Esse é o quinto artigo da série escrita pelo engenheiro Ismael Lopes da Silva, exclusivamente para o site "www.embarcados.com.br". Nessa série focarei no Microcontrolador da STMicroelectronics, o MCU STM32F103C8T6, que é um ARM Cortex-M3. Os pré-requisitos para uma boa compreensão dos artigos é ter o domínio da Linguagem C Embedded e conceitos de eletrônica.

Modos Operacionais do Processador ARM Cortex-M3

O Processador ARM Cortex-M3 opera em dois modos diferentes, que são:

✔ Modo Thread

✔ Modo Handler

No contexto dos microprocessadores, o modo Thread representa uma ordem de execução, com instruções encadeadas que são desempenhadas uma por vez. O código normal da aplicação sempre é executado em modo Thread. Também conhecido como "Modo de Usuário" ou "User Mode". Todas as exceções ou interrupções de nossa aplicação rodam em modo Handler.

Outro ponto importante é que o processador sempre inicia, por padrão, em modo Thread. Também sempre que processador encontrar uma exceção do sistema ou qualquer interrupção externa, ele muda seu modo para Handler, para atender à Rotina de Serviço de Interrupção (ISR), associada a essa exceção do sistema ou interrupção.

Em modo Thread há níveis de privilégio de acesso, portanto, dependendo do privilégio podem haver restrições a acesso a Registradores e outros recursos. No mode Handler o privilégio é total, sem restrições. Quando o processador inicia em modo Thread, o nível de privilégio é total. Veremos níveis de privilégios em outro artigo.

Modos de Operação na Prática

Usando o STM32CubeIDE, no mesmo workspace que criamos no artigo 3, então, vamos copiar o projeto "01HelloWorld" como "02ModoProcessador". Na janela "Projetc Explorer", clique com o botão direito do mouse sobre o projeto "01HelloWorld" e selecione "Copy". Novamente na janela "Projetc Explorer", clique com o botão direito do mouse sobre o projeto "01HelloWorld" e selecione "Paste". Uma janela para renomear a aplicação será mostrada, portanto, entre como o nome "02ModoProcessador". Depois clique no botão [Copy].

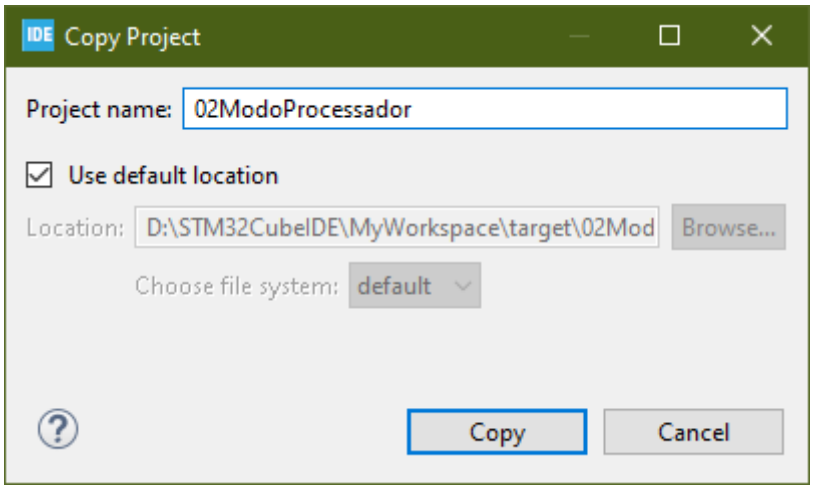

Figura 1 – Copiar e colar uma aplicação

Fizemos uma cópia porque tudo que preparamos é mantido, então, vamos apenas editar o arquivo main.c e outro que for necessário. Apenas temos que refazer a configuração do depurador, para habilitar o SWV (Serial Wire Viewer), já detalhado em outro artigo.

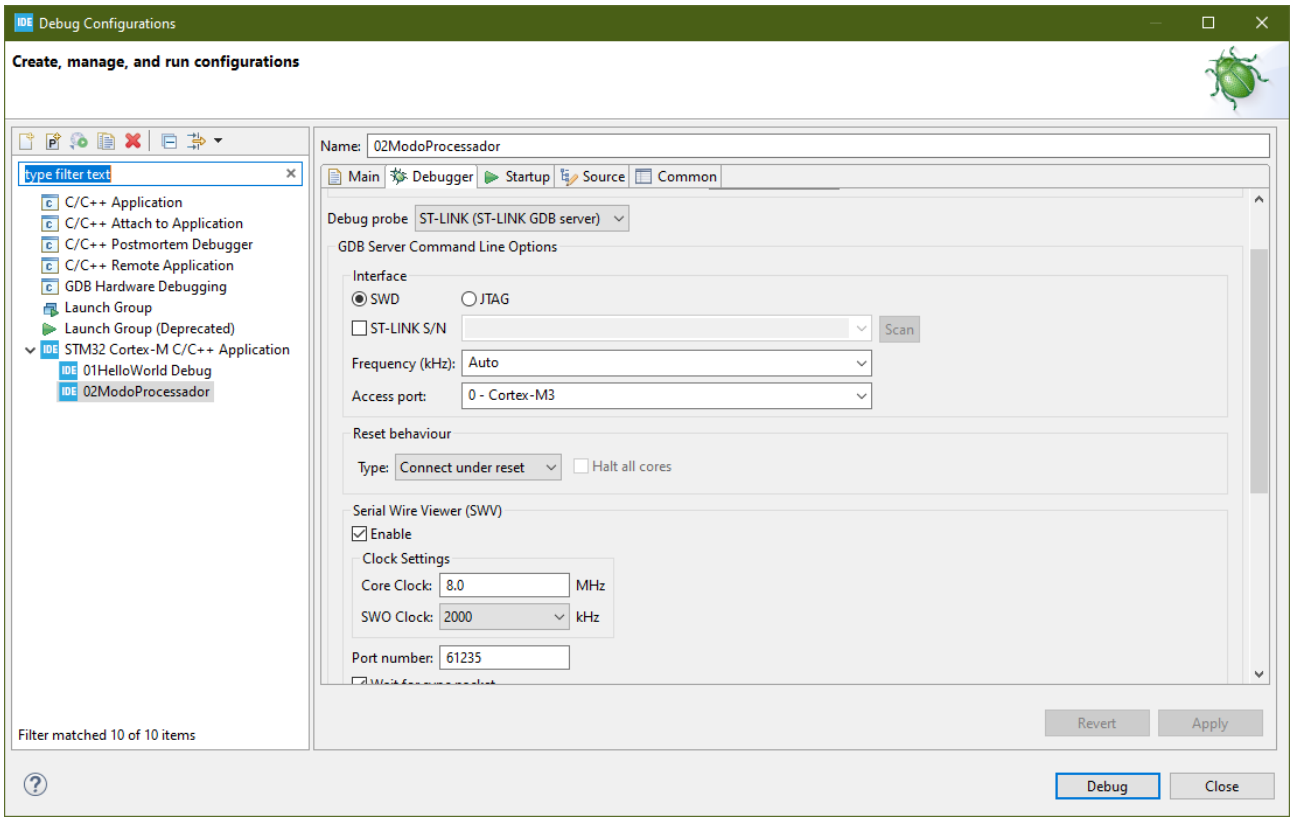

Figura 2 – Reconfigurando o Debugger habilitando SWV

Como nessa aplicação continuamos usando a função "printf", então, o processo de depuração precisa captar as mensagens da aplicação, então, temos que manter o uso da janela "SWV ITM Data Console". Para mais detalhes vejam os arquivos anteriores.

Segue o arquivo main.c editado para mostrar os modos de operação do processador. Os comentários nas linhas do programa também auxiliam no entendimento do código.

```
/******************************************************************************
 * @file           : main.c
 * @author         : Auto-generated by STM32CubeIDE
 * @brief          : Main program body
 ******************************************************************************
 * @attention
 *
 * <h2><center>&copy; Copyright (c) 2019 STMicroelectronics.
 * All rights reserved.</center></h2>
 *
 * This software component is licensed by ST under BSD 3-Clause license,
 * the "License"; You may not use this file except in compliance with the
 * License. You may obtain a copy of the License at:
 *                        opensource.org/licenses/BSD-3-Clause
 *
 ******************************************************************************/
```

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

#include<stdio.h>
#include<stdint.h>

// Função executada com processador em modo Thread
// Dentro dela é gerada uma interrupção que leva ao modo Handler
void gerar_interrupcao()
{
	// Cria ponteiro para o Registrador NVIC_ISER0 - NVIC Interrupt set-enable registers
	// Cria ponteiro para o Registrador NVIC_STIR - NVIC Software trigger interrupt register
	uint32_t *pNVIC_ISER0 = (uint32_t*)0xE000E100;
	uint32_t *pNVIC_STIR  = (uint32_t*)0xE000EF00;

	// Habilita interrupção IRQ3
	*pNVIC_ISER0 |= (1 << 3);

	//Gera uma interrupção de software IRQ3
	*pNVIC_STIR = (3 & 0x1FF);
}

/* Função executada com processador em modo Thread */
int main(void)
{
	printf("Em modo Thread. Antes da Interrupcao\n");
	gerar_interrupcao();
	printf("Em modo Thread. Depois da Interrupcao\n");
	for(;;);
}

/* Função executada com processador em modo Handler */
void RTC_IRQHandler(void)
{
	printf("Em modo Handler. Tratando a Rotina de Servico de Interrupcao\n");
}
```

Não vou repetir ações que já detalhei em artigos anteriores, portanto, vou somente citar o que deve ser realizado. Obviamente somente novas situações serão detalhadas.

Como temos uma nova aplicação “02ModoProcessador”, novo arquivo main.c, então, vamos dar um “Clean Project”, “Build Project” e ative a perspectiva de depuração. Quando iniciar a depuração (debugging), habilite janela “SWV ITM Data Console”, ative a porta 0, para capturar as mensagens enviadas pelas funções “printf”.

ATENÇÃO! No repositório de recursos dessa série eu adicionei o seguinte documento que vamos utilizá-lo (https://github.com/IsmaelLopesSilva/IoT/blob/master/STM32/STM32F103C8T6/ST

%20Docs/STM32F10xxx%20Programming%20Manual%20(PM0056%20-%20DocID15491%20Rev%206).pdf).

Nessa aplicação não exploraremos seu código, porque apenas quero demonstrar os modos de operação do processador ARM Cortex-M3, mas, é uma aplicação simples e seus comentários auxiliam no entendimento prévio, porém, o foco é monitorar um Registrador especial que nos mostrará o modo de operação em cada trecho da aplicação.

Onde Monitorar o Modo de Operação do Processador ARM Cortex-M3?

No documento, ARM Generic User Guide Cortex-M3 (DUI0552A), disponibilizado no repositório do github dessa série, na página 19, encontraremos detalhes sobre o Registrador IPSR (Interrupt Program Status Register). Os nove bits iniciais do IPSR [8:0] compõe o campo chamado ISR_NUMBER (Número da Rotina do Serviço de Interrupção), que são:

- ✔ 0 = Thread mode;
- ✔ 1 = Reserved;
- ✔ 2 = NMI;
- ✔ 3 = HardFault;
- ✔ 4 = MemManage;
- ✔ 5 = BusFault;
- ✔ 6 = UsageFault;
- ✔ 7-10 = Reserved;
- ✔ 11 = SVCall
- ✔ ;12 = Reserved for Debug;
- ✔ 13 = Reserved;
- ✔ 14 = PendSV;
- ✔ 15 = SysTick;
- ✔ 16 = IRQ0;
- ✔ .
- ✔ .
- ✔ .
- ✔ n+15 = IRQ(n-1)ª – O número de interrupções, n, pode ser de 1 à 240.

Se o processador estiver em modo Thread o valor do ISR_NUMBER = 0. Qualquer valor maior do que zero o processador estará em modo Handler. Cada valor do ISR_NUMBER > 0, representa um tipo de exceção ou interrupção, que correntemente está sendo tratada.

Monitorar o Modo de Operação do Processador na Prática

Com a aplicação está sendo depurada (debugging), na STM32CubeIDE, insira um breakpoint na linha do programa, conforme mostrado a seguir:

```
void RTC_IRQHandler(void)
{
    printf("Em modo Handler. Tratando a Rotina de Servico de Interrupcao\n");
}
```

Mas, como estamos iniciando a depuração, então, a linha atual de depuração é a seguinte:

```
int main(void)
{
    printf("Em modo Thread. Antes da Interrupcao\n");
    gerar_interrupcao();
    printf("Em modo Thread. Depois da Interrupcao\n");
    for(;;);
}
```

Temos que monitorar o Registrador IPSR, portanto, temos que visualizar a janela chamada "Registers". No menu "Window", parte superior da tela, selecione "Show View" e "Registers". A janela "Registers" será mostrada conforme ilustrado na figura 3. Selecione o Registrador "xpsr', que é o Registrador onde monitoraremos o valor do campo ISR_NUMBER [8:0].

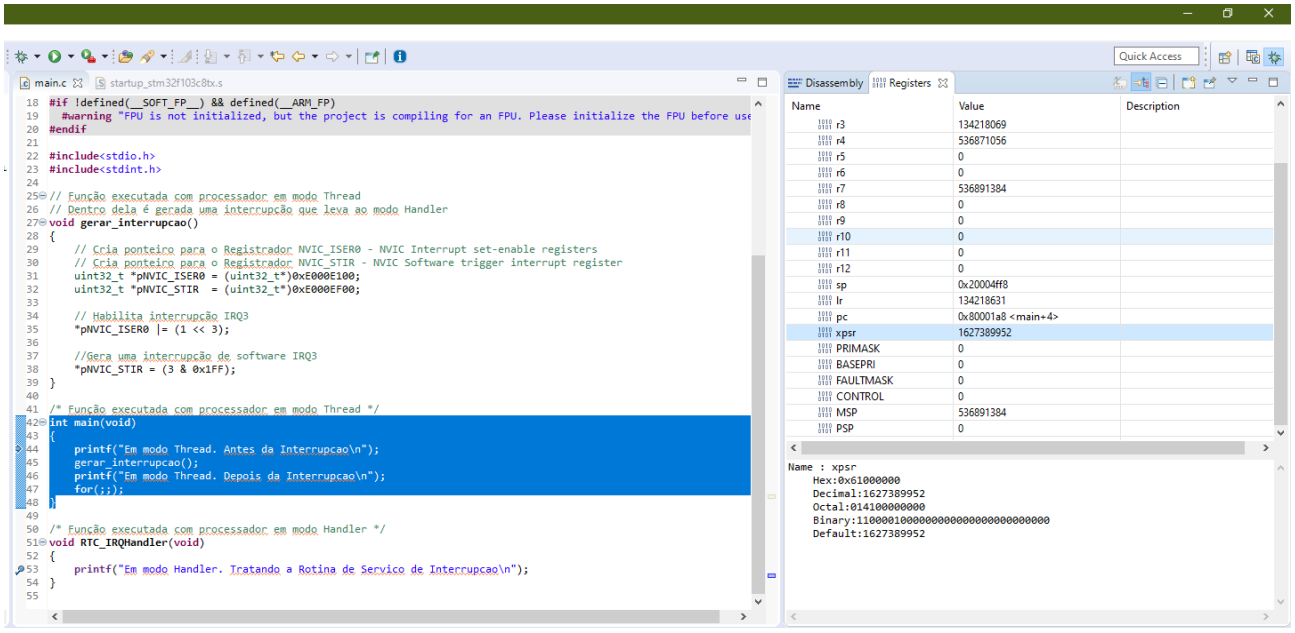

Figura 3 – Janela "Registers" destaque no Registrador "xpsr"

Voltando a depuração, estamos parado na primeira linha da função main.c. Na janela "Registers", quando selecionamos o Registrador "xpsr" seu conteúdo é mostrado logo abaixo, conforme a seguir:

Name : xpsr
Hex:0x1000000
Decimal:16777216
Octal:0100000000

Binary:10000000000000000000000000 (modo Thread)
Default:16777216

Os valores são mostrados em vários formatos numéricos, portanto, destaquei apenas o formato binário, que nos mostrará em detalhes o conteúdo do campo ISR_NUMBER. Inicialmente o processador opera em modo Thread, o que é confirmado porque ISR_NUMBER = 0.

1) Agora vamos pressionar a tecla [F5], que é um "step into", onde caminharemos linha por linha no processo de depuração. Nesse momento focaremos apenas no valor do campo ISR_NUMBER, para acompanhar o modo de operação do processador em cada parte da aplicação.

```
int main(void)
{
        printf("Em modo Thread. Antes da Interrupcao\n");
        gerar_interrupcao();
        printf("Em modo Thread. Depois da Interrupcao\n");
        for(;;);
}
```

Name : xpsr
Binary:10000000000000000000000000 (modo Thread)

2) Vamos dar mais um passo, pressionando a tecla [F5]. Entramos na função "gerar_interrupcao", e continuamos operando em modo Thread.

```
void gerar_interrupcao()
{
        // Cria ponteiro para o Registrador NVIC_ISER0 - NVIC Interrupt set-enable registers
        // Cria ponteiro para o Registrador NVIC_STIR - NVIC Software trigger interrupt register
        uint32_t *pNVIC_ISER0 = (uint32_t*)0xE000E100;
        uint32_t *pNVIC_STIR  = (uint32_t*)0xE000EF00;

        // Habilita interrupção IRQ3
        *pNVIC_ISER0 |= (1 << 3);

        //Gera uma interrupção de software IRQ3
        *pNVIC_STIR = (3 & 0x1FF);
}
```

Name : xpsr
Binary:10000000000000000000000000 (modo Thread)

3) Vamos dar mais um passo, pressionando a tecla [F5]. Continuamos em modo Thread.

```
void gerar_interrupcao()
{
        // Cria ponteiro para o Registrador NVIC_ISER0 - NVIC Interrupt set-enable registers
        // Cria ponteiro para o Registrador NVIC_STIR - NVIC Software trigger interrupt register
        uint32_t *pNVIC_ISER0 = (uint32_t*)0xE000E100;
```

```
        uint32_t *pNVIC_STIR  = (uint32_t*)0xE000EF00;

        // Habilita interrupção IRQ3
        *pNVIC_ISER0 |= (1 << 3);

        //Gera uma interrupção de software IRQ3
        *pNVIC_STIR = (3 & 0x1FF);
}
```

Name : xpsr
Binary:10000000000000000000000000 (modo Thread)

4) Vamos dar mais um passo, pressionando a tecla [F5]. Continuamos em modo Thread.

```
void gerar_interrupcao()
{
        // Cria ponteiro para o Registrador NVIC_ISER0 - NVIC Interrupt set-enable registers
        // Cria ponteiro para o Registrador NVIC_STIR - NVIC Software trigger interrupt register
        uint32_t *pNVIC_ISER0 = (uint32_t*)0xE000E100;
        uint32_t *pNVIC_STIR  = (uint32_t*)0xE000EF00;

        // Habilita interrupção IRQ3
        *pNVIC_ISER0 |= (1 << 3);

        //Gera uma interrupção de software IRQ3
        *pNVIC_STIR = (3 & 0x1FF);
}
```

Name : xpsr
Binary:10000000000000000000000000 (modo Thread)

5) Vamos dar mais um passo, pressionando a tecla [F5]. Continuamos em modo Thread.

```
void gerar_interrupcao()
{
        // Cria ponteiro para o Registrador NVIC_ISER0 - NVIC Interrupt set-enable registers
        // Cria ponteiro para o Registrador NVIC_STIR - NVIC Software trigger interrupt register
        uint32_t *pNVIC_ISER0 = (uint32_t*)0xE000E100;
        uint32_t *pNVIC_STIR  = (uint32_t*)0xE000EF00;

        // Habilita interrupção IRQ3
        *pNVIC_ISER0 |= (1 << 3);

        //Gera uma interrupção de software IRQ3
        *pNVIC_STIR = (3 & 0x1FF);
}
```

Name : xpsr
Binary:10000000000000000000000000 (modo Thread)

6) Agora não podemos dar mais um passo usando "step into", porque o depurador não gatilhará a interrupção de software IRQ3, portanto, como adicionamos o breakpoint dentro da função de tratamento de interrupção "RTC_IRQHandler", então, para funcionar temos que pressionar a tecla [F8], que faz um "Resume", sendo assim, a próxima linha gera a interrupção IRQ3 e o programa rodará até o breakpoint normalmente.

```
void RTC_IRQHandler(void)
{
        printf("Em modo Handler. Tratando a Rotina de Servico de Interrupcao\n");
}
```

Name : xpsr
Binary:10000000000000000000010011 (modo Handler)

ATENÇÃO! O processador foi para o modo Handler, e o campo ISR_NUMBER = 19, que indica que a interrupção IRQ3 foi gatilhada.

7) Agora podemos voltar a dar mais um passo, pressionando a tecla [F5].

```
int main(void)
{
        printf("Em modo Thread. Antes da Interrupcao\n");
        gerar_interrupcao();
        printf("Em modo Thread. Depois da Interrupcao\n");
        for(;;);
}
```

Name : xpsr
Binary:10000000000000000000000000 (modo Thread)

ATENÇÃO! O processador voltou para o modo Thread, e o campo ISR_NUMBER = 0. Significa que a rotina de interrupção foi tratada e o programa voltou ao modo Thread.

Podemos interromper a depuração, porque já caminhamos pelos trechos de código que nos interessava.

Para finalizar colocaremos novamente em processo de depuração, e visualizar a saída da janela "SWV ITM Data Console", e apenas ver as mensagens capturadas. Destaque a janela "SWV ITM Data Console", "start trace" e pressione a tecla [F8], para resumir a depuração. A depuração rodará até o breakpoint, então, pressione novamente a tecla [F8] para concluir a depuração. Para finalizar interrompa o processo de depuração.

Figura 4 – Janela "SWV ITM Data Console" e as mensagens capturadas

As mensagens capturadas são mostradas a seguir, e podemos confirmar tudo que vimos em detalhes no passo a passo.

- ✔ Em modo Thread. Antes da Interrupcao
- ✔ Em modo Handler. Tratando a Rotina de Servico de Interrupcao
- ✔ Em modo Thread. Depois da Interrupcao