prepara os periféricos básicos, como memória e sistema de clock, para permitir a execução do programa do usuário, e também a definição dos vetores de interrupção.

Esse arquivo de inicialização inclui uma seção que define a tabela de vetores de interrupção da forma mostrada na imagem seguinte.

```
.section .isr_vector,"a",%progbits

.word _estack
.word Reset_Handler
.word NMI_Handler
.word HardFault_Handler
.word MemManage_Handler
.word BusFault_Handler
.word UsageFault_Handler
.word 0
.word 0
.word 0
.word 0
.word SVC_Handler
.word DebugMon_Handler
.word 0
.word PendSV_Handler
.word SysTick_Handler

...
```

O primeiro código executado na inicialização é referenciado pela label **Reset_Handler**. O código de Reset_Handler inclui uma parte que prepara a memória (copia na RAM os valores iniciais das variáveis declaradas no programa do usuário) e, na sequência, chama as funções: SystemInit, __libc_init_array e, finalmente, a função main, conforme mostrado abaixo:

```
  .section .text.Reset_Handler
...
Reset_Handler:
....
  bl SystemInit            //Call the clock system initialization function
  bl __libc_init_array     //Call static constructors
  bl main                  //Call the application's entry point
```

- SystemInit é uma função do usuário que tem o papel de inicializar o sistema, configurando o clock do processador, fazendo algum eventual remapeamento da tabela de vetores de interrupção, e/ou fazendo a configuração de algum periférico. Ela é colocada no arquivo system_stm32f4xx.c, no diretório de arquivos fonte do projeto, **src**.
- __libc_init_array é uma função de biblioteca que inicializa todas as estruturas necessárias para a libc. Ela é colocada nos arquivos de origem da libc.
- main é a função principal do código do usuário.

No arquivo **system_stm32f4xx.c**, que se encontra no diretório de arquivos fonte do projeto, **src**, encontram-se as definições do sistema de clock do microcontrolador. É necessário realizar algumas alterações nesse arquivo nos parâmetros de configuração para que o microcontrolador opere no clock máximo, de acordo com o tipo de cristal utilizado na placa do usuário.

O clock máximo de operação do microcontrolador STM32F407VET é de 168 MHz. Dessa forma, as seguintes linhas, que estão devidamente enumeradas, precisam ser alteradas quando um cristal oscilador externo de 8 MHz for utilizado:

```
 ...
371     #define PLL_M      4
 ...
384     #define PLL_Q      7
 ...
401     #define PLL_N      168
 ...
403     #define PLL_P      2
 ...
```

## 4. Redirecionamento das funções printf e scanf para as USARTs

A linguagem C define duas funções que podem ser usadas para escrever e ler valores. Os valores são escritos em um chamado "fluxo de saída" que normalmente é direcionado para um dispositivo referido como tela ou console. Os valores são lidos de um chamado "fluxo de entrada" que normalmente corresponde a um dispositivo referido como teclado. As funções são **printf** e **scanf**, respectivamente, as quais são pré compiladas e fazem parte da biblioteca padrão de entrada e saída <stdio.h>.

Ao chamar as funções printf ou scanf no código do usuário, elas invocam duas funções auxiliares, chamadas **_write( )** e **_read( )**, que respectivamente recebem ou repassam uma string formatada. Essas duas funções, por sua vez, invocam as funções de baixo nível responsáveis pela manipulação do dispositivo de entrada e saída padrão do hardware, **__io_putchar** e **__io_getchar**, ambas declaradas com o atributo **weak** (fraca). Funções declaradas com esse atributo podem ser redefinidas pelo código do usuário para redirecionamento dos fluxos de entrada e saída de dados.

Normalmente, nos microcontroladores, os dispositivos padrão utilizados como saída e entrada de dados são as portas seriais de comunicação: as USARTs ou UARTs. Dessa forma, o redirecionamento do fluxo de dados das funções printf e scanf para as USARTs ou UARTs é suportado pela biblioteca <stdio.h>, que deve ser incluída no projeto por meio da diretiva **#include <stdio.h>**.

Durante a criação de um projeto no System Workbench, será incluído o arquivo **syscalls.c** que define as funções **_write( )** e **_read( )** para interagir sobre a string formatada. Essas funções chamam as funções de baixo nível **__io_putchar( )** e **__io_getchar( )**, respectivamente.

A rotina _write( ) itera através do buffer que é passado para ela por printf, retirando caracteres um a um, e os enviando para a rotina chamada __io_putchar( ) (existem dois sublinhados antes do nome). Do mesmo modo, a rotina _read( ) itera através do buffer que é passado para ela por scanf, retirando caracteres um a um, e os enviando para a rotina chamada __io_getchar( ) (existem dois sublinhados antes do nome).

As declarações das rotinas __io_putchar( ) e __io_getchar( ) no arquivo syscalls.c é vazia. Ao serem chamadas, elas retornam imediatamente. Porém, como dito anteriormente, são sub-rotinas "fracas", que dizem ao compilador/linker que estas rotinas devem ser usadas a menos que haja outras com o mesmo nome e que não sejam declaradas como fracas.

Assim, é necessário apenas fornecer as rotinas próprias de baixo nível _io_putchar( ) e _io_getchar( ) a fim de manipular a cadeia de caracteres no dispositivo de saída e entrada de dados desejado, como as USARTs ou UARTs, que devem ser definidas da seguinte forma:

```
int __io_putchar(int ch)
{
   //código para escrever o caractere 'ch' na UART
}

int __io_getchar(void)
{
   //código para ler um caractere da UART
}
```

Como exemplo, o código abaixo implementa as funções de baixo nível para escrita e leitura de dados na USART1 do microcontrolador STM32F407:

```
int __io_putchar(int ch)
{
   USART1->DR = (ch & (uint16_t)0x01FF);
   //espera para evitar a segunda transmissão antes da primeira ser concluída
   while(!(USART1->SR & USART_SR_TXE));
   return ch;
}
```

```
int __io_getchar(void)
{
   return (uint16_t)(USART1->DR & (uint16_t)0x01FF);
}
```

## 5. Suporte à impressão de floats com a função printf

Por padrão, a versão "nano" da biblioteca padrão de entrada-saída (stdio) é selecionada. Isso ajuda a manter pequeno o tamanho do código gerado no processo de compilação, mas não permite o uso de números em ponto flutuante no printf.

Para manter o código pequeno e ainda habilitar suporte à leitura e escrita de floats com as funções printf e scanf, adicione "**-specs=nano.specs -u _printf_float -u _scanf_float**" (sem as aspas) nas configurações do projeto: **Project → Properties → C/C++ Build → Settings → Tool Settings → MCU GCC Linker → Miscellaneous** no campo **Linker flags**, como mostrado na figura 8.

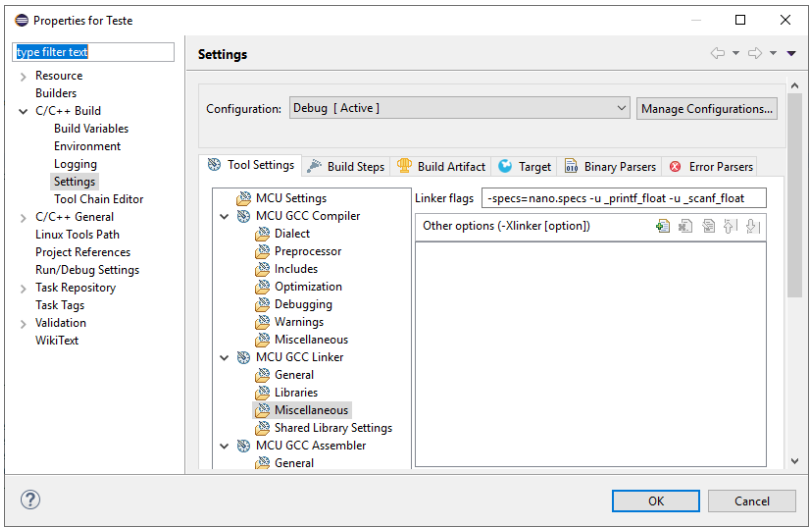


*Figura 8 – Propriedades do projeto para permitir suporte à impressão de floats usando printf.*

Uma janela do terminal serial para vizualização da string impressa por printf pode ser aberta a partir do System Workbench. Para isso, abra a visualização de conexões usando as opções no menu **Window → Show View → Other...**, abrindo a janela mostrada na figura 9.

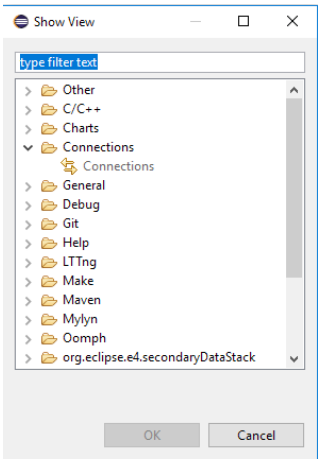


*Figura 9 – Janela Show View.*

Ao clicar duas vezes no ícone *Connections*, adicionará a janela de conexões na mesma seção da janela do console, como mostrado na figura 10.

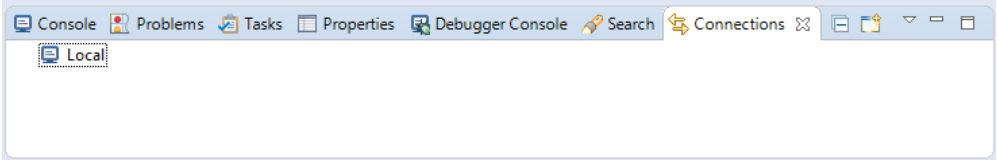


*Figura 10 – Janela de conexões.*

Para adicionar uma nova conexão, clique no ícone do botão New Connection no canto superior direito da janela de conexão e a janela de diálogo Tipo de Conexão, mostrada na figura 11, será exibida. Selecione a opção Serial Port (Porta serial) e clique em Next para abrir a caixa de diálogo New Serial Port Connection (Conexão de porta serial nova), mostrada na mesma figura, na qual a porta serial pode ser selecionada entre as que estão atualmente disponíveis. Atribua um nome adequado e selecione as propriedades da conexão como *baud rate*, quantidade de bits de dados, paridade e quantidade de bits de parada antes de clicar em *Finish*.

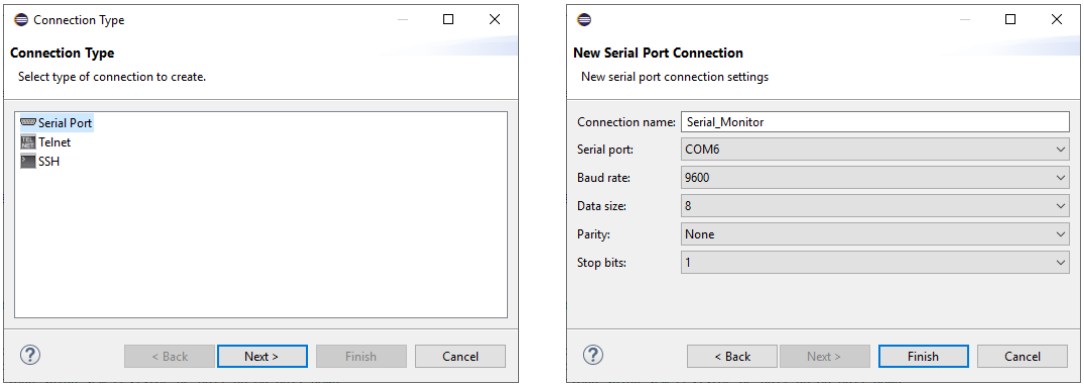


*Figura 11 – Janelas Connection Type e New Serial Port Connection.*

Na conclusão, uma nova conexão de porta serial será mostrada na janela de conexões com o nome atribuído anteriormente, como pode ser visto na figura 12.

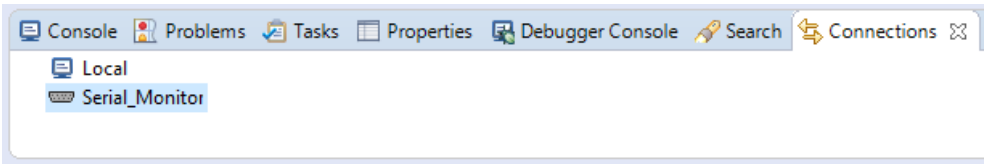


*Figura 12 – Janelas de conexões mostrando a conexão criada.*

Ao clicar com o botão direito do mouse na conexão da porta serial recém-adicionada, permitirá que você abra o Shell de Comando e conecte-se à porta COM.

## 6. Unidade aritmética de ponto flutuante

O processador ARM Cortex-M4 atende a aplicações que exigem uma combinação da funcionalidade tradicional de microcontroladores, bem como o processamento digital de sinais. Dispositivos que possuem ambas as características são frequentemente chamados de controladores digitais de sinais (*DSC*).

Um dos recursos opcionais que um núcleo Cortex-M4 pode incluir em seu design é uma poderosa unidade aritmética de ponto flutuante (FPU). Sendo este um recurso opcional, alguns microcontroladores com processador Cortex-M4 não possuem uma FPU. Se a FPU não estiver presente, a maioria das IDEs incluirá uma opção para emular o ponto flutuante usando operações inteiras em sua biblioteca *C Runtime*.

Os números de ponto flutuante podem ser de precisão simples (“float”) ou precisão dupla (“double”). A FPU no processador Cortex-M4 suporta operações de precisão simples, mas não duplas. Se o cálculo de precisão dupla estiver presente, o compilador usará funções da biblioteca C Runtime para manipular o cálculo via software.

Para um melhor desempenho, é melhor manipular o cálculo em precisão simples, quando possível. A maioria dos compiladores emulará a precisão dupla usando operações inteiras, e não operações de ponto flutuante de precisão simples.

A FPU fornece apenas algumas funções primitivas básicas (como add, sub, mul, div, sqrt) e várias outras funções de suporte. Ela não fornece resultados como sinf( ), sendo isto obtido por meio das bibliotecas matemáticas de software.

A precisão simples é suficiente para aplicações como:

- áudio, vídeo, sinais de sensores, controle e regulação, onde uma menor resolução seja aceitável;
- processamento digital de sinais com valores de ADC de 8 a 18 bits, FFT, IFFT, filtros FIR e IIR;

- impressão de valores reais com ordem de grandeza como pico, nano, ..., mega, giga, mas com até 6 dígitos de precisão;
- cálculos matemáticos onde o algoritmo pode aceitar a resolução mais baixa.

A precisão simples não é indicada para:

- navegação, gps, cartografia, levantamento topográfico;
- medição e cálculos com frequências;
- cálculo numérico em ciência e tecnologia - em matemática, física, astronomia, etc;
- calculadoras de bolso;
- sistemas CAD e simuladores;
- cálculo financeiro.

Pode haver casos em que você acidentalmente use um cálculo de precisão dupla sem que você saiba. Isso pode ser devido à expansão implícita dos tipos exigidos pelo padrão da linguagem C. Portanto, é útil verificar a saída do processo de compilação para ver se ele está chamando funções da biblioteca C Runtime. Uma maneira comum é gerar uma listagem assembly da imagem compilada e verificar o uso das instruções de ponto flutuante como **vadd**, ***vldr***, ***vmul*** e ***vstr***, ou durante uma sessão de debug, abrir uma janela de vizualização do código assembly em **Window → Show View → Disassembly**.

Os cálculos de ponto flutuante são realizados utilizando um banco de registradores em separado, dentro da unidade de ponto flutuante. Se tanto a thread principal (por exemplo, programa principal) quanto as rotinas de serviço de interrupção (ISR) usam a FPU, o salvamento do contexto extra e a restauração são necessários para garantir que a ISR não corrompa os dados usados pela thread principal. O salvamento e a restauração de contexto extra exigem ciclos de clock extras e, portanto, se você quiser ter um tempo de resposta rápido da ISR, uma maneira é evitar cálculos de ponto flutuante dentro de uma ISR. Desta forma, o empilhamento e desempilhamento dos registradores da FPU é evitado.

Quando as operações de ponto flutuante são realizadas no modo thread e ocorre uma interrupção, um recurso de hardware chamado *Lazy Stacking* reserva um espaço para os registradores da FPU na pilha para que eles possam ser colocados na pilha mais tarde, se necessário. Portanto, você precisa verificar a alocação do tamanho da pilha para garantir que haja espaço suficiente para acomodar esses dados.

Os arquivos de cabeçalho CMSIS fornecem uma abstração do núcleo Cortex-M4. Observe que os arquivos de cabeçalho CMSIS-Core usam duas macros da linguagem C:

- **__FPU_PRESENT**: definido se uma FPU estiver presente no chip;
- **__FPU_USED**: Definido se a FPU for usada na aplicação.

Quando essas macros são definidas, a função de inicialização do sistema **SystemInit( )** habilita a FPU. A FPU deve ser ativada antes que qualquer instrução FPU seja executada, caso contrário, uma exceção de hardware será gerada.

Para habilitar a FPU, modifique as configurações em **Project → Properties → C/C++ Build → Settings → Tool Settings → MCU GCC Compiler → Preprocessor** no campo **Defined Symbols (-D)** adicione as duas macros, como mostrado na figura 13.

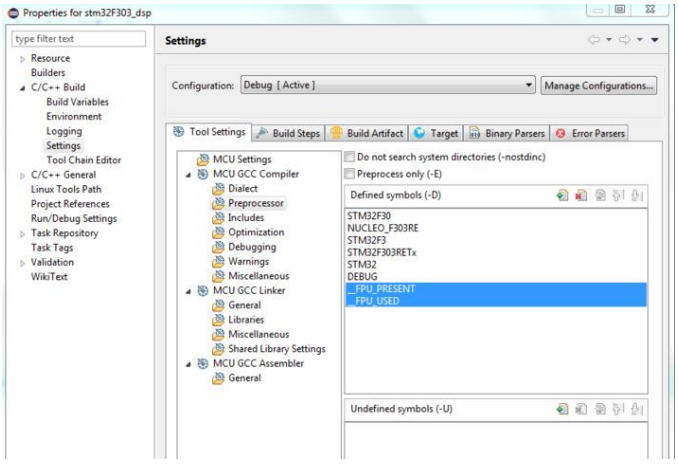


*Figura 13 – Inclusão das macros __FPU_PRESENT e __FPU_USED para habilitação da unidade aritmética de ponto flutuante.*

Em seguida, em **Project → Properties → C/C++ Build → Settings → Tool Settings → MCU Settings**, no campo **Floatinf Point ABI** escolha **hard**.

Se uma aplicação não precisar lidar com cálculos de ponto flutuante, a FPU poderá permanecer desligada o tempo todo, reduzindo o consumo de energia. Para isso, basta não definir a macro __FPU_USED.

Em algumas aplicações, você pode precisar usar a FPU apenas por um curto período de tempo e, em seguida, pode desativá-la novamente quando as operações da FPU forem concluídas. No entanto, neste caso, você deve reativar a FPU para poder utilizà-la novamente.

Mesmo que haja uma FPU no microcontrolador, alguns cálculos de ponto flutuante (por exemplo as funções trigonométricas) ainda precisam ser manipulados por funções da biblioteca C Runtime. Nesses casos, parâmetros e resultados devem ser passados entre o código do programa e as funções dessa biblioteca.

Esteja ciente de que há duas opções na Application Binary Interface (ABI) para a arquitetura ARM, ou seja, ABI hard e ABI soft. Na ABI hard, os valores são passados através dos registradores da FPU, e na ABI soft são passados através de registradores da unidade de ponto fixo.

As operações de ponto flutuante apresentam problemas numéricos (como arredondamentos), e elas podem levar a problemas de desempenho, como no seguinte exemplo:

```
float X,Y;
X = Y*3.5;
```

Nesse exemplo, o valor de **Y** seria convertido para double para realizar a multiplicação com a constante **3.5** (uma constante de ponto flutuante é definida como double na linguagem C). Posteriormente, o valor seria reconvertido para float para armazenamento na variável **X**. Isso faria com que a FPU do Cortex-M4 não fosse utilizada, uma vez que ela é de precisão simples, sendo o cálculo realizado pelas funções da biblioteca Runtime C, levando um número muito maior de ciclos de clock para ser executado, reduzindo o desempenho.

Uma solução seria marcar as constantes de ponto flutuante com precisão simples, forçando o cálculo a ser executado com precisão simples e, assim, a FPU seria utilizada, como no exemplo:

```
float X,Y;
X = Y*3.5f;
```

Alguns compiladores podem ter uma opção para forçar todas as operações de ponto flutuante para precisão simples, ou gerar mensagens de notificação quando o cálculo de precisão dupla for usado.

Para forçar a utilização de constantes de ponto flutuante com precisão simples no Cortex-M4, pode-se utilizar a opção **-fsingle-precision-constant**, nas configurações do projeto: **Project → Properties → C/C++ Build → Settings → Tool Settings → MCU GCC Linker → Miscellaneous** no campo **Linker flags,** fazendo com que as constantes de ponto flutuante de dupla precisão sejam arredondadas para float de precisão simples. Isso evita a promoção de variáveis de precisão simples para double. Observe que isso também força o uso de constantes de precisão simples em operações e variáveis de precisão dupla. Isso pode melhorar o desempenho devido ao menor tráfego de memória, mas reduz a precisão.

## 7. Utilização do gravador ST-LINK

O ST-LINK, mostrado na figura 14, é um hardware programador/gravador e depurador integrado para os microcontroladores STM32. As interfaces de programação/depuração de fio único JTAG e SWD (*Single Wire Debugging*) são usadas para se comunicar com qualquer microcontrolador STM32 localizado em uma placa de aplicação ou de desenvolvimento.

*Figura 14 – Versão genérica do ST-LINK.*

O ST-LIK usa a interface USB para se comunicar com o ambiente de desenvolvimento integrado e com o hardware de depuração embarcado nos microcontroladores STM32. Dessa forma, ele funciona como uma ponte/link entre o ambiente integrado de desenvolvimento e o microcontrolador.

A comunicação permite a gravação de novos programas na memória interna do STM32, a leitura de programas previamente gravados, além da depuração. Durante a depuração, o programador pode ter acesso a todos os registradores internos do microcontrolador, tanto os do processador ARM como também de todos os periféricos, valores de variáveis e conteúdos de posições de memória. Permite ainda a execução passo a passo de um programa, tanto em Assembly quanto em Linguagem C.

Para criar uma sessão de depuração/programação do microcontrolador, clique em **Run → Run Configurations...** e verifique se já foi automaticamente criada uma sessão de debug com o nome dado ao projeto, conforme o ícone em destque mostrado na figura 15.

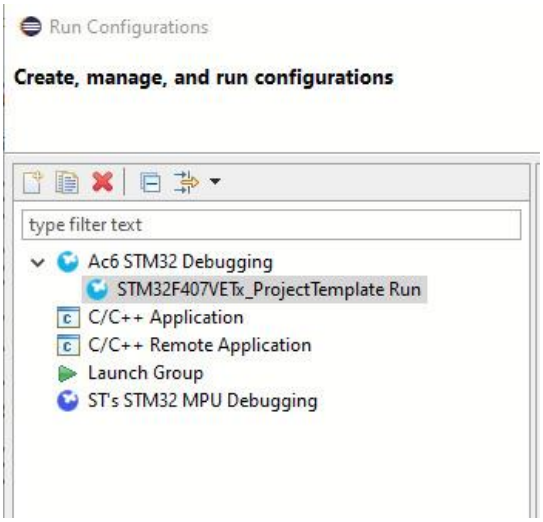


*Figura 15 – Configuração de uma sessão de depuração no System Workbench for STM32.*

Se abaixo do ícone **Ac6 STM32Debugging** não tiver nenhum ícone com o nome do projeto, dê dois cliques rápidos no primeiro ícone para que seja criada uma nova sessão de Debug.

Uma vez que a sessão foi criada, vá até a aba Debugger e clique em *"Show Generator Options..."*, e no campo *"Mode Setup"* selecione *"Software System Reset", conforme mostrado na figura 16.*

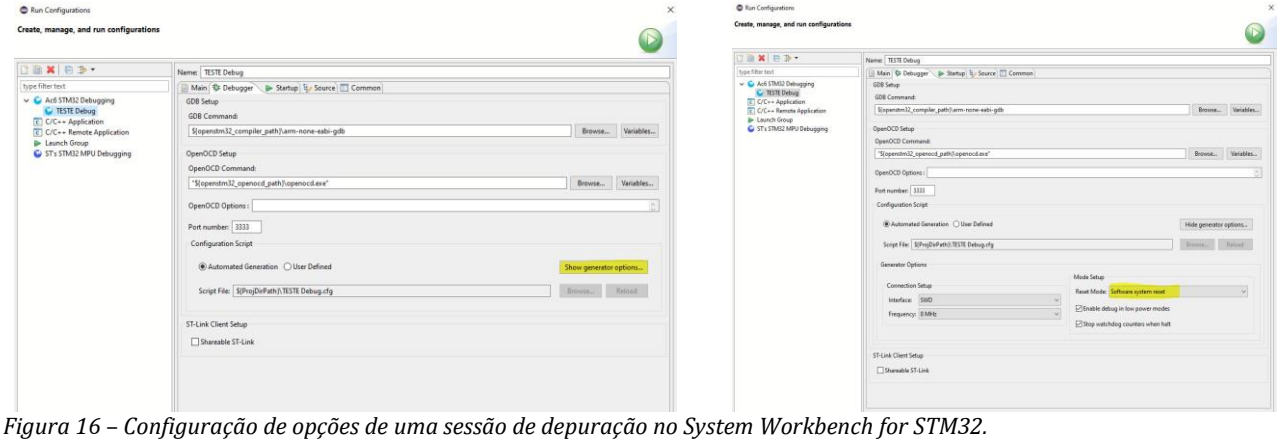

*Figura 16 – Configuração de opções de uma sessão de depuração no System Workbench for STM32.*

Para finalizar a criação da sessão, clique em *Apply*.

## 8. Geração de arquivo .hex para simulação com o Proteus

Durante a compilação, o System Workbench gera o arquivo executável que vai para a memória do microcontrolador. O gravador usa o arquivo executável com extensão .elf e é somente esse arquivo executável que o system workbench gera como saída. Porém, o Proteus não usa esse arquivo pra fazer a simulação, ele espera um arquivo .hex. Nesse caso, basta configurar o system workbench pra gerar, também, o .hex. Isso é feito da seguinte forma:

Vá em **Project → properties → C/C++ Build → Settings**. Na aba **Build Steps** existe um campo chamado **post-build steps**. Você precisa adicionar o seguinte comando:

**arm-none-eabi-objcopy -O ihex "${BuildArtifactFileBaseName}.elf" "${BuildArtifactFileBaseName}.hex"**

O campo deve ficar com a aparência mostrada na figura 17.

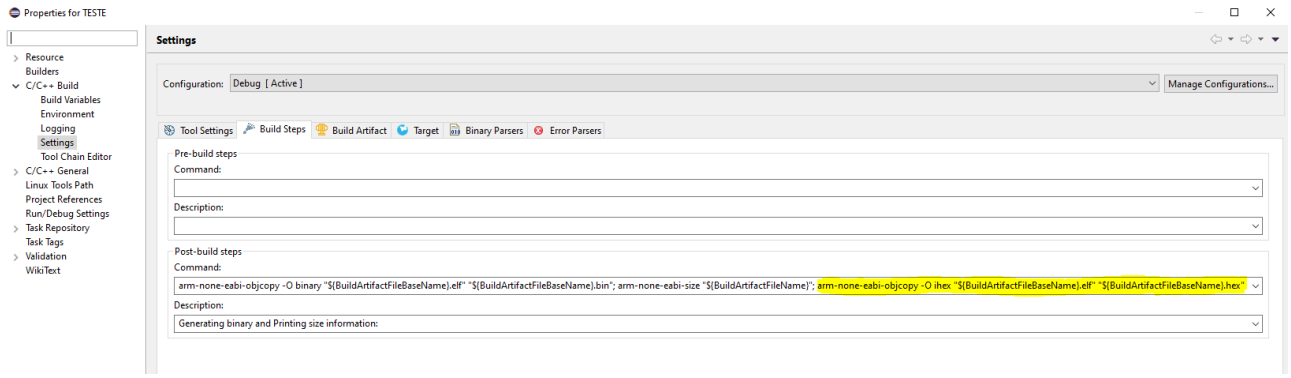


*Figura 17 – Configuração de opções para geração de arquivos .hex.*

Depois clique em *Apply* e *OK*.