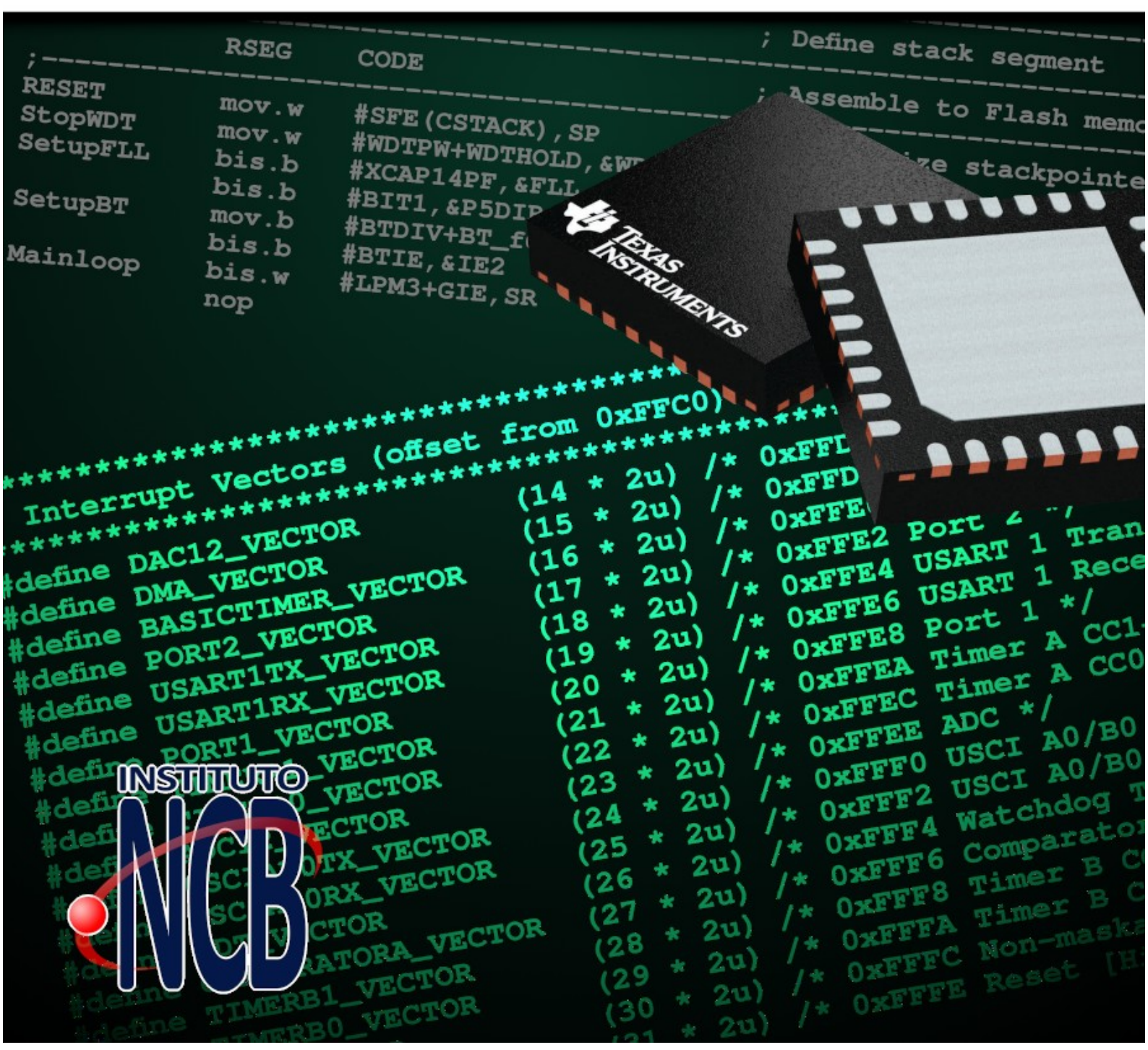


Alessandro Ferreira da Cunha

MSP430

LINGUAGEM ASSEMBLY



Alessandro Ferreira da Cunha

MSP430

LINGUAGEM ASSEMBLY

INSTITUTO NEWTON C. BRAGA



www.newtonbraga.com.br

leitor@newtonbraga.com.br

Diretor responsável: Newton C. Braga

Diagramação e Coordenação: Renato Paiotti

Revisão: Marcelo Braga

MSP430 – Linguagem Assembly: Alessandro Ferreira da Cunha
São Paulo - Brasil - 2021

Palavras-chaves: MSP430, microcontroladores, linguagem de programação,
linguagem assembly

Todos os direitos reservados. Proibida a reprodução total ou parcial, por qualquer meio ou processo, especialmente por sistemas gráficos, microfílmicos, fotográficos, reprográficos, fonográficos, videográficos, atualmente existentes ou que venham a ser inventados. Vedada a memorização e/ou a recuperação total ou parcial em qualquer parte da obra em qualquer programa juscibernético atualmente em uso ou que venha a ser desenvolvido ou implantado no futuro. Essas proibições aplicam-se também às características gráficas da obra e à sua editoração. A violação dos direitos autorais é punível como crime (art. 184 e parágrafos, do Código Penal, cf. Lei nº 6.895, de 17/12/80) com pena de prisão e multa, conjuntamente com busca e apreensão e indenização diversas (artigos 122, 123, 124, 126 da Lei nº 5.988, de 14/12/73, Lei dos Direitos Autorais).

Índice

Arquiteturas von-Neumann X Arquiteturas Harvard.....	8
von-Neumann.....	8
Harvard.....	9
Qual a escolha da Texas para o MSP430?.....	10
Máquina von-Newmann.....	11
Máquina von-Neumann modificada.....	13
CISC x RISC.....	13
CISC - Complex Instruction Set Computer.....	13
RISC - Reduced Instruction Set Computer.....	14
Famílias MSP430 e detalhamento de hardware.....	15
Clocks e LPM.....	17
Sinais de clock externo.....	18
Sinais de clock internos.....	18
Os registradores que controlam os clocks.....	22
As frequências possíveis no DCO.....	23
LPMs - LOW POWER MODES.....	25
Os registradores de trabalho, ou registradores especiais.....	29
Program counter (contador de programa).....	30
Stack Pointer (ponteiro da pilha).....	30
Status Register (R2).....	31
Constant Generator (R3).....	32
General Purpouse Registers (R4 – R15).....	34
Estrutura de memórias: RAM, ROM (Flash).....	35
Memórias no MSP430.....	35
Memórias de programa (ROM – FLASH).....	36
Reduced Instruction Set Code – RISC.....	38
Modos de endereçamento.....	39
Formatos das instruções.....	40
As 51 instruções.....	41
Ciclos de máquina e tamanho das instruções.....	44

A Placa de exercícios: Experimenter Board.....	47
Instalando os drives do gravador FET USB.....	52
AMBIENTE IAR.....	56
Começando do zero: Criando um novo projeto.....	56
Como Estruturar um programa em Assembly para MSP430?.....	69
Exercício 1: Botões e LEDs.....	71
Exercício 1a: 1 botão e 1 Led.....	71
Exercício 1b: 2 botões e 2 Leds.....	71
Exercício 1c: 1 botão, 1 Led e temporização simples.....	72
Exercício 1d: 2 botões, 2 Leds e temporização simples.....	72
Configuração básica dos I/Os.....	72
Alocação de pinos nas portas de I/O.....	74
Exemplos de configuração de portas de I/O.....	74
Exercício 2: Economizando energia para piscar o LED.....	75
Interrupções.....	77
Reset do sistema.....	78
NMI – Interrupções não mascaráveis.....	79
Interrupções mascaráveis.....	80
O processamento de uma interrupção.....	80
Entrada em uma interrupção.....	80
Saída de uma interrupção.....	81
Os vetores de interrupção.....	82
Vetores de interrupção na família 2 (MSP430F2013).....	83
Declaração dos Vetores de interrupção no MSP430F2013.....	84
Vetores de interrupção na família 4 (MSP430FG4618).....	84
Declaração dos Vetores de interrupção no MSP430FG4618.....	85
BASIC TIMER 1.....	86
Exercício 3: Botões e LEDs em Power Mode.....	91
Exercício 3a: 1 botão e 1 Led.....	92
Exercício 3b: 2 botões e 2 Leds.....	92
Exercício 3c: 1 botão, 1 Led e temporização com Basic Timer 1.....	92
Exercício 3d: 2 botões, 2 Leds e temporização Basic Timer 1.....	93
TIMER A.....	94
Inicializando o Timer A.....	94

O controle do Timer A.....	95
UP MODE.....	97
CONTINUOUS MODE.....	97
UP/DOWN MODE.....	97
Modos de saída do Timer A.....	98
Exemplos de saída em UP/DOWN MODE.....	99
Exemplos de saída em UP MODE.....	99
Exemplos de saída em CONTINUOS MODE.....	100
Continuous Mode Time Intervals.....	101
Exemplos de saída em UP/DOWN MODE.....	102
Os registradores de controle do Timer A.....	103
TIMER B.....	106
Os registradores de controle do Timer B.....	108
Exercício 4: Acender o LED em Low Power Mode com PWM.....	112
Exercício 4a: 1 Led com PWM em 50%.....	115
Exercício 4b: 1 Led com PWM ajustável por 2 botões - pressionando.....	115
Exercício 4c: 1 Led com PWM ajustável por 2 botões - contínuo.....	115
Comparador A.....	117
Os Registradores de controle do Comparador A.....	119
Exercício 5: Comparar as tensões de entrada com um referencial.....	121
Exercício 5a: Acender o Led toda vez que a tensão passar de um patamar.....	122
Exercício 5b: Tocar o Buzzer avisando que a tensão está baixa.....	122
Amplificadores Operacionais.....	123
Exercício 6: Funcionamento do A.O. no MSP430.....	127
Exercício 6a: Módulo AO0 como comparador.....	128
Exercício 6b: Módulo AO1 como buffer de ganho unitário.....	129
Exercício 6c: Módulos AO0, AO1 e AO2 como Amplificador diferencial.....	130
Anexos.....	132

Arquiteturas von-Neumann X

Arquiteturas Harvard

von-Neumann

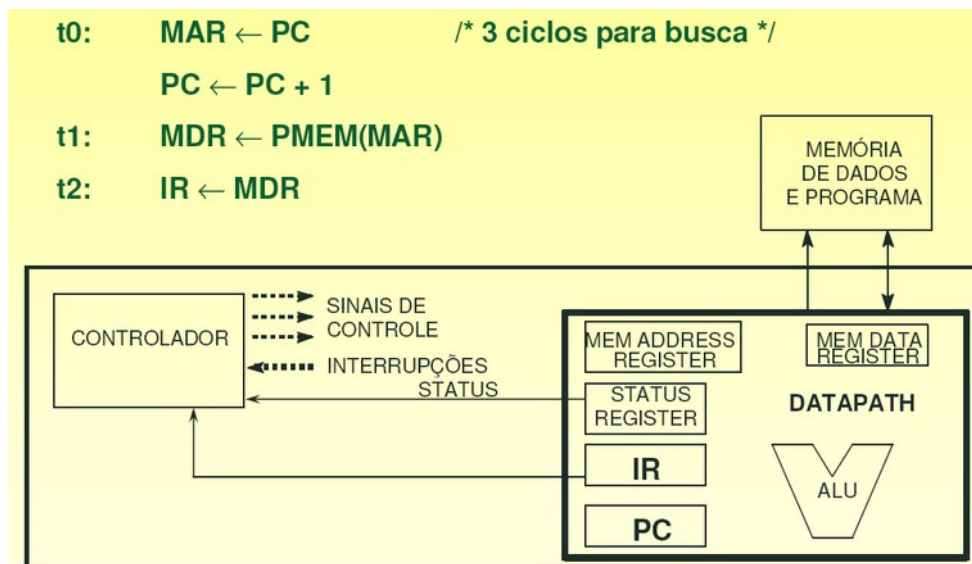
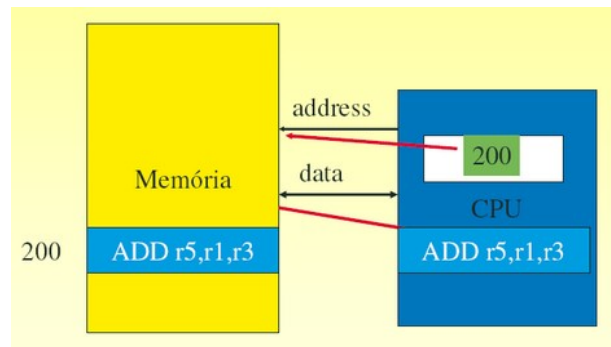
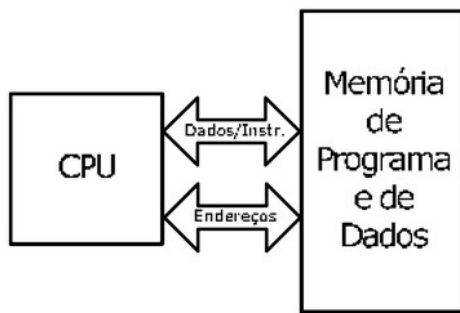
Algoritmos para computadores se baseiam em alguns conceitos básicos e em um modelo de computador, os quais devem ser bem entendidos para que se possa criar algoritmos eficientes. Este modelo foi proposto pelo matemático húngaro **Neumann János Lajos Margittai**. Em húngaro o nome de família aparece antes. Assim em português o seu nome seria **János Lajos Margittai Neumann**. O seu pai, que era rico, comprou um título de nobreza e ele passou a se chamar **János Lajos Margittai von Neumann**.

No modelo de computador proposto por **von Neumann** as instruções e os dados ficam juntos na memória. O processador busca na memória e executa uma instrução de cada vez. Portanto, as transferências entre a memória e o processador são feitas passo a passo. O ciclo normal da execução de um programa é então:

1. Busca instrução;
2. Decodifica instrução;
3. Executa instrução;
4. Volta para o passo 1 buscando a instrução seguinte na memória.

Dados e programas compartilham um meio de armazenamento único.

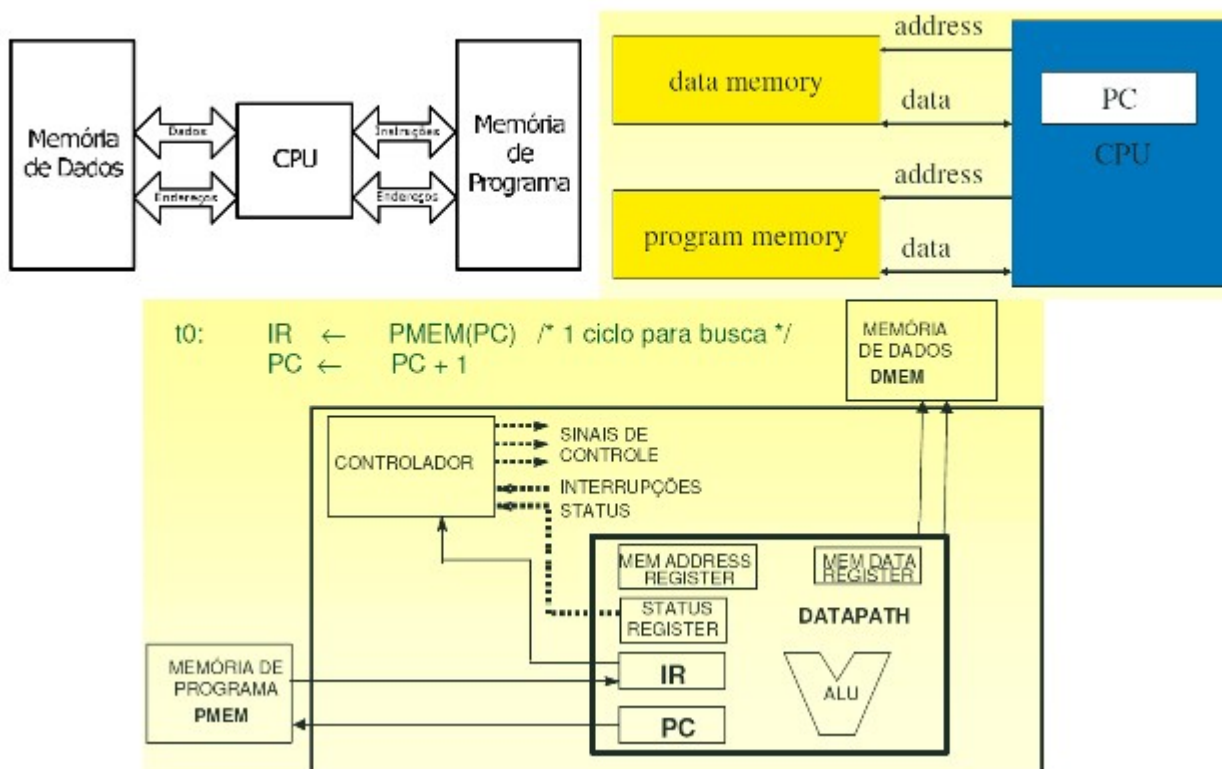
- Mais simples, menos restritivo, menos eficiente – dados e programas misturados permitem ao programador intercambiar a semântica de dados e programas ao longo do tempo



Harvard

Dados e programas estocados em meios de armazenamento distintos

- Mais propenso a fomentar paralelismo, mais caro, mais complexo – dados e programas separados permitem que ambos sejam facilmente tratados em paralelo.
- Harvard permite duas leituras de memória simultâneas (dado e instrução).
- A maioria dos processadores DSP (celulares, telecom, câmeras digitais,...) usam organização Harvard, pois isto permite maior largura de banda de memória e tempo de acesso a dados mais previsível.



Qual a escolha da Texas para o MSP430?

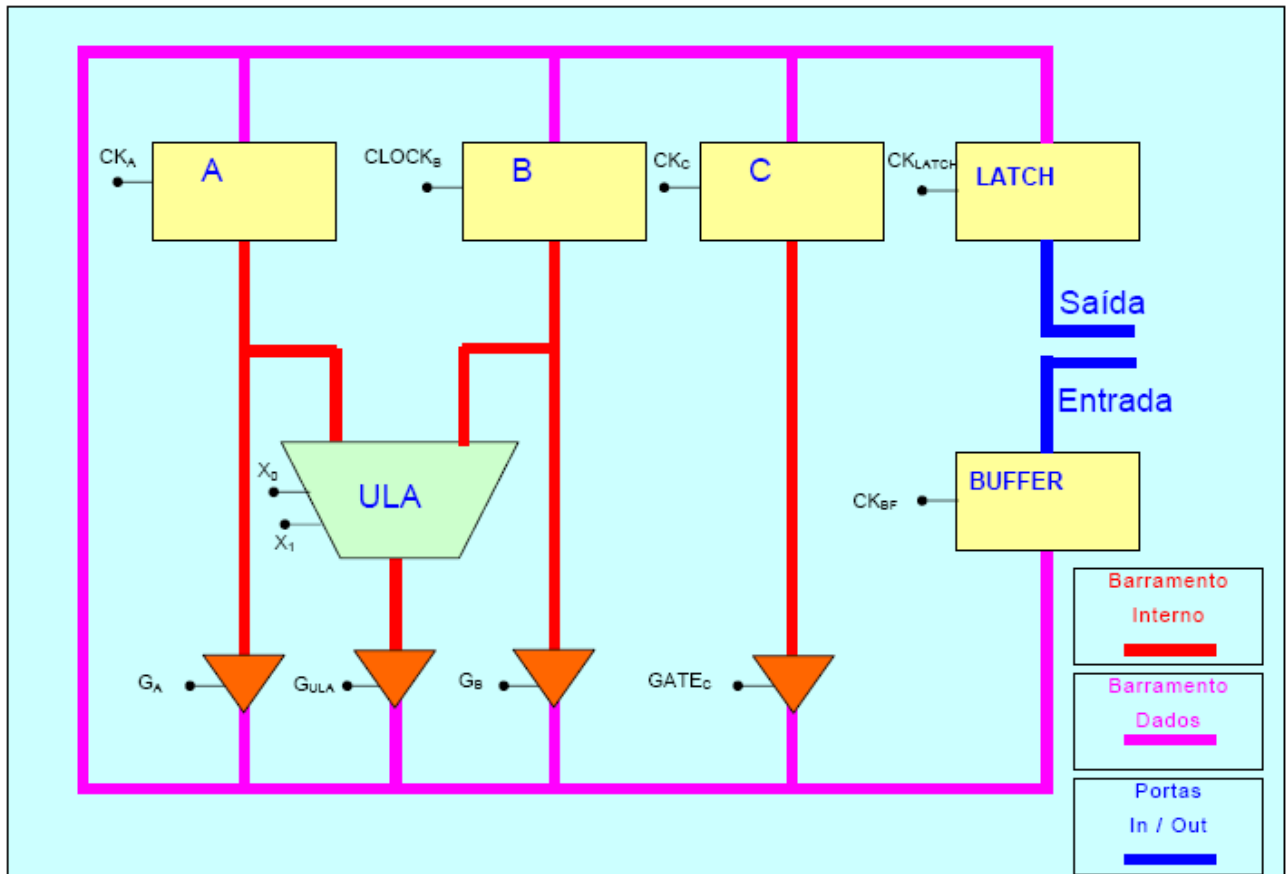
Veja o que dizem os manuais das famílias MSP430x4xx e MSP430x2xx:

“Architecture

The MSP430 incorporates a 16-bit RISC CPU, peripherals, and a flexible clock system that interconnect using a **von-Neumann** common memory address bus (MAB) and memory data bus (MDB). Partnering a modern CPU with modular memory-mapped analog and digital peripherals, the MSP430 offers solutions for demanding mixed-signal applications.”

Isso significa que a Texas optou por uma arquitetura **von-Neumann** “modificada”, tendo dois barramentos separados (como na arquitetura Harvard) mas fazendo acesso a apenas um barramento por vez (como na arquitetura **von-Neumann**). Como será visto ao longo deste livro, isto acaba tirando proveito das vantagens de cada uma das arquiteturas em um único chip.

Máquina von-Neumann



Exemplo de funcionamento de uma arquitetura Von Neumann: SOMAR DOIS NÚMEROS QUE SÃO INSERIDOS NA ENTRADA E COLOCAR O RESULTADO NA SAÍDA.

1. Colocar o **primeiro número** a ser somando na *porta de entrada* do μC .
2. Dar um pulso de clock no **BUFFER**.
 - 2.1. O **primeiro número** passa a ocupar o *barramento de dados* e fica armazenado no **BUFFER**.
3. Dar um pulso de clock no **Registrador A**.
 - 3.1. O **primeiro número** passa a ocupar o *barramento interno*, fica armazenado no **Registrador A** e está na entrada da ULA.

4. Colocar o **segundo número** a ser somado na *porta de entrada* do μC .
 - 4.1. Enquanto não for aplicado no **BUFFER** um pulso de clock, em sua entrada terá o **segundo número** e em sua saída terá o **primeiro número**.

5. Dar um pulso de clock no **BUFFER**.
 - 5.1. Ao fazer isso, o primeiro número é apagado da saída do **BUFFER**, que passa a ser ocupado pelo **segundo número**. Caso o **primeiro número** não tenha sido armazenado em algum **Registrador**, ele será perdido.

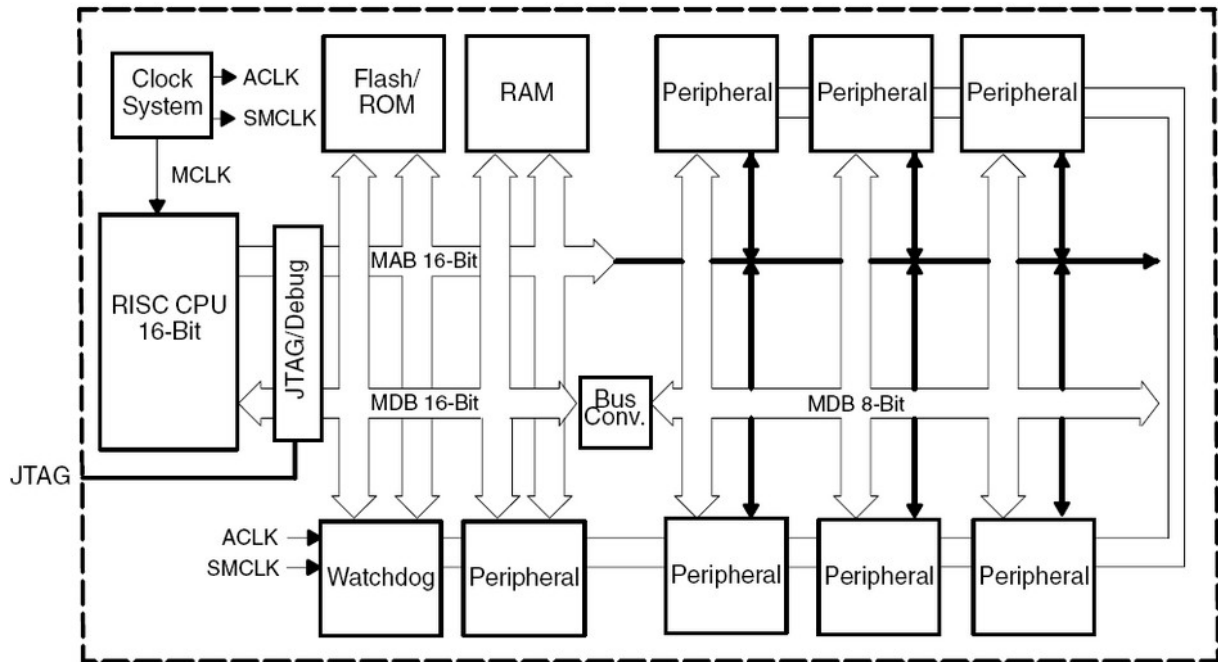
6. Dar um pulso de clock no **Registrador B**.
 - 6.1. O segundo número passa a ocupar o *barramento interno*, fica armazenado no Registrador B e está na entrada da ULA.

7. Colocar nos bits de comando da ULA a informação **$X_0 = 0$ e $X_1 = 0$** .
 - 7.1. Ao fazer isso, a ULA calculará a soma dos dados que estejam presentes em suas entradas. Se não houver dados na entrada, ela somará zero com zero. Automaticamente, o resultado é colocado na saída da ULA.

8. Dar um pulso de clock no *Gate da ULA* (GULA).
 - 8.1. Ao fazer isso o **segundo número** é apagado do *barramento de dados*, que passa a ser ocupado pelo resultado da soma dos dois números.

9. Dar um pulso de clock no **LATCH**.
 - 9.1. O resultado da soma é colocado na *porta de saída*.

Máquina von-Neumann modificada



CISC x RISC

A principal função de um microcontrolador é executar uma série de ordens (comandos ou instruções), armazenados em um programa, na sua memória. Cada uma destas instruções é interpretada pela CPU e, então, executada. Assim, cada tipo de computador digital, deve ter um conjunto de instruções (ou set de instruções) definidas, de modo que sua CPU possa interpretá-las e executá-las. Em função deste conjunto de instruções, existem duas classes de computadores digitais:

CISC - Complex Instruction Set Computer

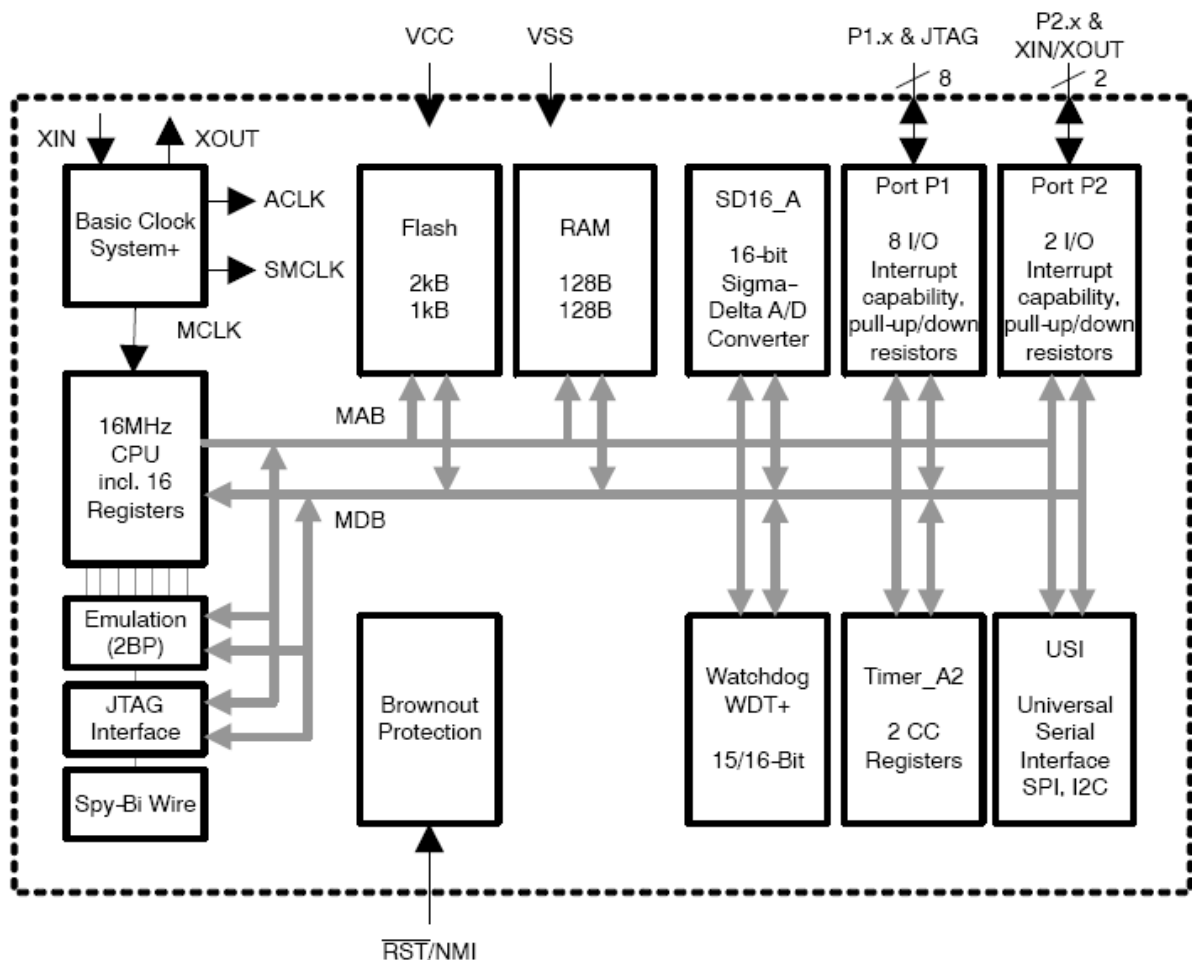
- Conjunto de instruções ampliado, ou seja, a CPU é capaz de executar um grande número de instruções (ex.: microcontrolador 8051, da Intel, com 256 instruções);
- É geralmente associada a computadores com arquitetura von-Neumann.

RISC - Reduced Instruction Set Computer

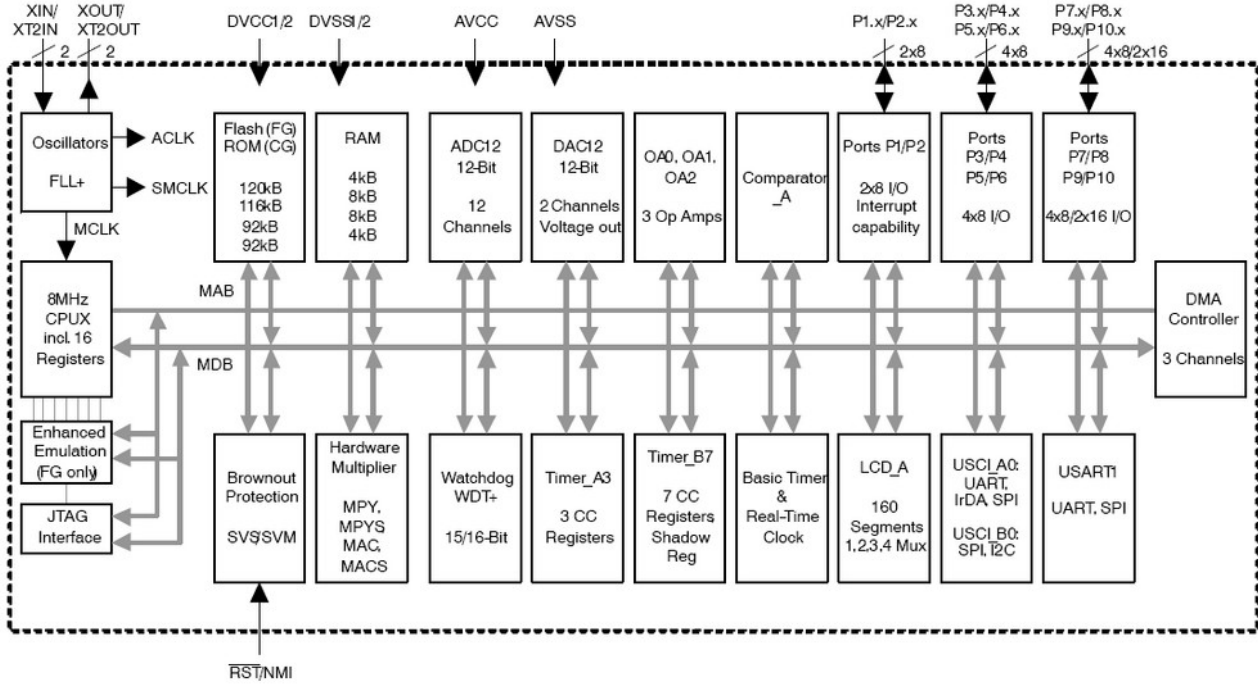
- Conjunto de instruções reduzido (ex.: família PIC, da Microchip, com 35 instruções, e família MSP430, da Texas, com 24 instruções emuladas e 27 instruções físicas);
- É geralmente associada a computadores com arquitetura Harvard.

Famílias MSP430 e detalhamento de hardware

Arquitetura do MSP430F2013.



Arquitetura do MSP430FG4618.



Clocks e LPM

Microcontroller	CPU Clock Divider
MSP430FG4619	1
MSP430F149	1
Microchip PIC24FJ128GA	2
Microchip PIC18F242	4
Generic 8051	1...12 ⁽¹⁾
Renesas H8/300H	2
MaxQ20	1
ARM7TDMI	1
Freescall HCS12	2
Atmel ATmega8	1

- (1) 8051 architectures typically use a divider of 12. However, some improved architectures can execute a subset of instructions in as little as one clock cycle per instruction.

FAMÍLIA 4 - 16-Bit RISC Architecture, 125-ns Instruction Cycle Time

FAMÍLIA 2 - 16-Bit RISC Architecture, 62.5 ns Instruction Cycle Time

Quanto tempo demora a ser executada cada uma das instruções no MSP430? Como acontece no 8051, a quantidade de ciclos de máquina necessária para executar uma instrução varia de acordo com o modo de endereçamento utilizado e com o formato da instrução. Sempre as instruções são referenciadas ao sinal de clock chamado de **MCLK**.

Os chips MSP430 se caracterizam pelo baixo consumo de energia. Um dos métodos utilizados para obter estas funcionalidades é o gerenciamento de clock e os modos de operação em baixa potência (LPM – Low Power Mode). Tanto a **família 2** quanto a **família 4** tem várias opções de clock que podem ser conectadas ao chip.

Sinais de clock externo

Na **família 2** o funcionamento do clock é gerenciado pelo **Basic Clock Module+**. Ele permite que até 4 tipos de clock diferentes sejam utilizados:

- **LFXT1CLK (Low-frequency/high-frequency oscillator):** Funciona com o uso de um cristal externo. Este cristal pode ser desde um modelo de baixa velocidade, fixado em 32.768 Hz, até cristais de quartzo ou ressonadores, com valores entre **400 khz e 16 Mhz**. Sinais de clock criados por geradores externos também são aceitos.
- **XT2CLK (Optional high-frequency oscillator):** Funciona com o uso de um cristal externo de alta velocidade. Isto pode ser feito através de cristais de quartzo, ressonadores ou fontes de clock externa (entre **400 khz e 16 Mhz**).
- **DCOCLK (Internal digitally controlled oscillator (DCO)).**
- **VLOCLK (Internal very low power, low frequency oscillator):** com 12-kHz de frequência típica.

Já na **família 4** o funcionamento do clock é bem similar ao da **família 2**, mas o gerenciamento é feito por um módulo que leva o nome de **FLL+ Clock Module**. Ele permite que somente 3 primeiros tipos de clock mostrados na **família 2** possam ser gerados, sendo que a opção **VLOCLK não está disponível**. As velocidades de clock externo admissíveis para a **família 4** vão de **450 khz a 8 Mhz**. Apenas os dispositivos MSP430F47x admitem velocidades até 16 Mhz.

Sinais de clock internos

Independente de qual fonte de clock foi utilizada pelo chip, sempre serão gerados três sinais de clock internamente:

- **ACLK (Auxiliary clock):** esta fonte de clock é selecionável por software quando as fontes são o **LFXT1CLK** ou o **VLOCLK**. **ACLK** pode ser dividido por 1, 2, 4, or 8. Esta é a fonte de clock utilizada por todos os módulos de periféricos.
- **MCLK (Master clock):** esta fonte de clock também é selecionável por software, para

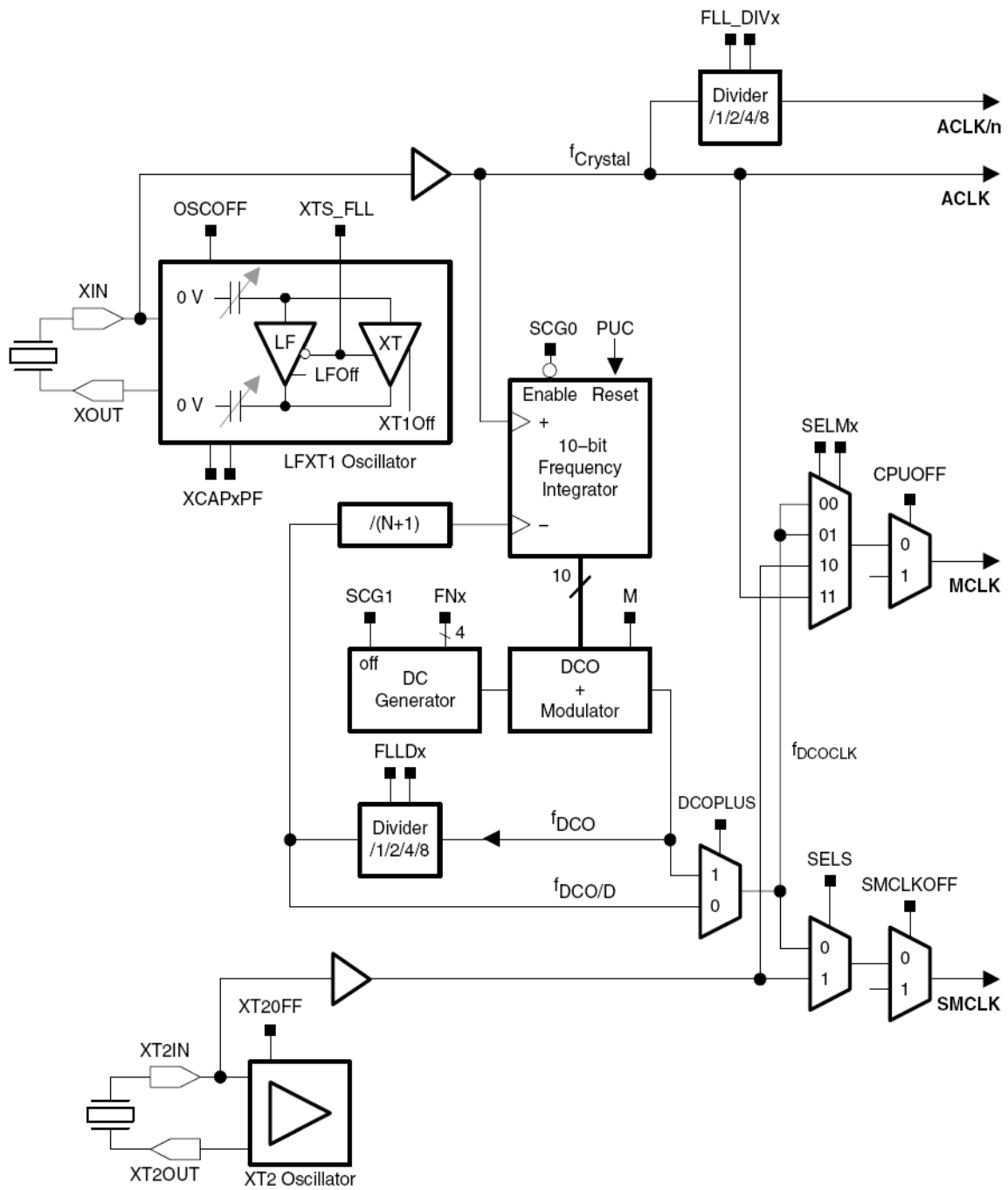
qualquer uma das quatro fontes de clock possíveis: **LFXT1CLK**, **VLOCLK**, **XT2CLK** ou **DCOCLK**. **MCLK** pode ser dividido por 1, 2, 4, ou 8. É utilizado para alimentar a CPU.

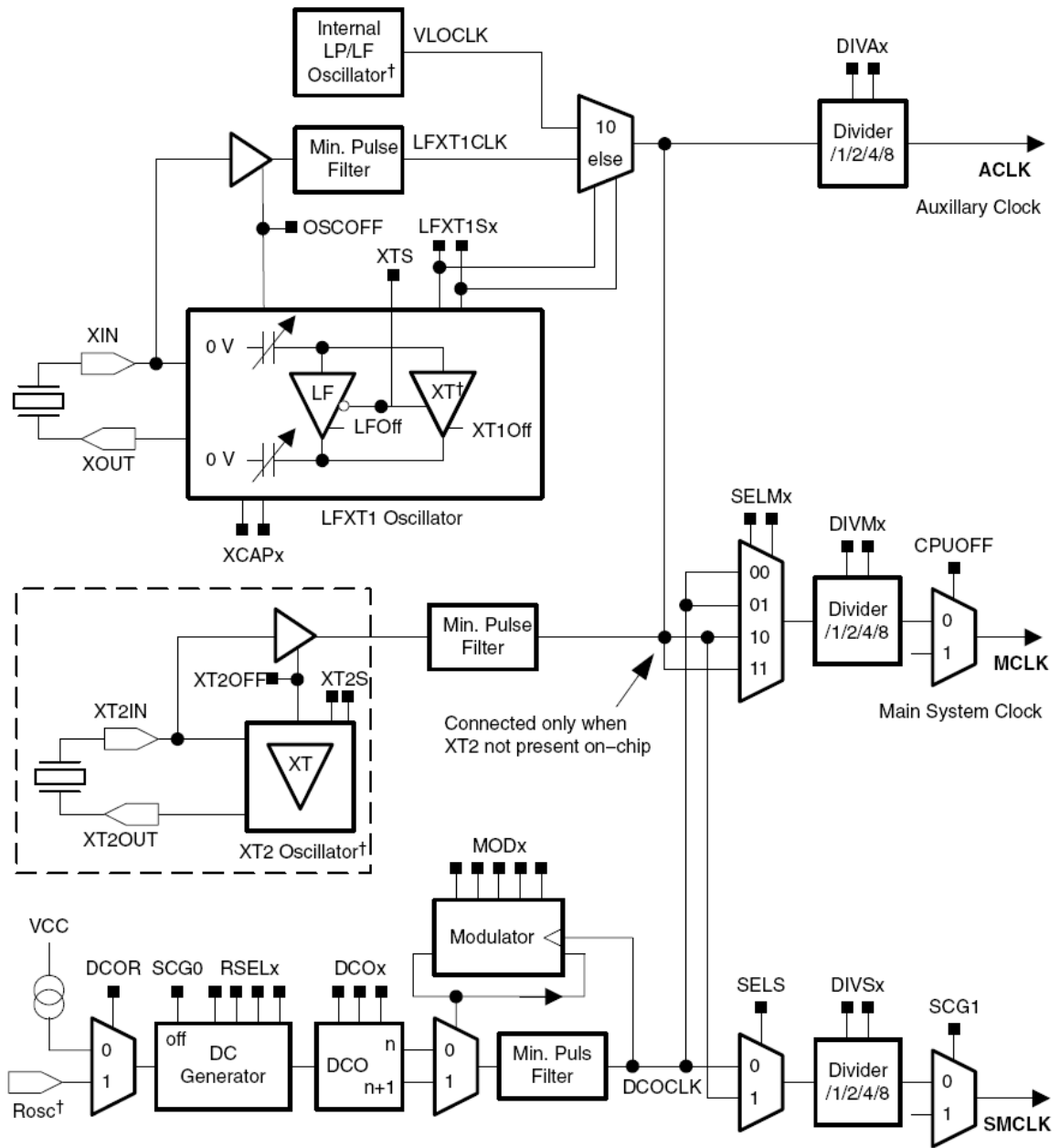
- **SMCLK (Sub-main clock)**: também é selecionável por software para qualquer uma das quatro fontes de clock possíveis, divisível por 1, 2, 4, ou 8. É utilizado individualmente para alimentar alguns periféricos.

Apenas na **família 4**, um quarto sinal de clock interno pode ser gerado, mas que é complementar aos já mostrados anteriormente:

- **ACLK/n (Auxiliary clock buffed output)**: faz um buffer de saída do sinal gerado pelo ACLK. Dedicar-se apenas ao uso externo ao chip, para gerar um sinal de sincronismo, por exemplo.

Na figura a seguir é possível visualizar o módulo que controla o clock do MSP430, para as famílias 43, 44 e 46. Depois é mostrado o módulo para a **família 2**.





Os registradores que controlam os clocks

FAMÍLIA 2

Table 5–1. Basic Clock module+ Registers

Register	Short Form	Register Type	Address	Initial State
DCO control register	DCOCTL	Read/write	056h	060h with PUC
Basic clock system control 1	BCSCTL1	Read/write	057h	087h with POR†
Basic clock system control 2	BCSCTL2	Read/write	058h	Reset with PUC
Basic clock system control 3	BCSCTL3	Read/write	053h	005h with PUC
SFR interrupt enable register 1	IE1	Read/write	000h	Reset with PUC
SFR interrupt flag register 1	IFG1	Read/write	002h	Reset with PUC

† Some of the register bits are also PUC initialized. See register summary.

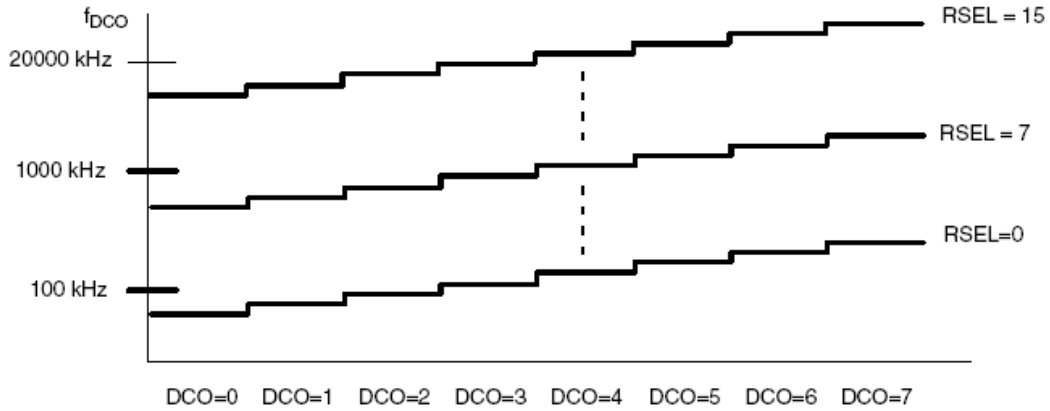
FAMÍLIA 4

Table 5–2. FLL+ Registers

Register	Short Form	Register Type	Address	Initial State
System clock control	SCFQCTL	Read/write	052h	01Fh with PUC
System clock frequency integrator 0	SCFI0	Read/write	050h	040h with PUC
System clock frequency integrator 1	SCFI1	Read/write	051h	Reset with PUC
FLL+ control register 0	FLL_CTL0	Read/write	053h	003h with PUC
FLL+ control register 1	FLL_CTL1	Read/write	054h	Reset with PUC
FLL+ control register 2 (F47x only)	FLL_CTL2	Read/write	055h	Reset with PUC
SFR interrupt enable register 1	IE1	Read/write	0000h	Reset with PUC
SFR interrupt flag register 1	IFG1	Read/write	0002h	Reset with PUC

As frequências possíveis no DCO

FAMÍLIA 2



$$f_{average} = \frac{32 \times f_{DCO(RSEL,DCO)} \times f_{DCO(RSEL,DCO+1)}}{MOD \times f_{DCO(RSEL,DCO)} + (32 - MOD) \times f_{DCO(RSEL,DCO+1)}}$$

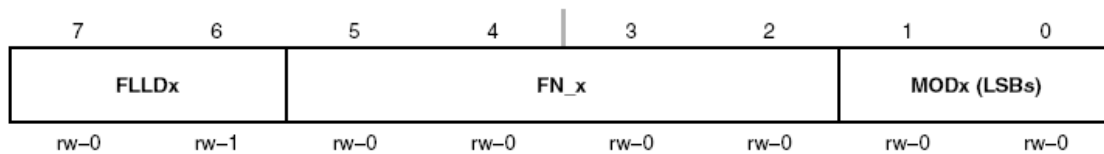
DCO frequency

PARAMETER		TEST CONDITIONS	VCC	MIN	TYP	MAX	UNIT
Vcc	Supply voltage range	RSELx < 14		1.8		3.6	V
		RSELx = 14		2.2		3.6	V
		RSELx = 15		3.0		3.6	V
f _{DCO(0,0)}	DCO frequency (0, 0)	RSELx = 0, DCOx = 0, MODx = 0	2.2 V/3 V	0.06		0.14	MHz
f _{DCO(0,3)}	DCO frequency (0, 3)	RSELx = 0, DCOx = 3, MODx = 0	2.2 V/3 V	0.07		0.17	MHz
f _{DCO(1,3)}	DCO frequency (1, 3)	RSELx = 1, DCOx = 3, MODx = 0	2.2 V/3 V	0.10		0.20	MHz
f _{DCO(2,3)}	DCO frequency (2, 3)	RSELx = 2, DCOx = 3, MODx = 0	2.2 V/3 V	0.14		0.28	MHz
f _{DCO(3,3)}	DCO frequency (3, 3)	RSELx = 3, DCOx = 3, MODx = 0	2.2 V/3 V	0.20		0.40	MHz
f _{DCO(4,3)}	DCO frequency (4, 3)	RSELx = 4, DCOx = 3, MODx = 0	2.2 V/3 V	0.28		0.54	MHz
f _{DCO(5,3)}	DCO frequency (5, 3)	RSELx = 5, DCOx = 3, MODx = 0	2.2 V/3 V	0.39		0.77	MHz
f _{DCO(6,3)}	DCO frequency (6, 3)	RSELx = 6, DCOx = 3, MODx = 0	2.2 V/3 V	0.54		1.06	MHz
f _{DCO(7,3)}	DCO frequency (7, 3)	RSELx = 7, DCOx = 3, MODx = 0	2.2 V/3 V	0.80		1.50	MHz
f _{DCO(8,3)}	DCO frequency (8, 3)	RSELx = 8, DCOx = 3, MODx = 0	2.2 V/3 V	1.10		2.10	MHz
f _{DCO(9,3)}	DCO frequency (9, 3)	RSELx = 9, DCOx = 3, MODx = 0	2.2 V/3 V	1.60		3.00	MHz
f _{DCO(10,3)}	DCO frequency (10, 3)	RSELx = 10, DCOx = 3, MODx = 0	2.2 V/3 V	2.50		4.30	MHz
f _{DCO(11,3)}	DCO frequency (11, 3)	RSELx = 11, DCOx = 3, MODx = 0	2.2 V/3 V	3.00		5.50	MHz
f _{DCO(12,3)}	DCO frequency (12, 3)	RSELx = 12, DCOx = 3, MODx = 0	2.2 V/3 V	4.30		7.30	MHz
f _{DCO(13,3)}	DCO frequency (13, 3)	RSELx = 13, DCOx = 3, MODx = 0	2.2 V/3 V	6.00		9.60	MHz
f _{DCO(14,3)}	DCO frequency (14, 3)	RSELx = 14, DCOx = 3, MODx = 0	2.2 V/3 V	8.60		13.9	MHz
f _{DCO(15,3)}	DCO frequency (15, 3)	RSELx = 15, DCOx = 3, MODx = 0	3 V	12.0		18.5	MHz
f _{DCO(15,7)}	DCO frequency (15, 7)	RSELx = 15, DCOx = 7, MODx = 0	3 V	16.0		26.0	MHz
S _{RSEL}	Frequency step between range RSEL and RSEL+1	S _{RSEL} = f _{DCO(RSEL+1,DCO)} /f _{DCO(RSEL,DCO)}	2.2 V/3 V			1.55	ratio
S _{DCO}	Frequency step between tap DCO and DCO+1	S _{DCO} = f _{DCO(RSEL,DCO+1)} /f _{DCO(RSEL,DCO)}	2.2 V/3 V	1.05	1.08	1.12	
Duty Cycle		Measured at P1.4/SMCLK	2.2 V/3 V	40	50	60	%

FAMÍLIA 4

FN_8	FN_4	FN_3	FN_2	Typical f_{DCO} Range
0	0	0	0	0.65–6.1
0	0	0	1	1.3–12.1
0	0	1	X	2–17.9
0	1	X	X	2.8–26.6
1	X	X	X	4.2–46

SCFI0, System Clock Frequency Integrator Register 0



FLLDx	Bits 7-6	FLL+ loop divider. These bits divide f_{DCOCLK} in the FLL+ feedback loop. This results in an additional multiplier for the multiplier bits. See also multiplier bits. 00 /1 01 /2 10 /4 11 /8
FN_x	Bits 5-2	DCO Range Control. These bits select the f_{DCO} operating range. 0000 0.65 - 6.1 MHz 0001 1.3 - 12.1 MHz 001x 2 - 17.9 MHz 01xx 2.8 - 26.6 MHz 1xxx 4.2 - 46 MHz
MODx	Bits 1-0	Least significant modulator bits. Bit 0 is the modulator LSB. These bits affect the modulator pattern. All MODx bits are modified automatically by the FLL+.

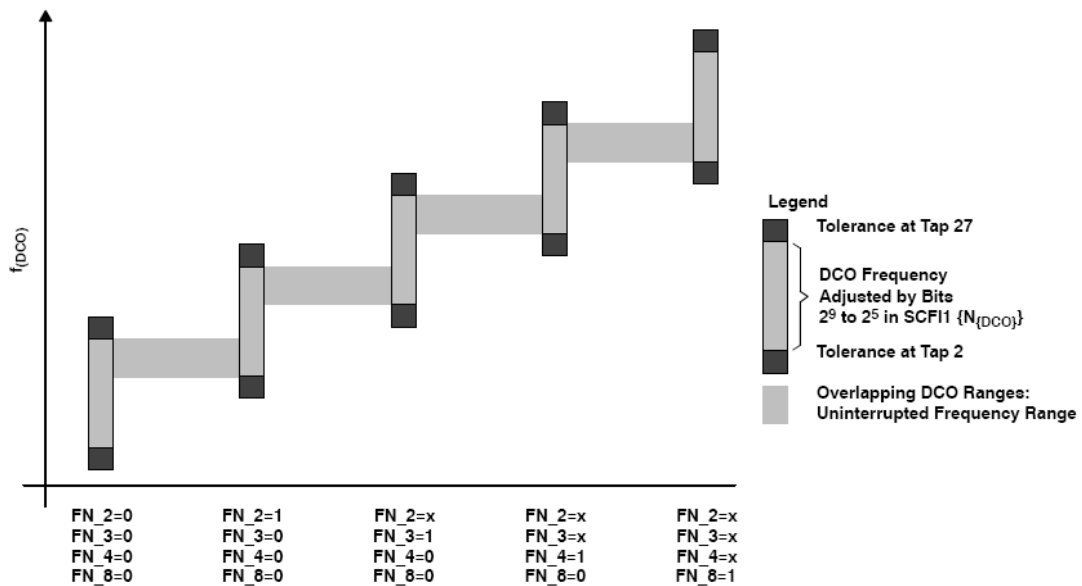
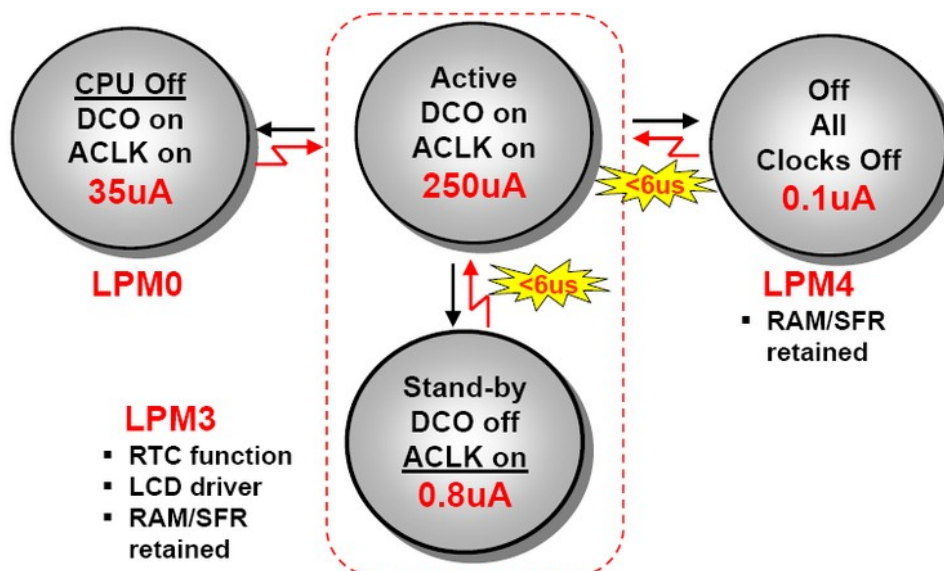


Figure 17. Five Overlapping DCO Ranges Controlled by FN_x Bits

LPMs - LOW POWER MODES

A existência de sinais de clock diferentes internamente permite que modos de consumo diferentes sejam utilizados de acordo com cada aplicação do usuário. Isto permite uma grande economia da energia consumida pelo chip.

Isto pode ser visualizado na figura a seguir.



A entrada ou saída em cada um dos modos de operação em baixo consumo é feito através de bits localizados no registrador especial R2, como será descrito em detalhes ao longo deste treinamento.

Reserved	V	SCG1	SCG0	OSC OFF	CPU OFF	GIE	N	Z	C
R2/SR									
Active Mode		0	0	0	0				~ 250uA
LPM0		0	0	0	1				~ 35uA
LPM3		1	1	0	1				~ 0.8uA
LPM4		1	1	1	1				~ 0.1uA

```
bis.w #CPUOFF,SR ; LPM0
```

São ao todo cinco modos de operação em baixa potência, além do modo ativo, como pode ser visto nas tabelas das famílias 2 e 4, logo abaixo:

FAMÍLIA 2

SCG1	SCG0	OSCOFF	CPUOFF	Mode	CPU and Clocks Status
0	0	0	0	Active	CPU is active, all enabled clocks are active
0	0	0	1	LPM0	CPU, MCLK are disabled SMCLK, ACLK are active
0	1	0	1	LPM1	CPU, MCLK are disabled, DCO and DC generator are disabled if the DCO is not used for SMCLK. ACLK is active
1	0	0	1	LPM2	CPU, MCLK, SMCLK, DCO are disabled DC generator remains enabled ACLK is active
1	1	0	1	LPM3	CPU, MCLK, SMCLK, DCO are disabled DC generator disabled ACLK is active
1	1	1	1	LPM4	CPU and all clocks disabled

FAMÍLIA 4

SCG1	SCG0	OSCOFF	CPUOFF	Mode	CPU and Clocks Status
0	0	0	0	Active	CPU is active, all enabled clocks are active
0	0	0	1	LPM0	CPU, MCLK are disabled (41x/42x peripheral MCLK remains on) SMCLK , ACLK are active
0	1	0	1	LPM1	CPU, MCLK, DCO osc. are disabled (41x/42x peripheral MCLK remains on) DC generator is disabled if the DCO is not used for MCLK or SMCLK in active mode SMCLK , ACLK are active
1	0	0	1	LPM2	CPU, MCLK, SMCLK, DCO osc. are disabled DC generator remains enabled ACLK is active
1	1	0	1	LPM3	CPU, MCLK, SMCLK, DCO osc. are disabled DC generator disabled ACLK is active
1	1	1	1	LPM4	CPU and all clocks disabled

A entrada ou saída de qualquer um destes modos pode ser feita através de mudanças nos bits do registrador R2, como pode ser visto nos exemplos abaixo:

```

; Enter LPM0 Example
BIS #GIE+CPUOFF,SR ; Enter LPM0
; Program stops here

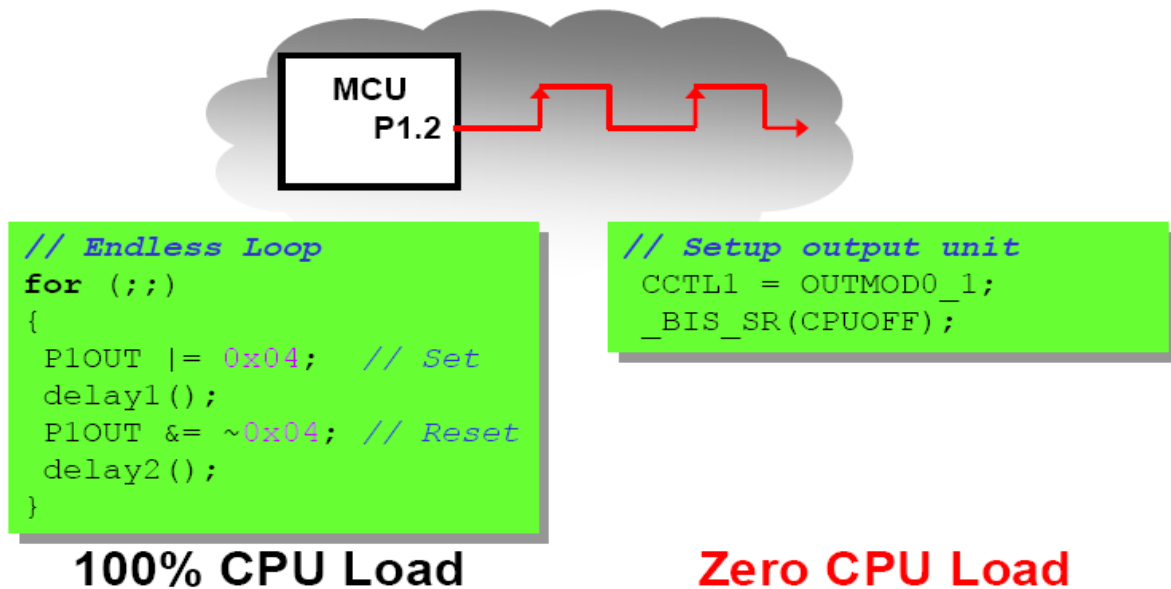
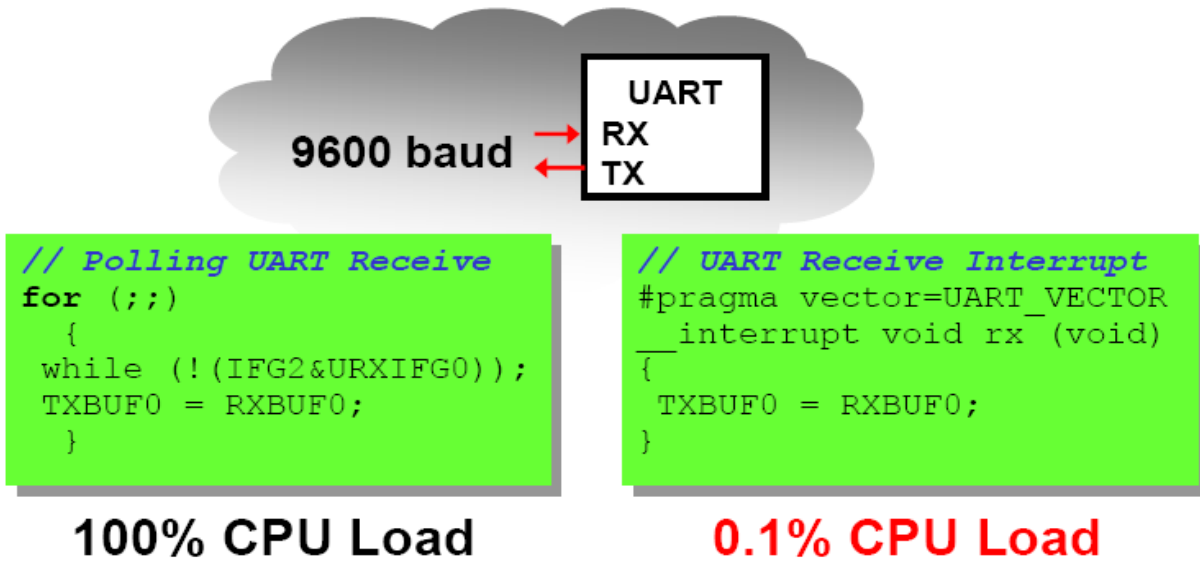
; Exit LPM0 Interrupt Service Routine
BIC #CPUOFF,0(SP) ; Exit LPM0 on RETI
RETI

; Enter LPM3 Example
BIS #GIE+CPUOFF+SCG1+SCG0,SR ; Enter LPM3
; Program stops here

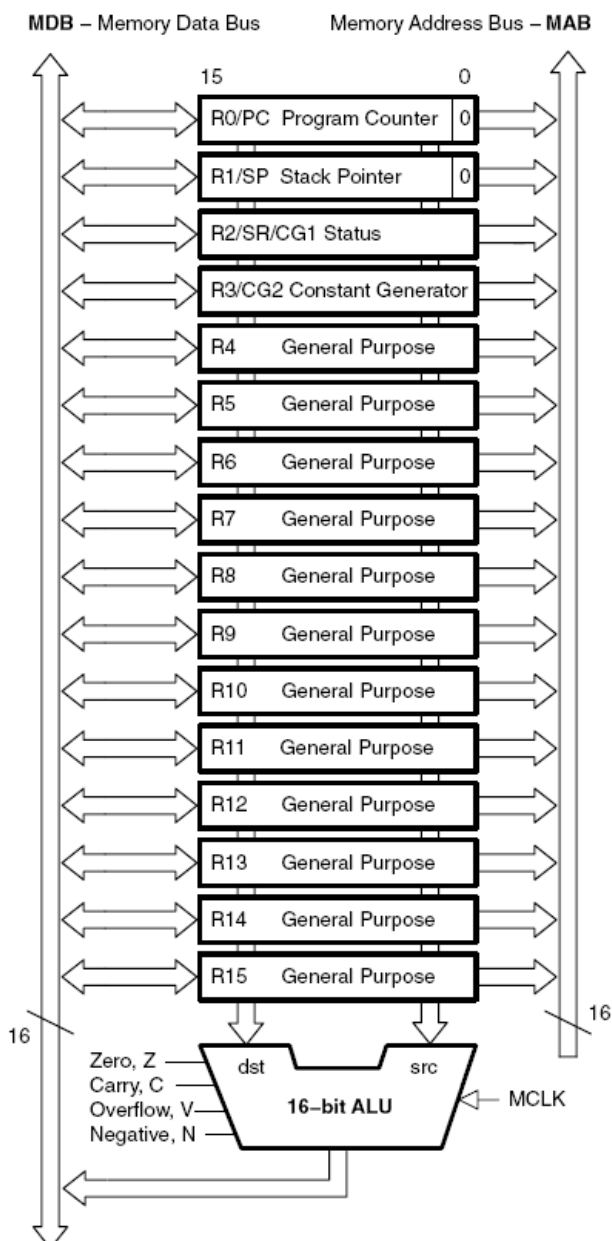
; Exit LPM3 Interrupt Service Routine
BIC #CPUOFF+SCG1+SCG0,0(SP) ; Exit LPM3 on RETI
RETI

```

Como em cada modo de potência apenas alguns clocks são desligados, pode-se deixar a CPU desligada e manter periféricos funcionando, o que comprova o baixo consumo do dispositivo.



Os registradores de trabalho, ou registradores especiais

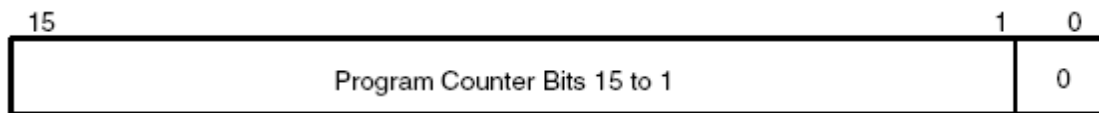


Diferente de outros fabricantes de microcontroladores, a Texas colocou nas famílias MSP430 16 registradores de trabalho com acesso direto a CPU e aos barramentos de dados e memória, como pode ser visto na figura abaixo. Isto gera uma grande praticidade de uso do chip, com facilidades que serão discutidas ao longo deste treinamento.

Vamos analisar cada um destes registradores em detalhes.

Program counter (contador de programa)

Aponta qual será a próxima instrução dentro do programa a ser executada pela CPU.

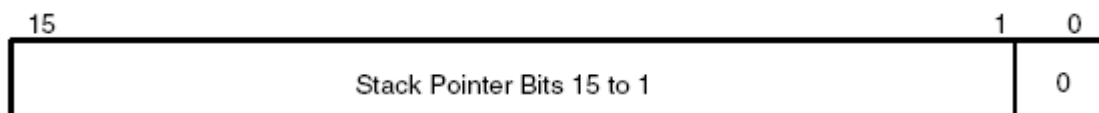


Ele pode ser endereçado por qualquer um dos sete modos existentes no MSP430. Alguns exemplos são mostrados a seguir.

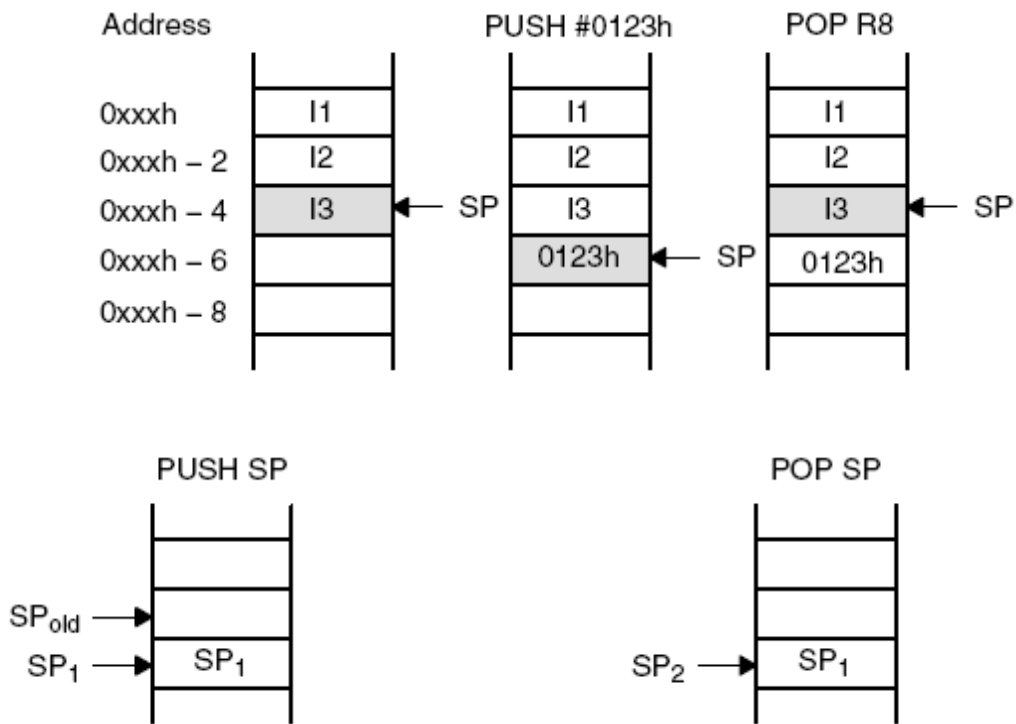
```
MOV #LABEL,PC ;desvio para o endereço do LABEL
MOV LABEL,PC ;desvio para o endereço indicado pelo LABEL
MOV @R14,PC ;desvio indireto indicado pelo R4
```

Stack Pointer (ponteiro da pilha)

A pilha é um recurso utilizado para armazenar informações enquanto são executadas rotinas de interrupção ou chamadas de sub-rotinas. Para tanto é necessário apontar qual posição da pilha será utilizada. Isto é feito através do registrador Stack Pointer.



```
MOV 2(SP),R6 ; Item I2 -> R6
MOV R7,0(SP) ; Overwrite TOS with R7
PUSH #0123h ; Put 0123h onto TOS
POP R8 ; R8 = 0123h
```

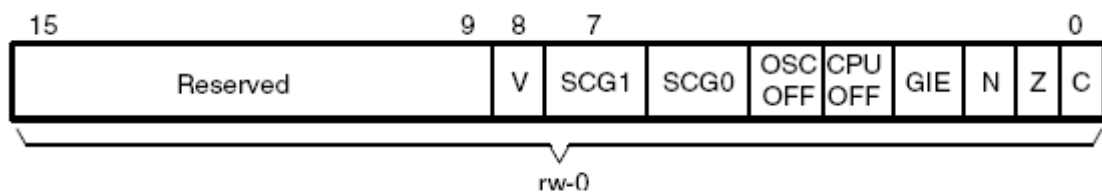


The stack pointer is changed after a PUSH SP instruction.

The stack pointer is not changed after a POP SP instruction. The POP SP instruction places SP₁ into the stack pointer SP (SP₂=SP₁)

Status Register (R2)

O registrador de status contém os bits de controle aritmético, atualizados a cada operação realizada pela CPU. Além disto é nele que são ajustados os bits de controlam os modos de operação em baixa potência, como pode ser visto abaixo.



A descrição bit a bit de funcionamento e suas respectivas configurações são mostradas na tabela a seguir.

Constant Generator (R3)

As seis constantes mais utilizadas durante toda a operação da CPU são geradas de modo automaticamente pelo gerador de constantes, que é composto pelos registradores R2 e R3, sem a necessidade de qualquer código adicional de programa. As constantes são selecionadas de acordo com o endereçamento do registrador fonte, como é mostrado na tabela a seguir.

BITS DO STATUS REGISTER

Bit	Description
V	<p>Overflow bit. This bit is set when the result of an arithmetic operation overflows the signed-variable range.</p> <p>ADD (.B) , ADDC (.B) Set when: Positive + Positive = Negative Negative + Negative = Positive, otherwise reset</p> <p>SUB (.B) , SUBC (.B) , CMP (.B) Set when: Positive - Negative = Negative Negative - Positive = Positive, otherwise reset</p>
SCG1	System clock generator 1. This bit, when set, turns off the DCO dc generator, if DCOCLK is not used for MCLK or SMCLK.
SCG0	System clock generator 0. This bit, when set, turns off the FLL+ loop control
OSCOFF	Oscillator Off. This bit, when set, turns off the LFXT1 crystal oscillator, when LFXT1CLK is not use for MCLK or SMCLK
CPUOFF	CPU off. This bit, when set, turns off the CPU.
GIE	General interrupt enable. This bit, when set, enables maskable interrupts. When reset, all maskable interrupts are disabled.
N	<p>Negative bit. This bit is set when the result of a byte or word operation is negative and cleared when the result is not negative.</p> <p>Word operation: N is set to the value of bit 15 of the result</p> <p>Byte operation: N is set to the value of bit 7 of the result</p>
Z	Zero bit. This bit is set when the result of a byte or word operation is 0 and cleared when the result is not 0.
C	Carry bit. This bit is set when the result of a byte or word operation produced a carry and cleared when no carry occurred.

CONSTANTES GERADAS AUTOMATICAMENTE

Register	As	Constant	Remarks
R2	00	-----	Register mode
R2	01	(0)	Absolute address mode
R2	10	00004h	+4, bit processing
R2	11	00008h	+8, bit processing
R3	00	00000h	0, word processing
R3	01	00001h	+1
R3	10	00002h	+2, bit processing
R3	11	0FFFFh	-1, word processing

São estas constantes que permitem que o set de instruções do MSP430, que tem originalmente apenas 27 instruções (RISC) possa ser expandido em mais 24, totalizando 51 instruções. Tudo isto sem a necessidade de acrescentar nenhuma outra linha de código.

Alguns exemplos de como isto funciona são mostrados abaixo:

INSTRUÇÃO PARA LIMPAR UM REGISTRADOR (CLR)

CLR dst

Isto é emulado em uma instrução de duplo operando, com o mesmo comprimento, fazendo:

MOV R3, dst ;onde R3 recebe o valor de #00

INSTRUÇÃO INCREMENTAR UM REGISTRADOR (CLR)

INC dst

Isto é emulado em uma instrução de duplo operando, com o mesmo comprimento, fazendo:

ADD 0(R3), dst ;onde R3 recebe o valor #01

General Purpose Registers (R4 – R15)

Os demais registradores conectados diretamente a CPU (R4 a R15) são de propósito geral, podendo ser utilizados para qualquer função desejada pelo usuário, como armazenamento de constantes ou valores, ponteiros de endereçamento, indexadores de valores, etc.

Alguns exemplos das operações que podem ser realizadas com estes registradores são mostradas na figura a seguir.



Example Register-Byte Operation

R5 = 0A28Fh
R6 = 0203h
Mem(0203h) = 012h

ADD .B R5, 0 (R6)

$$\begin{array}{r} 08Fh \\ + 012h \\ \hline 0A1h \end{array}$$

Mem (0203h) = 0A1h
C = 0, Z = 0, N = 1

$$\begin{array}{r} \text{(Low byte of register)} \\ + \text{(Addressed byte)} \\ \hline \text{--> (Addressed byte)} \end{array}$$

Example Byte-Register Operation

R5 = 01202h
R6 = 0223h
Mem(0223h) = 05Fh

ADD .B @R6, R5

$$\begin{array}{r} 05Fh \\ + 002h \\ \hline 00061h \end{array}$$

R5 = 00061h
C = 0, Z = 0, N = 0

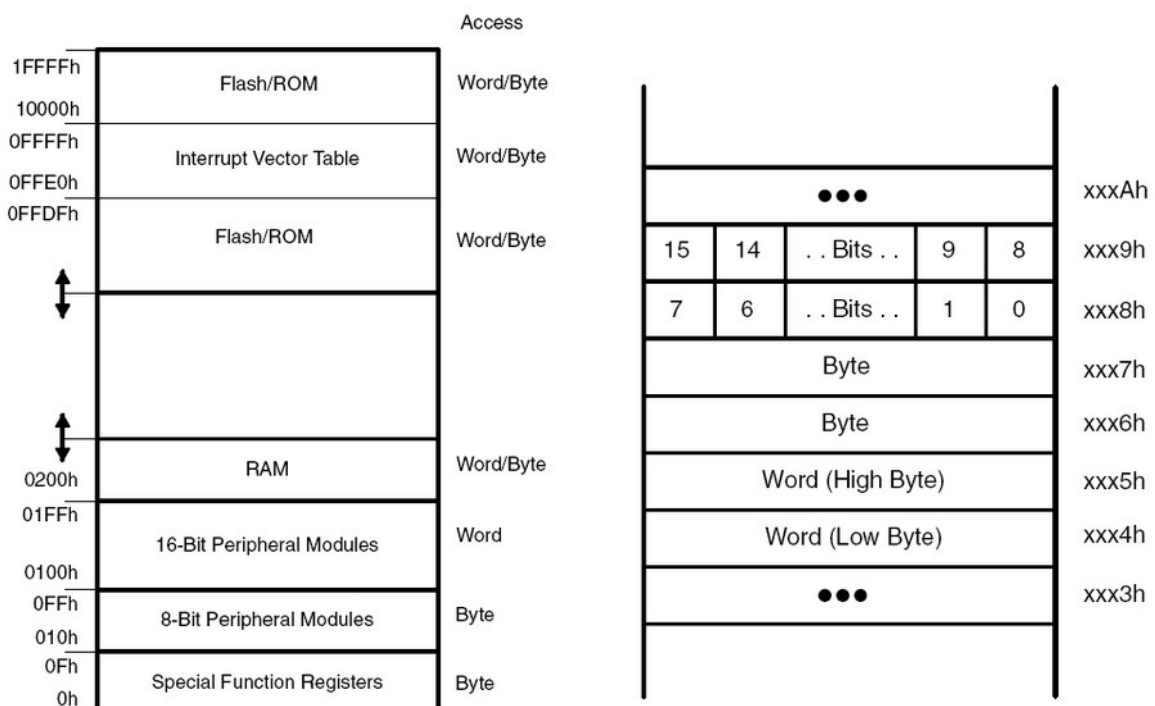
$$\begin{array}{r} \text{(Addressed byte)} \\ + \text{(Low byte of register)} \\ \hline \text{--> (Low byte of register, zero to High byte)} \end{array}$$

Estrutura de memórias: RAM, ROM (Flash)

Como pode ser visto no item 2 deste material, as memórias RAM, ROM (*Flash*) são dois trechos distintos dentro do hardware. Porém seu mapeamento é contínuo, incluindo os vetores de interrupção, de *reset*, periféricos e registradores com funções especiais.

Memórias no MSP430

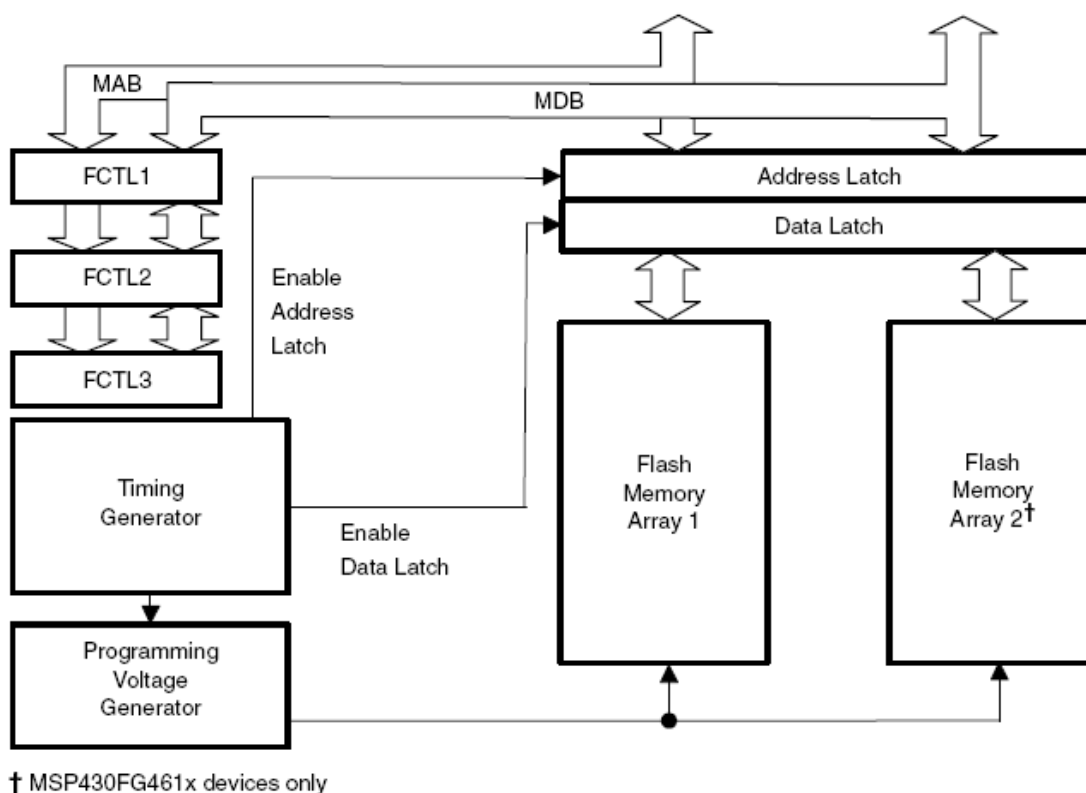
O mapeamento de memória no MSP430 é igual para as **famílias 2 e 4** e seguem a estrutura mostrada na figura a seguir.



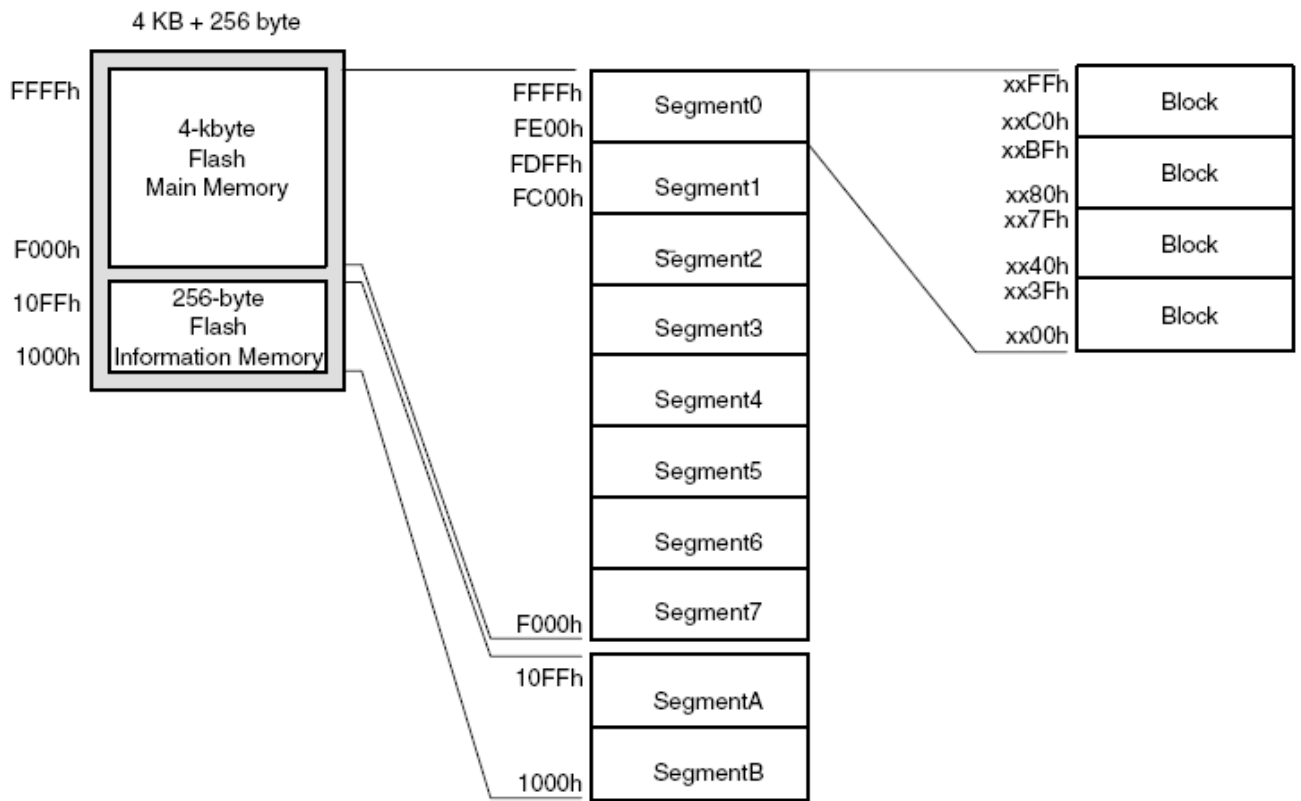
A quantidade de dados armazenada em cada espaço do mapa de memória é de 8 bits. Porém o MSP430 é um microcontrolador de 16 bits. Como resolver isto? O acesso ao mapa memória pode ser feito em Word (16 bits), byte (8 bits) ou em bit, como pode ser visto na figura a seguir.

Memórias de programa (ROM – FLASH)

O microcontrolador MSP430 armazena todo o programa que será executado em sua memória ROM/FLASH. Perceba que na arquitetura destes processadores não há memória do tipo EEPROM. Deste modo, dados permanentes, que não podem ser perdidos em caso de falta de energia elétrica, devem também ser armazenados na memória FLASH. O Hardware onde está a memória FLASH é mostrada na figura a seguir.



Toda a memória FLASH do MSP430 é particionada em segmentos. O processo de gravação pode acontecer em trechos de Words, Bytes ou mesmo bits. Mas para apagamento isto só pode ocorrer por segmentos completos. O particionamento para uma memória de 4 Kbytes é mostrado na figura abaixo. Isto sempre ocorre em segmentos, sendo que cada segmento é sub dividido em 4 blocos.



Reduced Instruction Set Code – RISC

O MSP430, apesar de ter uma arquitetura do tipo von-Neumann, trabalha com um set de instruções reduzido (RISC). São apenas 27 instruções físicas (core instructions). Com o uso dos registradores que geram constantes (R2 e R3) é possível emular mais 24 instruções, totalizando 51 instruções.

Basicamente as instruções são de três tipos:

- **Dual-operand**: dois operandos → fonte e destino;
- **Single-operand**: apenas um operando, que pode ser uma fonte ou um destino;
- **Jump**: instruções de salto no programa.

Todas as instruções de operando simples ou duplo podem ser escritas em bytes (8 bits) ou words (16 bits). Na construção das instruções sempre é seguida a seguinte nomenclatura:

- **src**: o operador fonte é definido por As e S-reg;
- **dst**: o operador destino é definido por Ad e D-reg;
- **As**: determina qual modo de endereçamento utilizado pela instrução, especificando quem é o registrador fonte;
- **S-reg**: quem, dos 16 registradores diretos à CPU, é utilizado como fonte;
- **Ad**: determina qual modo de endereçamento utilizado pela instrução, especificando quem é o registrador destino;
- **D-reg**: quem, dos 16 registradores diretos à CPU, é utilizado como destino;
- **B/W**: indica se a instrução é orientada a byte (1) ou a word (0);

As/Ad	Addressing Mode	Syntax	Description
00/0	Register mode	Rn	Register contents are operand
01/1	Indexed mode	X(Rn)	(Rn + X) points to the operand. X is stored in the next word.
01/1	Symbolic mode	ADDR	(PC + X) points to the operand. X is stored in the next word. Indexed mode X(PC) is used.
01/1	Absolute mode	&ADDR	The word following the instruction contains the absolute address. X is stored in the next word. Indexed mode X(SR) is used.
10/-	Indirect register mode	@Rn	Rn is used as a pointer to the operand.
11/-	Indirect autoincrement	@Rn+	Rn is used as a pointer to the operand. Rn is incremented afterwards by 1 for .B instructions and by 2 for .W instructions.
11/-	Immediate mode	#N	The word following the instruction contains the immediate constant N. Indirect autoincrement mode @PC+ is used.

Modos de endereçamento

Register Mode	<code>mov.w R10, R11</code> Single cycle
Indexed Mode	<code>mov.w 2(R5), 6(R6)</code> Table processing
Symbolic Mode	<code>mov.w EDE, TONI</code> Easy to read code, PC relative
Absolute Mode	<code>mov.w &EDE, &TONI</code> Directly access any memory location
Indirect Register Mode	<code>mov.w @R10, 0(R11)</code> Access memory with pointers
Indirect Autoincrement	<code>mov.w @R10+, 0(R11)</code> Table processing
Immediate Mode	<code>mov.w #45h, &TONI</code> Unrestricted constant values



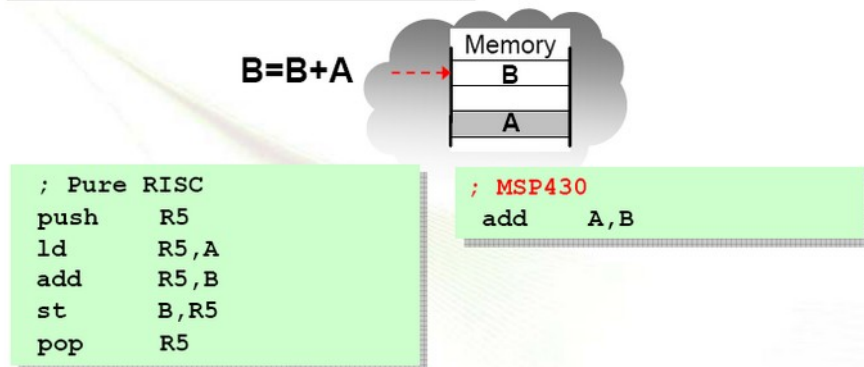
Atomic

Table 2. Address Mode Descriptions

ADDRESS MODE	S	D	SYNTAX	EXAMPLE	OPERATION
Register	●	●	MOV Rs,Rd	MOV R10,R11	R10 --> R11
Indexed	●	●	MOV X(Rn),Y(Rm)	MOV 2(R5),6(R6)	M(2+R5)--> M(6+R6)
Symbolic (PC relative)	●	●	MOV EDE,TONI		M(EDE) --> M(TONI)
Absolute	●	●	MOV &MEM,&TCDAT		M(MEM) --> M(TCDAT)
Indirect	●		MOV @Rn,Y(Rm)	MOV @R10,Tab(R6)	M(R10) --> M(Tab+R6)
Indirect autoincrement	●		MOV @Rn+,Rm	MOV @R10+,R11	M(R10) --> R11 R10 + 2--> R10
Immediate	●		MOV #X,TONI	MOV #45,TONI	#45 --> M(TONI)

NOTE: S = source D = destination

Atomic Addressing



Formatos das instruções

Three Instruction Formats

Op-Code	S-Register	Ad	B/W	As	D-Register
; Format I Source and Destination					
5405	add.w	R4,R5			; R4+R5=R5 xxxx
5445	add.b	R4,R5			; R4+R5=R5 00xx

Op-Code	B/W	As	D/S-Register
; Format II Destination Only			
6404	rlc.w	R4	
6444	rlc.b	R4	

Op-Code	Condition	10-bit PC offset
; Format III There are 8 (Un)conditional Jumps		
3C28	jmp	Loop_1 ; Goto Loop_1

As 51 instruções

SOURCE AND DESTINATION INSTRUCTIONS

Mnemonic	S-Reg, D-Reg	Operation	Status Bits			
			V	N	Z	C
MOV (.B)	src, dst	src → dst	–	–	–	–
ADD (.B)	src, dst	src + dst → dst	*	*	*	*
ADDC (.B)	src, dst	src + dst + C → dst	*	*	*	*
SUB (.B)	src, dst	dst + .not.src + 1 → dst	*	*	*	*
SUBC (.B)	src, dst	dst + .not.src + C → dst	*	*	*	*
CMP (.B)	src, dst	dst – src	*	*	*	*
DADD (.B)	src, dst	src + dst + C → dst (decimally)	*	*	*	*
BIT (.B)	src, dst	src .and. dst	0	*	*	*
BIC (.B)	src, dst	.not.src .and. dst → dst	–	–	–	–
BIS (.B)	src, dst	src .or. dst → dst	–	–	–	–
XOR (.B)	src, dst	src .xor. dst → dst	*	*	*	*
AND (.B)	src, dst	src .and. dst → dst	0	*	*	*

- * The status bit is affected
- The status bit is not affected
- 0 The status bit is cleared
- 1 The status bit is set

Note: Instructions CMP and SUB

The instructions `CMP` and `SUB` are identical except for the storage of the result. The same is true for the `BIT` and `AND` instructions.

DESTINATION ONLY INSTRUCTIONS

Mnemonic	S-Reg, D-Reg	Operation	Status Bits			
			V	N	Z	C
RRC (.B)	dst	C → MSB →.....LSB → C	*	*	*	*
RRA (.B)	dst	MSB → MSB →....LSB → C	0	*	*	*
PUSH (.B)	src	SP - 2 → SP, src → @SP	-	-	-	-
SWPB	dst	Swap bytes	-	-	-	-
CALL	dst	SP - 2 → SP, PC+2 → @SP dst → PC	-	-	-	-
RETI		TOS → SR, SP + 2 → SP TOS → PC, SP + 2 → SP	*	*	*	*
SXT	dst	Bit 7 → Bit 8.....Bit 15	0	*	*	*

- * The status bit is affected
- The status bit is not affected
- 0 The status bit is cleared
- 1 The status bit is set

All addressing modes are possible for the `CALL` instruction. If the symbolic mode (`ADDRESS`), the immediate mode (`#N`), the absolute mode (`&EDE`) or the indexed mode `x(RN)` is used, the word that follows contains the address information.

JUMPS INSTRUCTIONS

Mnemonic		Description		V	N	Z	C
ADC(.B)†	dst	Add C to destination	dst + C → dst	*	*	*	*
ADD(.B)	src, dst	Add source to destination	src + dst → dst	*	*	*	*
ADDC(.B)	src, dst	Add source and C to destination	src + dst + C → dst	*	*	*	*
AND(.B)	src, dst	AND source and destination	src .and. dst → dst	0	*	*	*
BIC(.B)	src, dst	Clear bits in destination	.not.src .and. dst → dst	-	-	-	-
BIS(.B)	src, dst	Set bits in destination	src .or. dst → dst	-	-	-	-
BIT(.B)	src, dst	Test bits in destination	src .and. dst	0	*	*	*
BR†	dst	Branch to destination	dst → PC	-	-	-	-
CALL	dst	Call destination	PC+2 → stack, dst → PC	-	-	-	-
CLR(.B)†	dst	Clear destination	0 → dst	-	-	-	-
CLRC†		Clear C	0 → C	-	-	-	0
CLRN†		Clear N	0 → N	-	0	-	-
CLRZ†		Clear Z	0 → Z	-	-	0	-
CMP(.B)	src, dst	Compare source and destination	dst - src	*	*	*	*
DADC(.B)†	dst	Add C decimally to destination	dst + C → dst (decimally)	*	*	*	*
DADD(.B)	src, dst	Add source and C decimally to dst.	src + dst + C → dst (decimally)	*	*	*	*
DEC(.B)†	dst	Decrement destination	dst - 1 → dst	*	*	*	*
DECD(.B)†	dst	Double-decrement destination	dst - 2 → dst	*	*	*	*
DINT†		Disable interrupts	0 → GIE	-	-	-	-
EINT†		Enable interrupts	1 → GIE	-	-	-	-
INC(.B)†	dst	Increment destination	dst + 1 → dst	*	*	*	*
INCD(.B)†	dst	Double-increment destination	dst + 2 → dst	*	*	*	*
INV(.B)†	dst	Invert destination	.not.dst → dst	*	*	*	*
JC/JHS	label	Jump if C set/Jump if higher or same		-	-	-	-
JEQ/JZ	label	Jump if equal/Jump if Z set		-	-	-	-
JGE	label	Jump if greater or equal		-	-	-	-
JL	label	Jump if less		-	-	-	-
JMP	label	Jump	PC + 2 x offset → PC	-	-	-	-
JN	label	Jump if N set		-	-	-	-
JNC/JLO	label	Jump if C not set/Jump if lower		-	-	-	-
JNE/JNZ	label	Jump if not equal/Jump if Z not set		-	-	-	-
MOV(.B)	src, dst	Move source to destination	src → dst	-	-	-	-
NOP†		No operation		-	-	-	-
POP(.B)†	dst	Pop item from stack to destination	@SP → dst, SP+2 → SP	-	-	-	-
PUSH(.B)	src	Push source onto stack	SP - 2 → SP, src → @SP	-	-	-	-
RET†		Return from subroutine	@SP → PC, SP + 2 → SP	-	-	-	-
RETI		Return from interrupt		*	*	*	*
RLA(.B)†	dst	Rotate left arithmetically		*	*	*	*
RLC(.B)†	dst	Rotate left through C		*	*	*	*
RRA(.B)	dst	Rotate right arithmetically		0	*	*	*
RRC(.B)	dst	Rotate right through C		*	*	*	*
SBC(.B)†	dst	Subtract not(C) from destination	dst + 0FFFFh + C → dst	*	*	*	*
SETC†		Set C	1 → C	-	-	-	1
SETN†		Set N	1 → N	-	1	-	-
SETZ†		Set Z	1 → C	-	-	1	-
SUB(.B)	src, dst	Subtract source from destination	dst + .not.src + 1 → dst	*	*	*	*
SUBC(.B)	src, dst	Subtract source and not(C) from dst.	dst + .not.src + C → dst	*	*	*	*
SWPB	dst	Swap bytes		-	-	-	-
SXT	dst	Extend sign		0	*	*	*
TST(.B)†	dst	Test destination	dst + 0FFFFh + 1	0	*	*	1
XOR(.B)	src, dst	Exclusive OR source and destination	src .xor. dst → dst	*	*	*	*

† Emulated Instruction

Mnemonic	S-Reg, D-Reg	Operation
JEQ/JZ	Label	Jump to label if zero bit is set
JNE/JNZ	Label	Jump to label if zero bit is reset
JC	Label	Jump to label if carry bit is set
JNC	Label	Jump to label if carry bit is reset
JN	Label	Jump to label if negative bit is set
JGE	Label	Jump to label if (N .XOR. V) = 0
JL	Label	Jump to label if (N .XOR. V) = 1
JMP	Label	Jump to label unconditionally

Conditional jumps support program branching relative to the PC and do not affect the status bits. The possible jump range is from –511 to +512 words relative to the PC value at the jump instruction. The 10-bit program-counter offset is treated as a signed 10-bit value that is doubled and added to the program counter:

$$PC_{\text{new}} = PC_{\text{old}} + 2 + PC_{\text{offset}} \times 2$$

Ciclos de máquina e tamanho das instruções

INTERRUPÇÕES E RESET

Action	No. of Cycles	Length of Instruction
Return from interrupt (RETI)	5	1
Interrupt accepted	6	–
WDT reset	4	–
Reset ($\overline{\text{RST}}$ /NMI)	4	–

INSTRUÇÕES DE FORMATO II (DESTINATION ONLY)

Addressing Mode	No. of Cycles			Length of Instruction	Example
	RRA, RRC SWPB, SXT	PUSH	CALL		
Rn	1	3	4	1	SWPB R5
@Rn	3	4	4	1	RRC @R9
@Rn+	3	5	5	1	SWPB @R10+
#N	(See note)	4	5	2	CALL #0F00h
X(Rn)	4	5	5	2	CALL 2 (R7)
EDE	4	5	5	2	PUSH EDE
&EDE	4	5	5	2	SXT &EDE

Note: Instruction Format II Immediate Mode

Do not use instructions RRA, RRC, SWPB, and SXT with the immediate mode in the destination field. Use of these in the immediate mode results in an unpredictable program operation.

INSTRUÇÕES DE FORMATO III (JUMP)

Todas as instruções desta categoria necessitam de **um Word de comprimento e dois ciclos de máquina** para serem executadas, independente da ação de salto acontecer ou não.

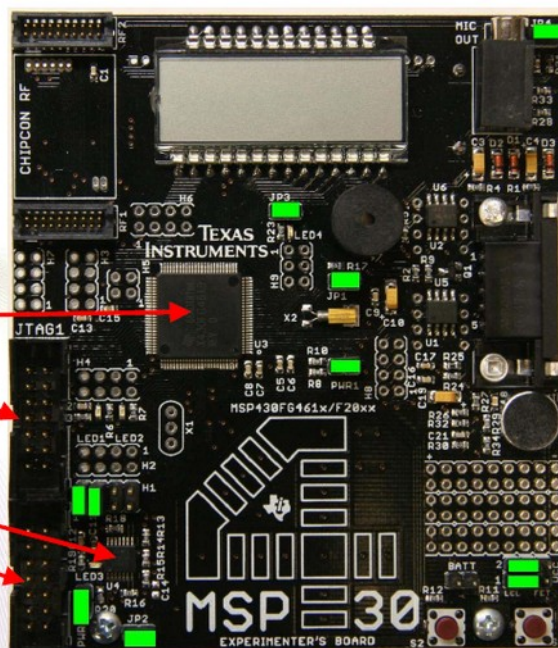
INSTRUÇÕES DE FORMATO I (SOURCE AND DESTINATION)

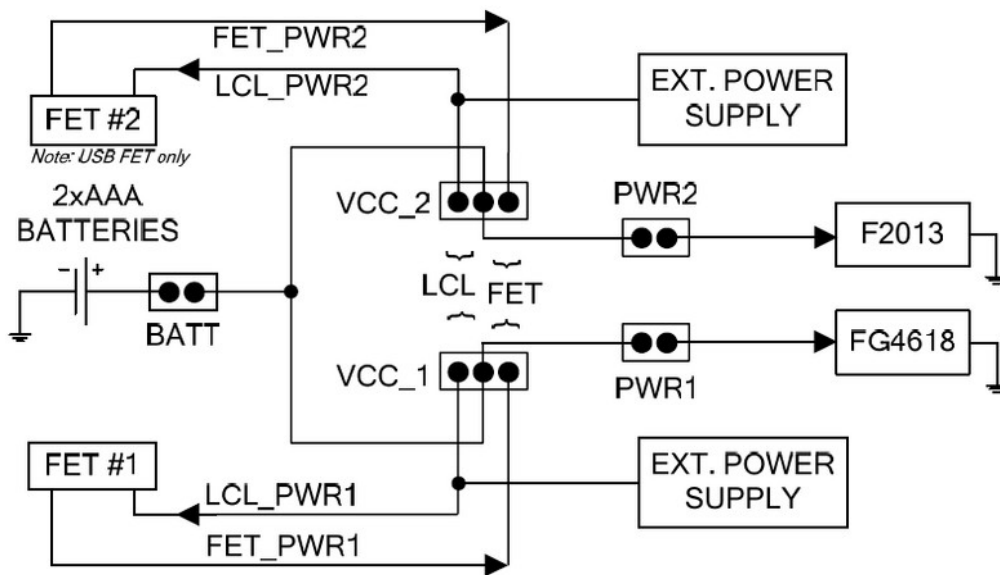
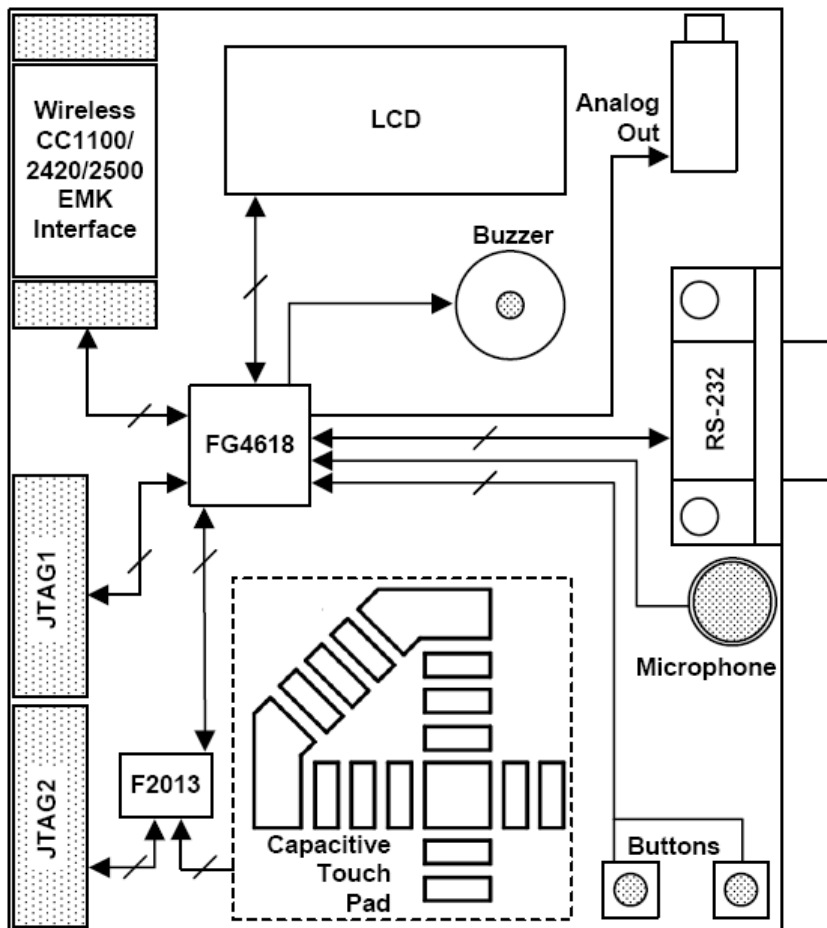
Addressing Mode		No. of Cycles	Length of Instruction		Example
Src	Dst				
Rn	Rm	1	1	MOV	R5 , R8
	PC	2	1	BR	R9
	x(Rm)	4	2	ADD	R5 , 4 (R6)
	EDE	4	2	XOR	R8 , EDE
	&EDE	4	2	MOV	R5 , &EDE
@Rn	Rm	2	1	AND	@R4 , R5
	PC	2	1	BR	@R8
	x(Rm)	5	2	XOR	@R5 , 8 (R6)
	EDE	5	2	MOV	@R5 , EDE
	&EDE	5	2	XOR	@R5 , &EDE
@Rn+	Rm	2	1	ADD	@R5+ , R6
	PC	3	1	BR	@R9+
	x(Rm)	5	2	XOR	@R5 , 8 (R6)
	EDE	5	2	MOV	@R9+ , EDE
	&EDE	5	2	MOV	@R9+ , &EDE
#N	Rm	2	2	MOV	#20 , R9
	PC	3	2	BR	#2AEh
	x(Rm)	5	3	MOV	#0300h , 0 (SP)
	EDE	5	3	ADD	#33 , EDE
	&EDE	5	3	ADD	#33 , &EDE
x(Rn)	Rm	3	2	MOV	2 (R5) , R7
	PC	3	2	BR	2 (R6)
	TONI	6	3	MOV	4 (R7) , TONI
	x(Rm)	6	3	ADD	4 (R4) , 6 (R9)
	&TONI	6	3	MOV	2 (R4) , &TONI
EDE	Rm	3	2	AND	EDE , R6
	PC	3	2	BR	EDE
	TONI	6	3	CMP	EDE , TONI
	x(Rm)	6	3	MOV	EDE , 0 (SP)
	&TONI	6	3	MOV	EDE , &TONI
&EDE	Rm	3	2	MOV	&EDE , R8
	PC	3	2	BRA	&EDE
	TONI	6	3	MOV	&EDE , TONI
	x(Rm)	6	3	MOV	&EDE , 0 (SP)
	&TONI	6	3	MOV	&EDE , &TONI

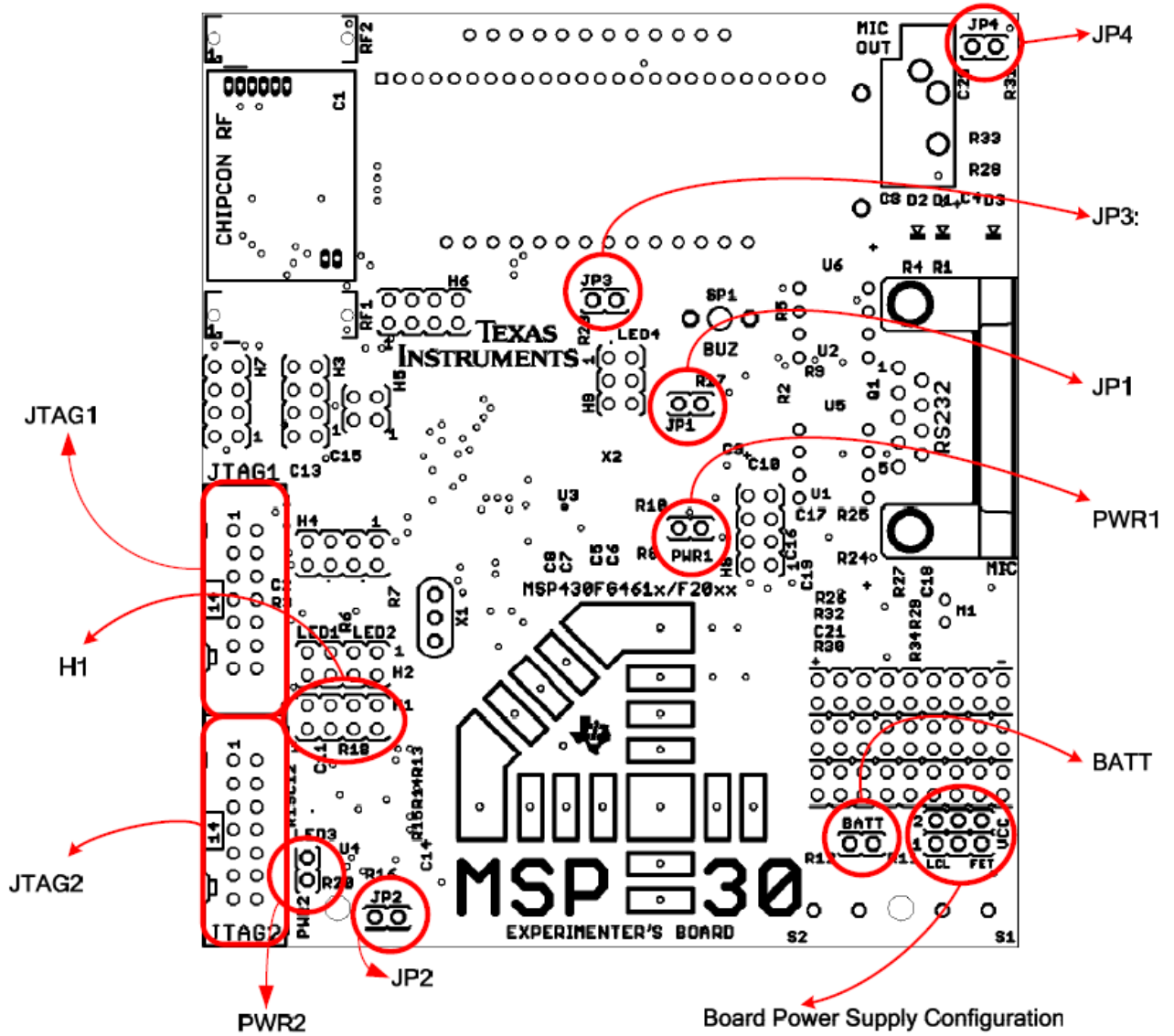
A Placa de exercícios: Experimenter Board

ATC Board

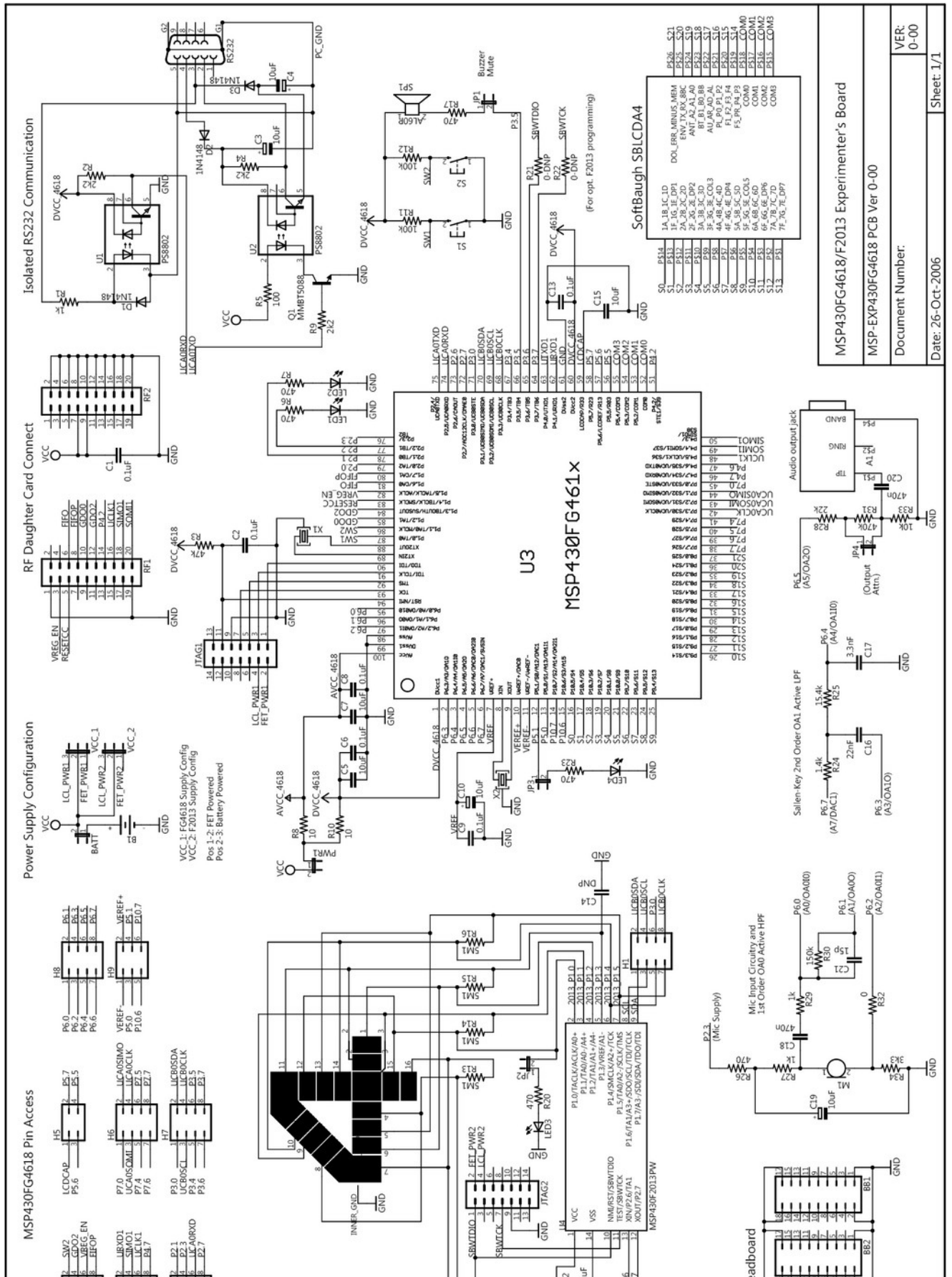
- Default jumper settings
- **MSP430FG4619**
 - '4619 JTAG
- **MSP430F2013**
 - '2013 JTAG







Header	Functionality when jumper present	Functionality when jumper absent	Requirement
PWR1	Provides power to MSP430FG4618 Also used to measure current consumption of the MSP430FG4618	MSP430FG4618 is not powered	Required for MSP430FG4618 use
PWR2	Provides power to MSP430F2013 Also used to measure current consumption of the MSP430F2013	MSP430F2013 is not powered	Required for MSP430F2013 use
BATT	On-board batteries provide power Also used to measure current consumption	Batteries will not provide power to either MSP430	Required for use with AAA batteries
JP1	Buzzer enabled and connected to P3.5 of the MSP430FG4618	Buzzer muted	Optional
JP2	LED3 enabled and connected to P1.0 of the MSP430F2013	LED3 connection disabled	Optional / Required for LED3 use
JP3	LED4 enabled and connected to P5.1 of MSP430FG4618	LED4 connection disabled	Optional / Required for LED4 use
JP4	Attenuation set to approximately 69% of the DAC12 audio output	98% attenuation of the DAC12 audio output	Optional
Header H1 (Pins 1-2, 3-4)	I2C Configuration 1-2: SDA – UCB0SDA 3-4: SCL – UCB0SCL	No communication possible via I2C	Required for inter-processor communication
Header H1 (Pins 1-2, 3-4, 5-6, 7-8)	SPI Configuration 1-2: SDI – UCB0SIMO 3-4: SDO – UCB0SOMI 5-6: P1.4 – P3.0 (CS) 7-8: SCLK – UCB0CLK	No communication possible via SPI	Required for inter-processor communication



MSP430FG4618/F2013 Experimenter's Board
MSP-EXP430FG4618 PCB Ver 0-00
Document Number:
Date: 26-Oct-2006

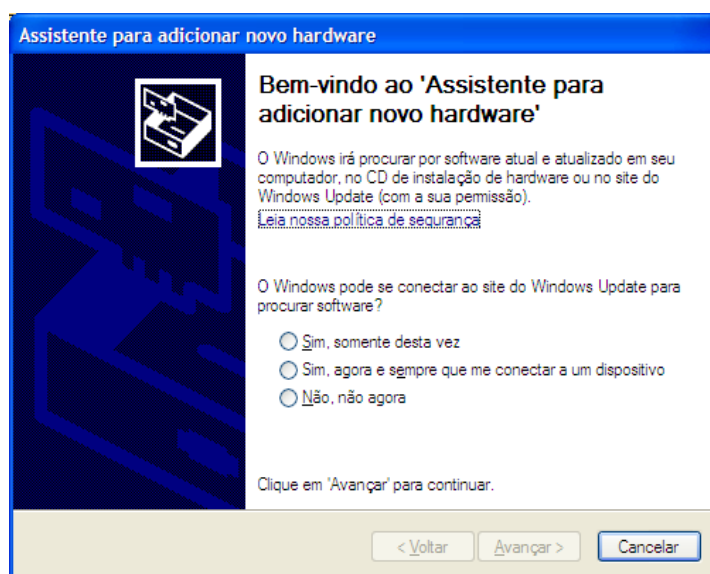
VER: 0-00

Sheet: 1/1

Instalando os drives do gravador FET USB

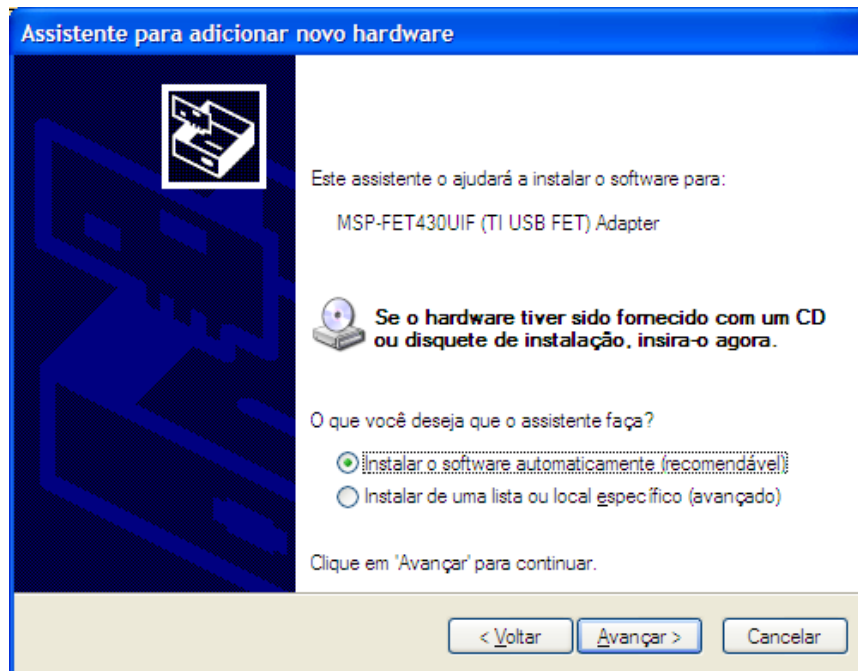


Ao colocar o FET numa porta USB, o Windows XP abrirá a tela abaixo, solicitando a instalação do driver. Para tanto o IAR já deve estar instalado.

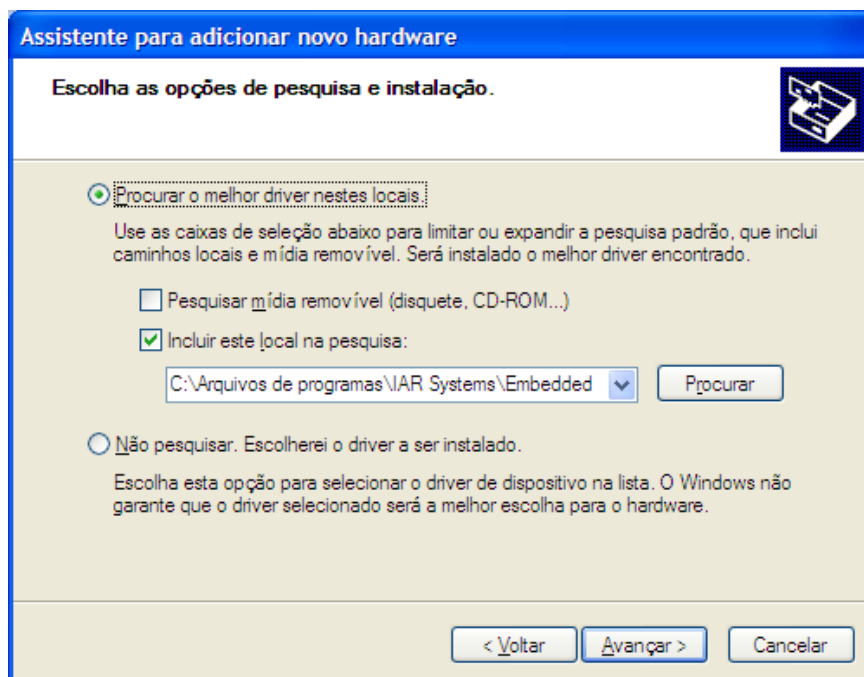


Deve-se clicar em **NÃO, NÃO AGORA**, e em seguida em **AVANÇAR**.

O próximo passo é indicar onde estão os arquivos do driver do FET para o MSP430. Assim deve-se selecionar a opção **INSTALAR DE UMA LISTA OU LOCAL ESPECÍFICO** na tela abaixo, clicando em seguida no botão **AVANÇAR**.



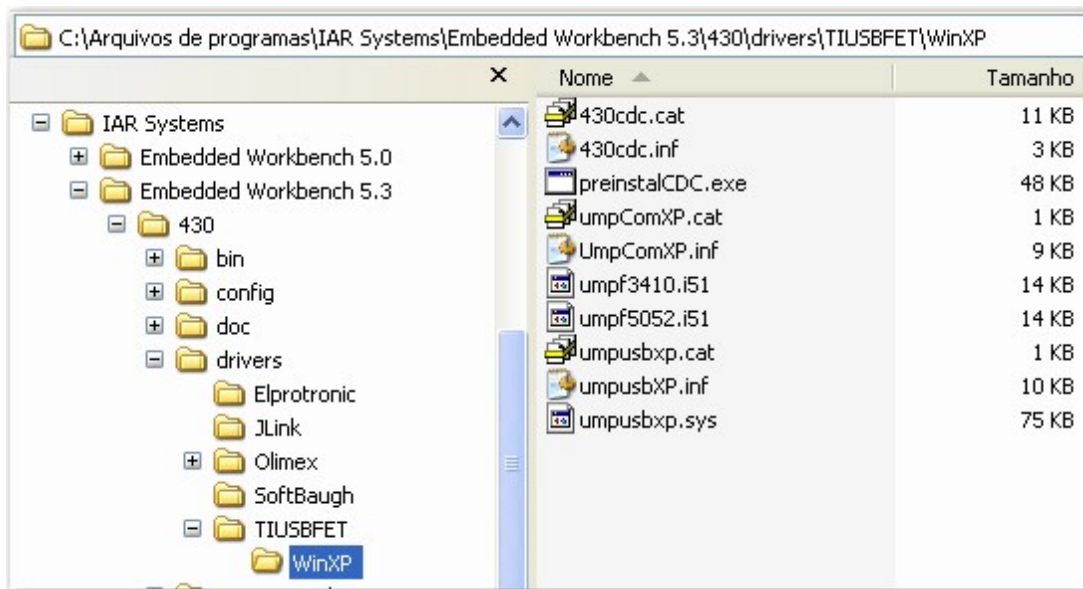
Ao clicar em avançar, clique no combo Box **INCLUIR ESTE LOCAL NA PERQUISA**, e clique no botão procurar:



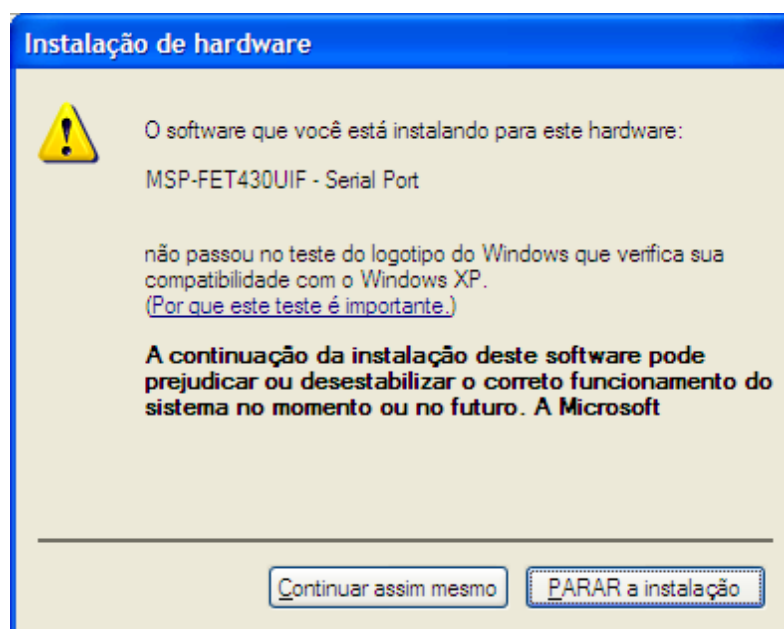
O endereço a ser selecionado é:

C:\Arquivos de programas\IAR Systems\Embedded Workbench5.3\430\drivers\
TIUSBFET\WinXP

Como pode ser visto na figura a seguir.

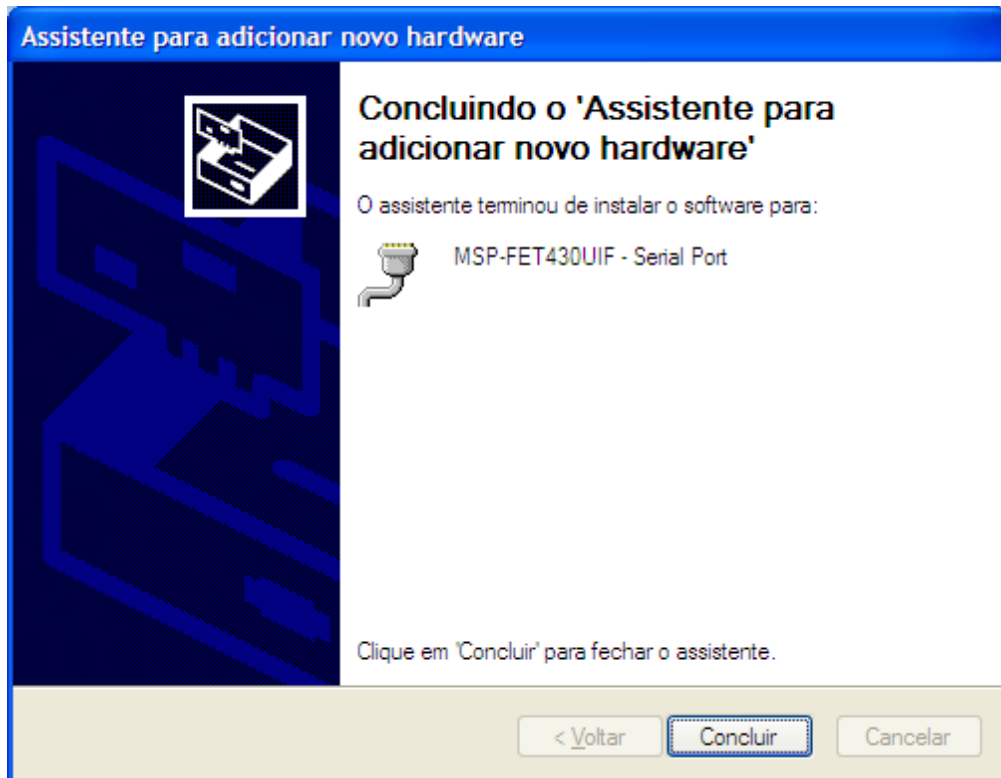


Com o caminho selecionado, clique em **AVANÇAR**. Aparecerá a janela de confirmação de instalação, como é mostrado a seguir:



Deve-se então clicar em **CONTINUAR ASSIM MESMO**.

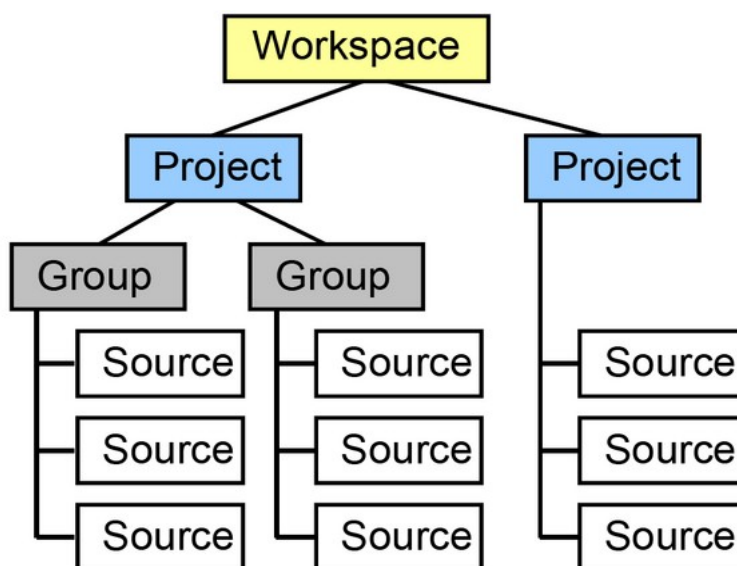
Isto deve finalizar a instalação do TI FET USB, como pode ser visto na figura a seguir.



Ao clicar em **CONCLUIR**, a instalação do **TI FET USB** será finalizada. Mas isto iniciará o procedimento de instalação da **PORTA SERIAL**. Os passos a serem executados são exatamente os mesmos, como foi feito para o **TI FET USB**.

AMBIENTE IAR

O IAR é um ambiente IDE completo e trabalha com o conceito de área de trabalho (Workspace) que pode conter diversos projetos (Project).

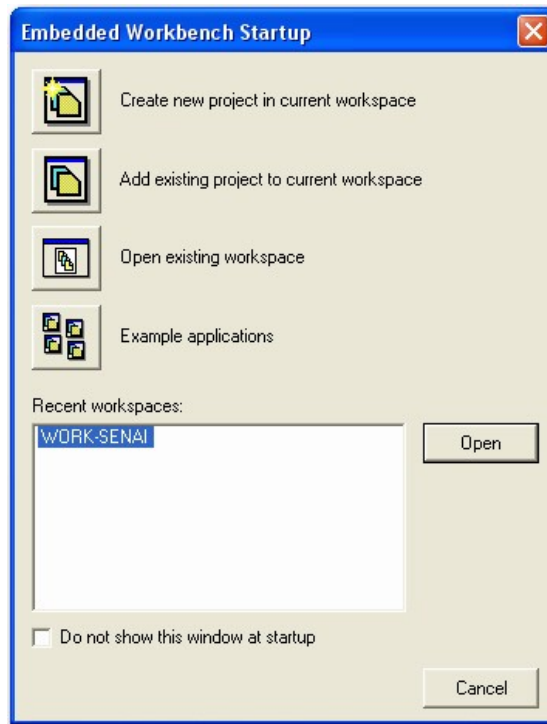


Esta organização será mantida em nossos estudos, facilitando a portabilidade de código e projetos de um computador para outro.

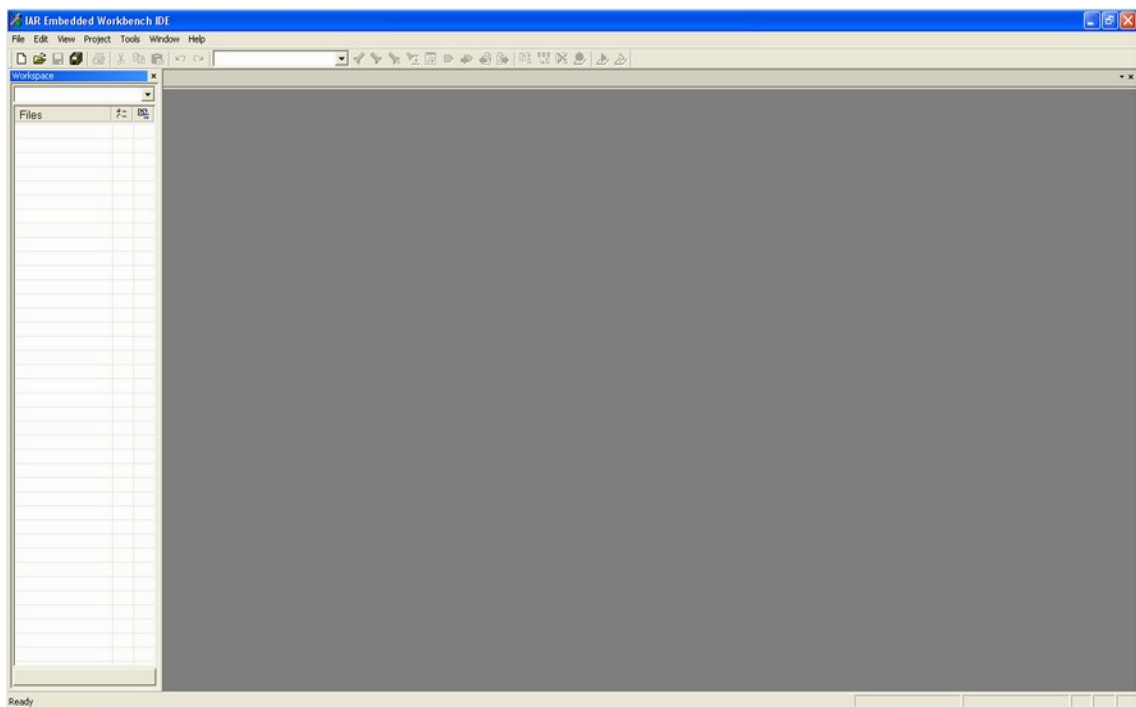
Começando do zero: Criando um novo projeto

Abra o IAR. A primeira tela que será mostrada é para a seleção de uma ação (criar um novo projeto, adicionar um projeto, abrir uma área de trabalho ou abrir uma pasta com exemplos), como pode ser visto na figura a seguir.

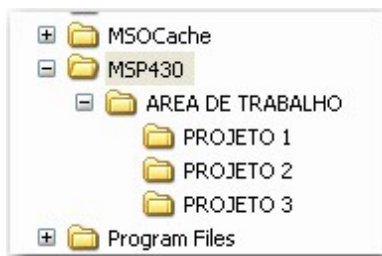
Como a ideia aqui será começar do zero, partindo do pressuposto que não há projetos anteriores, então clicaremos em cancelar nesta janela.



Assim, ao clicar em cancelar na janela anterior, o ambiente de trabalho do IAR se abrirá, porém nenhum projeto ou área de trabalho estará ativa, como se pode ver na imagem abaixo:

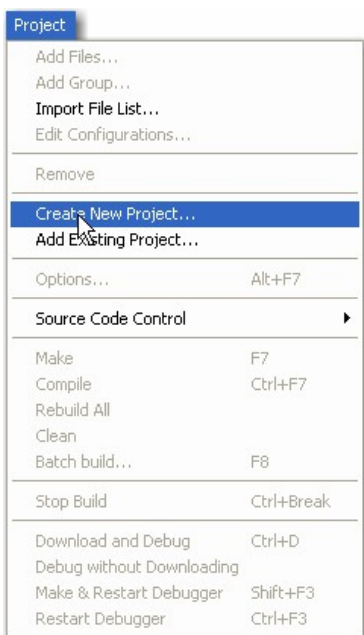


Para manter a estrutura de área de trabalho e projetos, é necessário primeiramente criar um projeto e em seguida salvá-lo em alguma área de trabalho. Por isto é extremamente interessante manter uma estrutura de pastas no computador que seja coerente com a organização feita pelo IAR. Veja o exemplo mostrado a seguir:



Na raiz do disco rígido foi criada uma pasta chamada **MSP430**. Dentro dela uma pasta chamada **ÁREA DE TRABALHO**. É dentro desta pasta que será salva a área de trabalho (workspace) do IAR. Dentro desta pasta serão criadas tantas pastas quanto forem os projetos a serem criados. Para facilitar o entendimento, criamos, inicialmente, três pastas de projeto, respectivamente **PROJETO 1**, **PROJETO 2** e **PROJETO 3**.

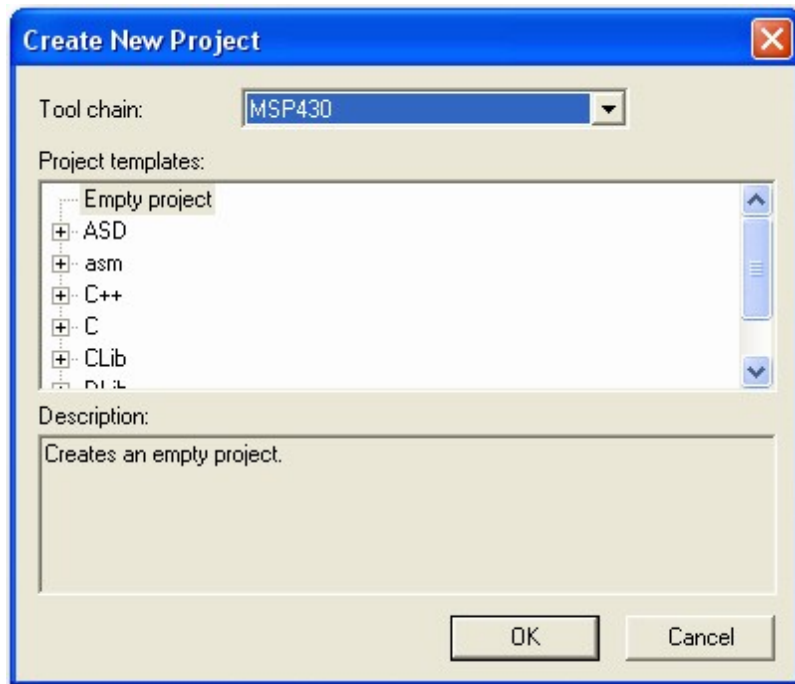
Para que um código e um projeto seja portátil de um micro para outro (você faz um exercício no computador da escola e quer levá-lo para o micro da empresa ou de sua casa e quer executá-lo sem erros) é necessário que esta estrutura de pastas seja mantida em todos os locais onde será executado. Mudanças de encaminhamento geram erros de compilação no IAR.



Uma vez que a estrutura de pastas está pronta, temos que criar um novo projeto para em seguida salvá-lo em uma área de trabalho.

Começemos pelo projeto. Isto é feito no **MENU** principal. Lá escolha a opção **PROJECT** e clique em seguida em **CREATE NEW PROJECT**, como mostra a figura a seguir.

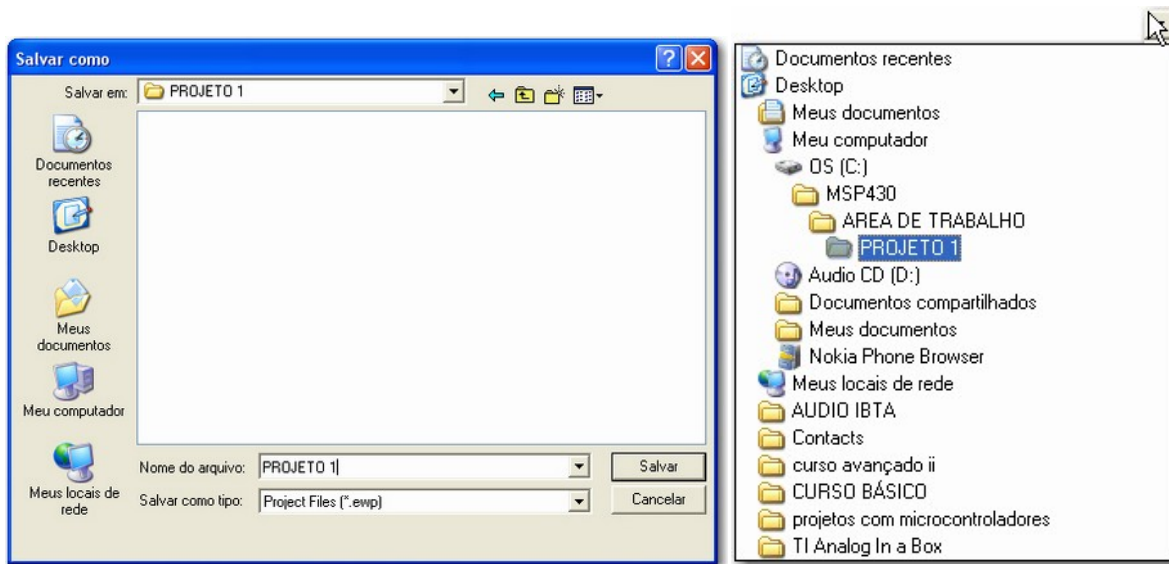
A janela para criação de projetos aparecerá, como mostrado a seguir:



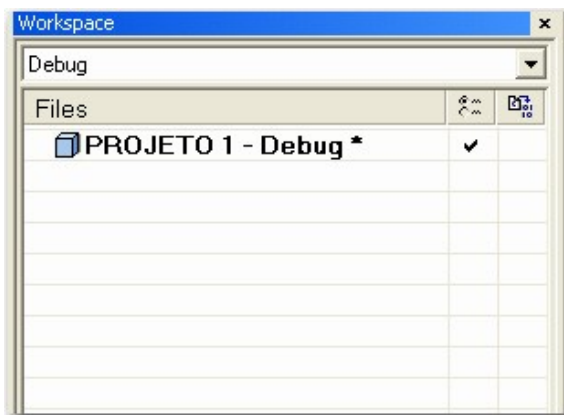
Deste modo o usuário pode optar por começar um projeto a partir de diversos modelos (*templates*) que a própria IAR fornece, seja para linguagem C ou Assembly.

Como nosso objetivo é partir do zero, neste primeiro momento não faremos uso de nenhuma *template* pronto. Para isto selecionaremos a opção de criar um **EMPTY PROJECT** (projeto vazio).

Após esta seleção se abria uma janela pedindo que aquele projeto seja salvo, como mostra a figura a seguir. A extensão de um projeto sempre é **.ewp** e não precisa ser digitada, pois o software faz o complemento de extensão automaticamente. Não se esqueça de selecionar o local onde o projeto será salvo de acordo com a organização de pastas mostradas anteriormente, antes de clicar em salvar, como mostra a figura a seguir.



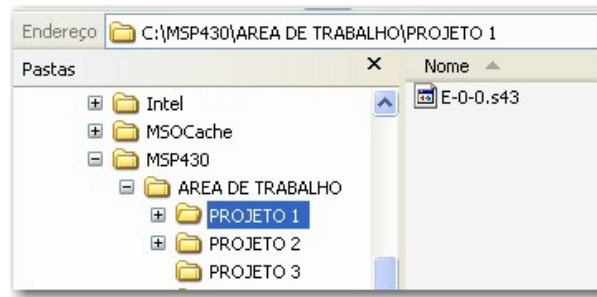
A partir do momento em que o projeto está salvo ele aparecerá na área de trabalho, como mostra a figura a seguir.



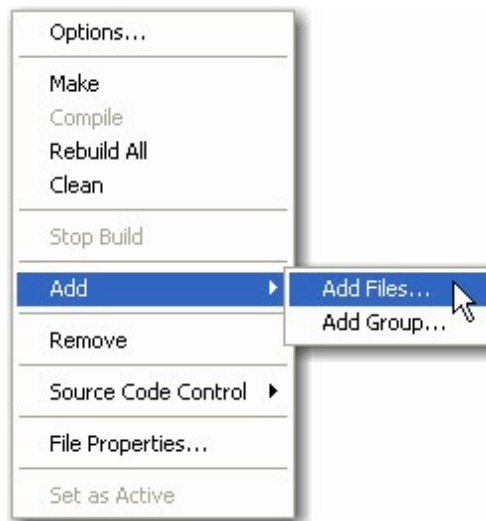
Note que abaixo da caixa azul que representa o projeto não há nenhum arquivo associado. Portanto faz-se necessário associar um arquivo de código (neste caso será um arquivo de linguagem de máquina Assembly, cuja extensão adotada pelo IAR é a **.s43**).

Podemos escrever um arquivo a partir do zero clicando no menu **FILE** e em seguida **NEW** e escolhendo a opção **FILE (ctrl + N)**. Isto fará aparecer um arquivo em branco onde você digitará seu código.

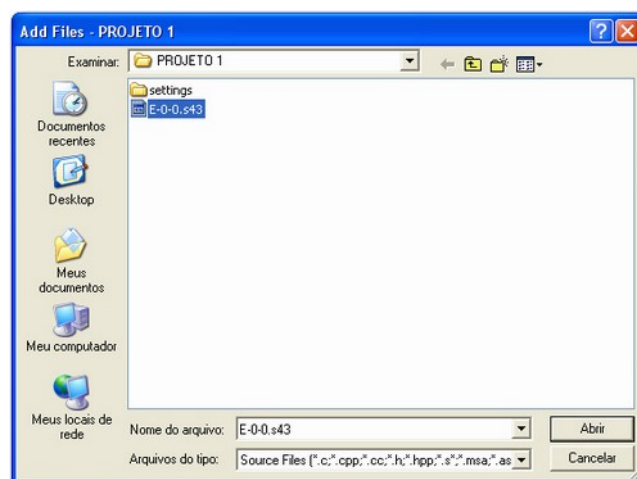
Para este primeiro projeto faremos outra opção. Associaremos a este projeto um arquivo Assembly já existente. Usaremos o exemplo **E-0-0.s43**. Para isto é necessário salvar uma cópia do arquivo dentro do diretório do projeto, como pode ser visto na figura a seguir:



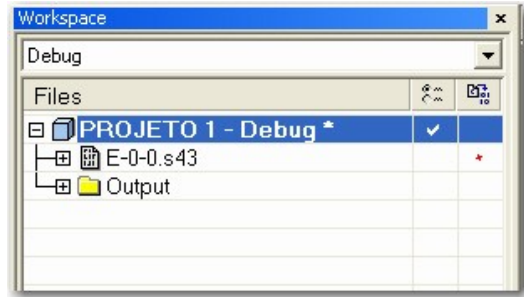
Com o arquivo **.s43** já colocado dentro da pasta do projeto, voltamos a área de trabalho do IAR e clicamos com o botão direito sobre o quadrado azul que representa o projeto criado. Isto fará aparecer um menu de opção de projeto, como mostrado na figura abaixo:



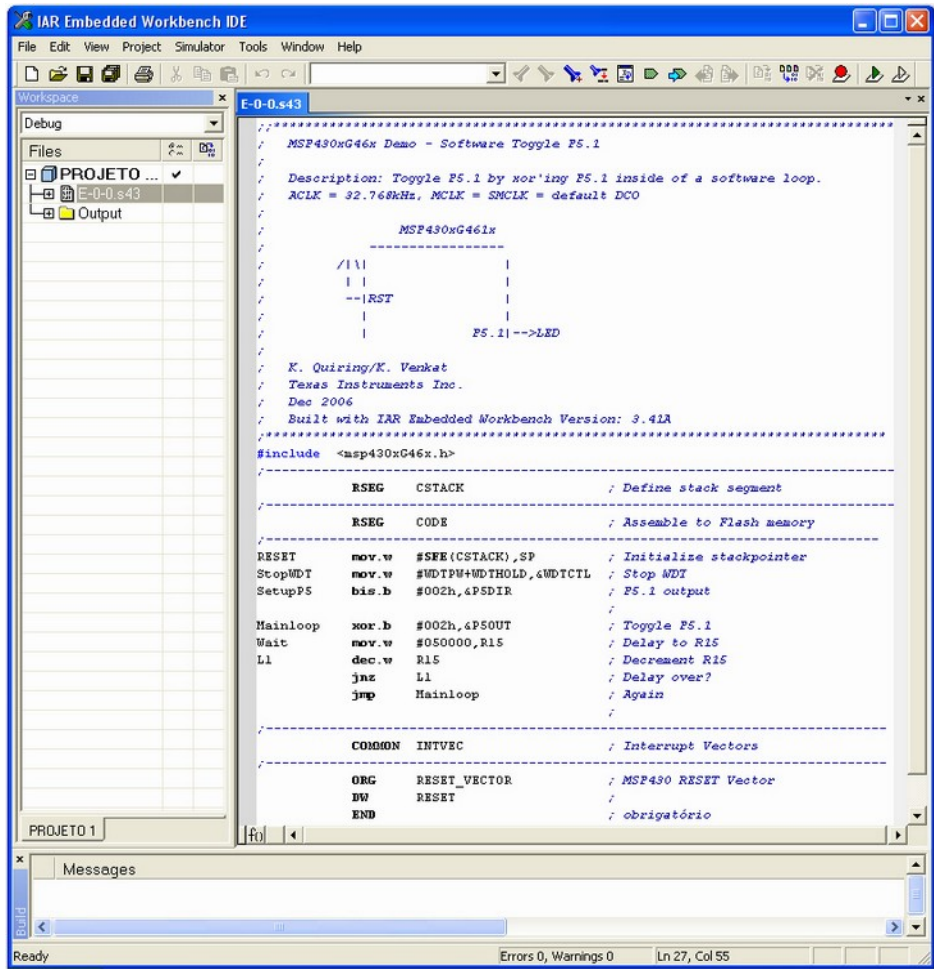
Neste menu selecionaremos a opção **ADD** e em seguida **ADD FILES**, como pode ser visto na figura acima. Deste modo será possível associar um arquivo Assembly (**.s43**) ao projeto já existente, como mostra a figura a seguir.



Com isto a área de trabalho, onde está o projeto, muda de figura, informando que determinado arquivo de código está associado a um determinado projeto, como se pode ver na figura a seguir.



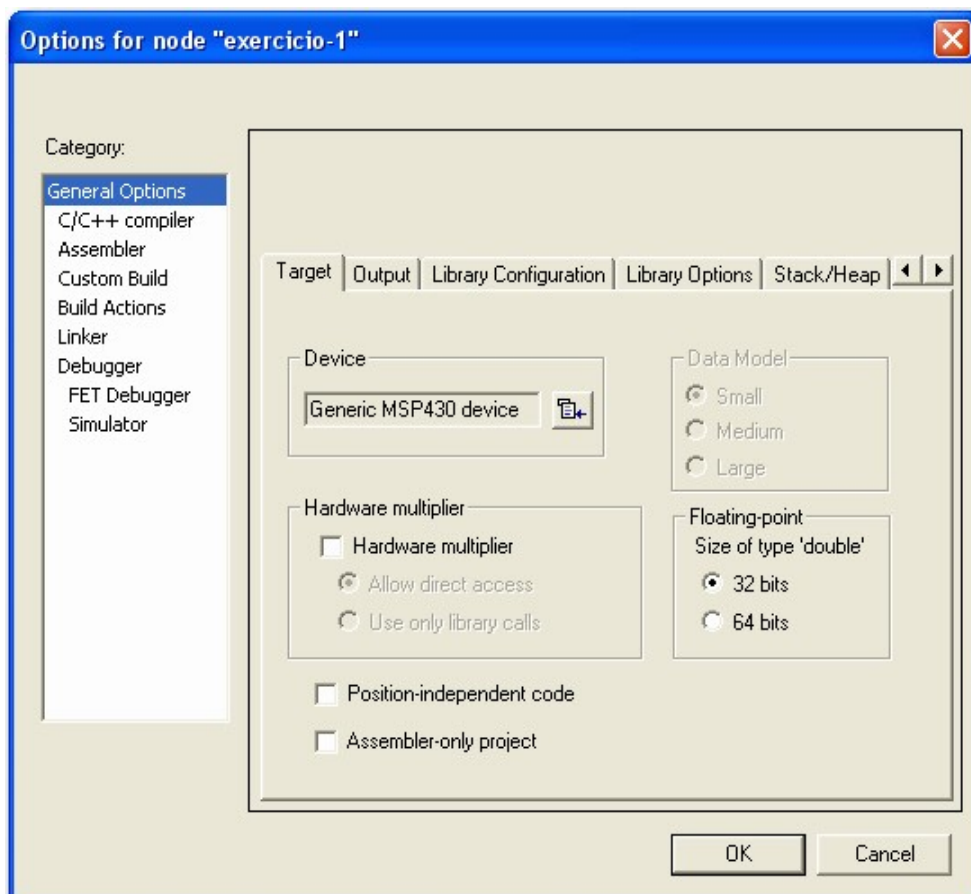
Note que o asterisco existente no projeto indica que modificações foram feitas (foi inserido uma arquivo de código) mas ele ainda não foi salvo. Ao clicar duas vezes sob o arquivo do código fonte (**E-0-0.s43**) ele se abrirá na janela ao lado, permitindo qualquer edição que o programador julgue necessário, como pode ser visto na figura a seguir.



Antes de salvar o projeto ou compilar o programa que foi escrito/editado, é interessante primeiramente configurar o projeto. Para isto, devemos clicar novamente com o botão direito sobre o ícone azul do projeto. Desta vez faremos a seleção do item **OPTIONS**, como pode ser visto na figura a seguir.

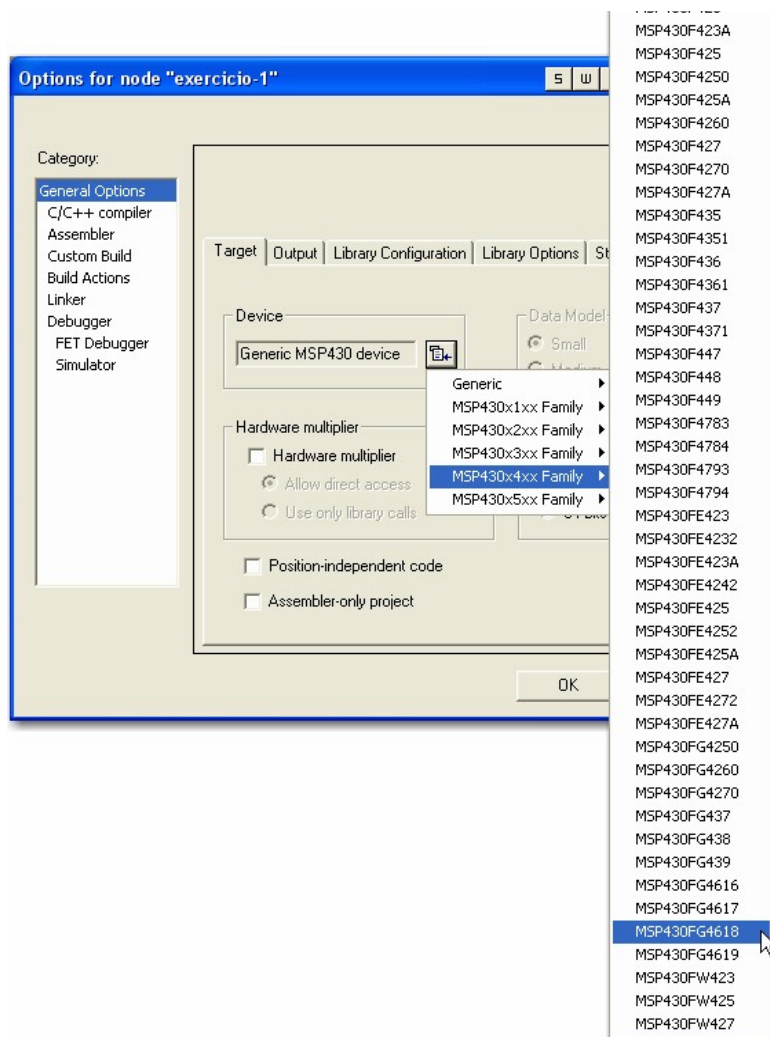


Isto fará com que a janela de configuração de projetos seja aberta, como pode ser visto na figura a seguir.

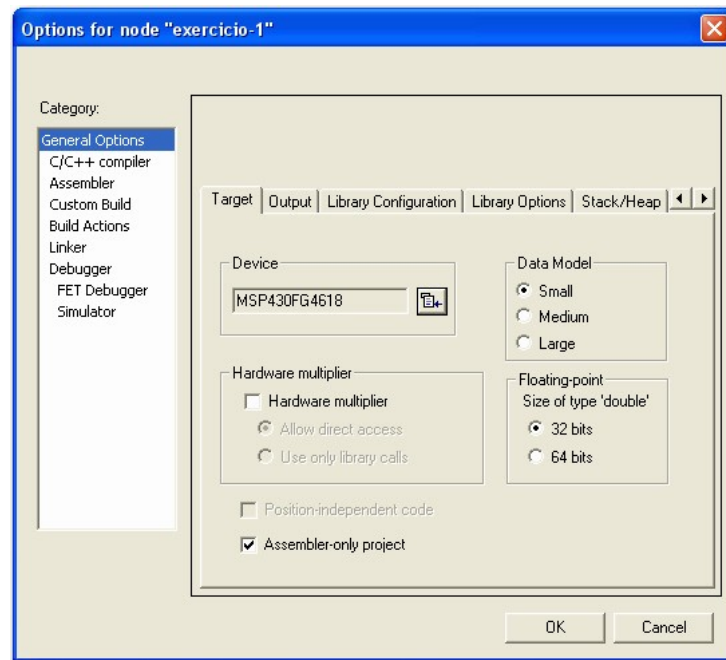


Esta janela contém uma série de configurações para o projeto. Muitas vezes o bom funcionamento de um projeto depende exclusivamente da configuração correta desta tela. Não abordaremos aqui todos os itens destas configurações, o que será feito ao longo do treinamento. Nos prenderemos, neste primeiro momento, apenas naquelas que são essenciais para compilar o primeiro projeto. Veja que esta tela tem **categorias** de configurações (**GENERAL OPTIONS**, **C/C++**, **ASSEMBLER**, etc.) e que cada uma destas categorias tem uma série de **paletas** (**TARGET**, **OUTPUT**, **LIBRARY CONFIGURATION**, etc.) correspondentes.

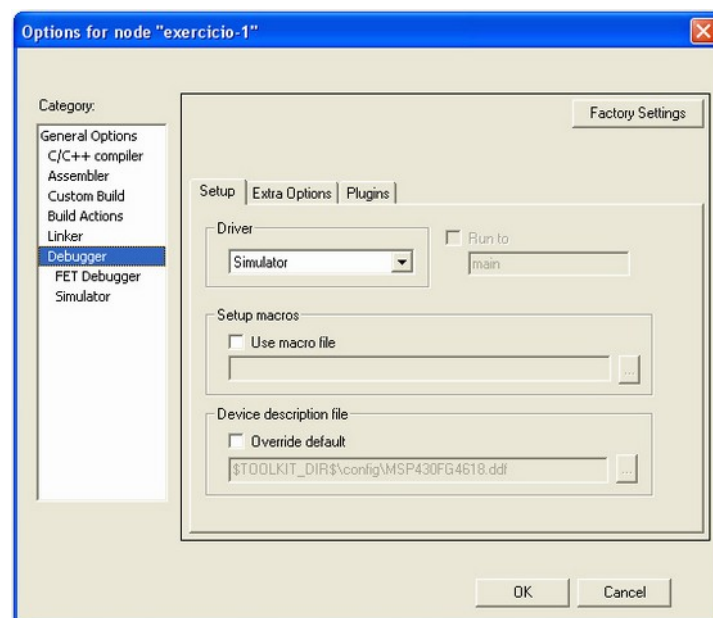
Para uma primeira configuração inicial, selecionaremos a na **paleta TARGET** na **categoria GENERAL OPTIONS**. A primeira coisa a fazer é selecionar qual é o dispositivo que será gravado. Como faremos uso da Experimenter Board neste treinamento, devemos selecionar o chip correspondente (**MSP430FG4618**) na caixa de seleção **DEVICE**, como pode ser visto na figura a seguir.



O próximo passo é clicar no **Box ASSEMBLY-ONLY PROJECT**, pois como o código está todo escrito em linguagem de máquina. Deste modo as diretivas correspondentes a linguagem C serão ignoradas, como mostra a figura a seguir.



Várias outras configurações podem ser feitas nesta **categoria (GENERAL OPTIONS)**. Mas para nosso primeiro projeto funcionar pararemos aqui e passaremos direto à **categoria DEBUGGER**. Nesta **categoria**, selecionaremos a **aba SETUP**, como pode ser visto na figura abaixo.

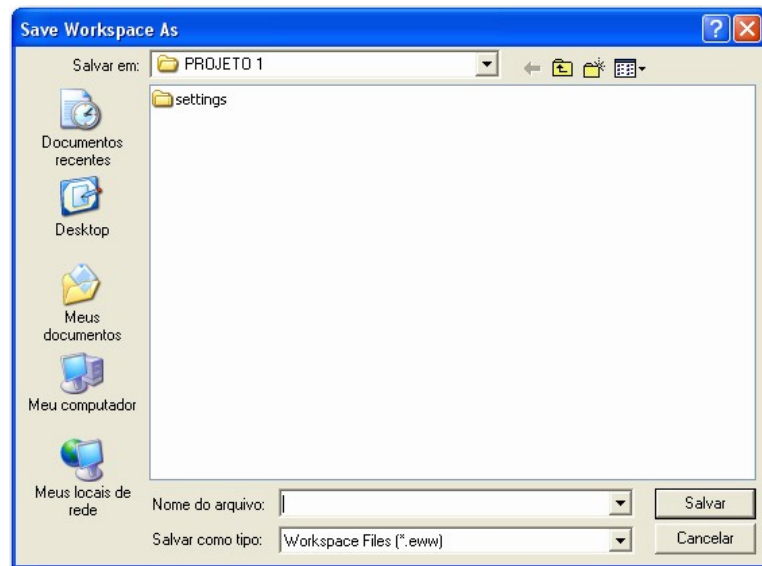


Na **aba SETUP** é necessário selecionar o driver que será utilizado para a compilação do projeto. As opções são **SIMULATOR** (o projeto não será gravado em nenhum dispositivo e será simulado apenas no computador) ou **FET DEBUGGER** (o programa será gravado no dispositivo selecionado anteriormente e executado em tempo real).

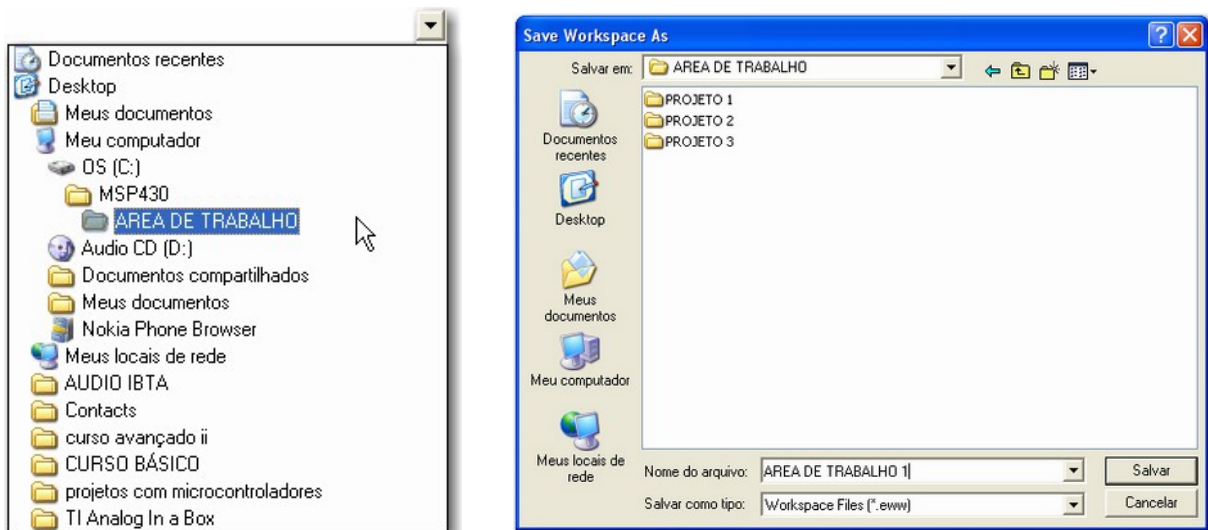
Com todas estas configurações feitas, chegou a hora de **compilar o projeto e colocá-lo para rodar**. Para isto, vá até o menu principal e selecione **PROJECT**, e em seguida **DOWNLOAD AND DEBUG (CTRL + D)**.



Note que até aqui fizemos toda a configuração de projeto, mas ainda não salvamos uma área de trabalho. Por este motivo, antes de compilar um projeto é necessário salvar uma área de trabalho. Deste modo, ao tentar compilar pela primeira vez, aparecerá a tela de salvamento de área de trabalho, como pode ser visto na figura a seguir.

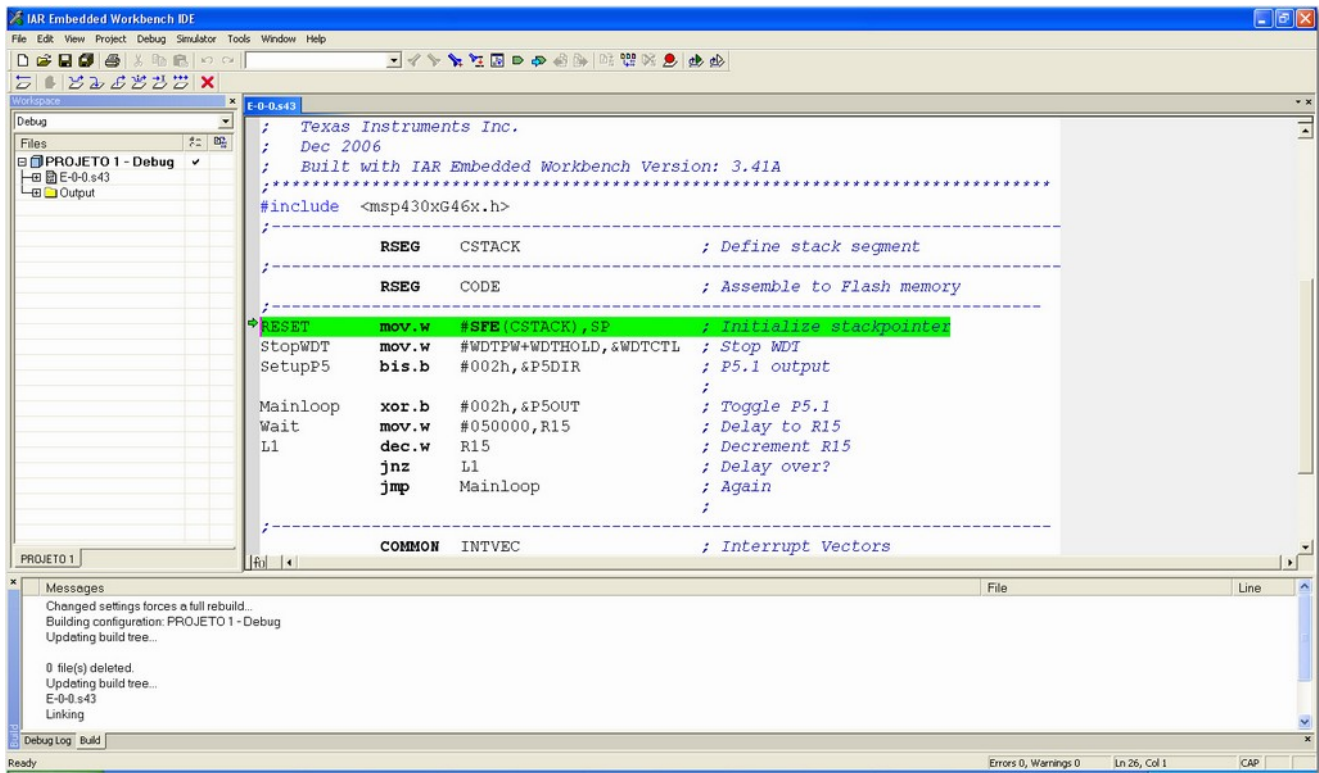


Fique atento para o local onde a área de trabalho será salva. Lembre-se da organização de pastas que criamos no começo deste processo. Uma área de trabalho não deve ser salva dentro de uma pasta de projetos, e sim dentro da pasta de área de trabalho, como você pode ver na figura abaixo. Uma área de trabalho tem a extensão **.eww**, que não precisa ser colocada, pois o software preenche automaticamente.



Se tudo ocorreu bem, e o arquivo assembly não contém erros, o projeto será compilado e aparecerá a tela de execução, como pode ser visto na próxima figura.

MSP430 – Linguagem Assembly



```

; Texas Instruments Inc.
; Dec 2006
; Built with IAR Embedded Workbench Version: 3.41A
;*****
#include <msp430xG46x.h>
;-----
RSEG CSTACK ; Define stack segment
;-----
RSEG CODE ; Assemble to Flash memory
;-----
RESET mov.w #SFE(CSTACK),SP ; Initialize stackpointer
StopWDT mov.w #WDTPW+WDTHOLD,&WDCTL ; Stop WDT
SetupP5 bis.b #002h,&P5DIR ; P5.1 output
;
Mainloop xor.b #002h,&P5OUT ; Toggle P5.1
Wait mov.w #050000,R15 ; Delay to R15
L1 dec.w R15 ; Decrement R15
jnz L1 ; Delay over?
jmp Mainloop ; Again
;
;-----
COMMON INTVEC ; Interrupt Vectors

```

Messages

- Changed settings forces a full rebuild...
- Building configuration: PROJETO1 - Debug
- Updating build tree...
- 0 file(s) deleted.
- Updating build tree...
- E-0-0.s43
- Linking

Ready Errors 0, Warnings 0 Ln 26, Col 1 CAP

Como Estruturar um programa em Assembly para MSP430?

- Must do** 
- Initialize Stack
 - Configure Watchdog
 - Initialize RESET Vector

```

*****
;   MSP430xG46x Demo - Software Toggle P5.1
;
;   Description: Toggle P5.1 by xor'ing P5.1 inside of a software loop.
;   ACLK = 32.768kHz, MCLK = SMCLK = default DCO
;
;           MSP430xG461x
;           -----
;           /|\
;           | |
;           --|RST
;           |
;           |           P5.1|-->LED
;
;   K. Quiring/K. Venkat
;   Texas Instruments Inc.
;   Dec 2006
;   Built with IAR Embedded Workbench Version: 3.41A
*****
#include <msp430xG46x.h>
;-----
;           RSEG    CSTACK
;                                     ; Define stack segment
;-----
;           RSEG    CODE
;                                     ; Assemble to Flash memory
;-----
RESET      mov.w    #SFE(CSTACK),SP      ; Initialize stackpointer
StopWDT    mov.w    #WDTPW+WDTHOLD,&WDTCTL ; Stop WDT
SetupP5    bis.b    #002h,&P5DIR         ; P5.1 output
;
Mainloop   xor.b    #002h,&P5OUT         ; Toggle P5.1
Wait       mov.w    #050000,R15         ; Delay to R15
L1         dec.w    R15                  ; Decrement R15

```

```
        jnz    L1                ; Delay over?  
        jmp    Mainloop          ; Again  
        ;  
-----  
COMMON INTVEC                    ; Interrupt Vectors  
-----  
ORG     RESET_VECTOR            ; MSP430 RESET Vector  
DW      RESET                    ;  
END                                           ; obrigatório
```

Exercício 1: Botões e LEDs

Identifique no diagrama elétrico da Experimenter Board, mostrado no **item 7**, as seguintes conexões, envolvendo pinos do MSP430FG4618 e hardwares externos:

- a) Botão S1 → pino P1.0;
- b) Botão S2 → pino P1.1;
- c) LED1 → pino P2.1;
- d) LED2 → pino P2.2;
- e) LED4 → pino P5.1.

Com estas informações faça modificações no programa exemplo, mostrado no **item 9**, de modo que aconteçam as operações solicitadas nos itens a seguir.

Exercício 1a: 1 botão e 1 Led

Ao **pressionar** o botão **S1** deve **acender** o **LED1**. Se o botão **não estiver pressionado**, o **LED1** deve se manter **apagado**.

Exercício 1b: 2 botões e 2 Leds

Ao **pressionar** o botão **S1** deve **apagar** o **LED1**. Se o botão **S1 não estiver pressionado**, o **LED1** deve se manter **aceso**. Ao mesmo tempo se o botão **S2** for **pressionado**, o **LED2** deve se **apagar**. Se o botão **S2 não estiver pressionado**, o **LED2** deve se manter **aceso**.

Exercício 1c: 1 botão, 1 Led e temporização simples

Ao **pressionar** o botão **S2** deve **acender** o **LED1**, que deve permanecer aceso por alguns milissegundos (tempo suficiente para perceber a retenção da informação). Se o botão **não estiver pressionado**, o **LED1** deve se manter **apagado**.

Exercício 1d: 2 botões, 2 Leds e temporização simples

Ao **pressionar** o botão **S2** deve **apagar** o **LED1**, o que deve acontecer por alguns milissegundos (tempo suficiente para perceber a retenção da informação). Se o botão **não estiver pressionado**, o **LED1** deve se manter **aceso**. Ao mesmo tempo, se o botão **S1** estiver **pressionado** o **LED2** deve ficar **apagado**, o que também deve acontecer por alguns milissegundos (tempo suficiente para perceber a retenção da informação). Se o botão **não estiver pressionado**, o **LED2** deve se manter **aceso**.

Configuração básica dos I/Os

As seguintes configurações devem ser observadas para fazer os exercícios propostos:

REGISTRADOR DE ENTRADA (Input Register PxIN)

Cada bit colocado em um pino do microcontrolador tem seu valor refletido neste registrador. Isto ocorre quando a porta está configurada para entrada, sendo válida as seguintes informações:

Bit = 0: A entrada está em nível lógico baixo (0 V).

Bit = 1: A entrada está em nível lógico alto (+Vcc).

REGISTRADOR DE SAÍDA (Output Registers PxOUT)

Cada bit escrito neste registrador refletirá em um valor de tensão no pino de saída. Isto

ocorre quando a porta está configurada para saída, sendo válida as seguintes informações:.

Bit = 0: A saída será levada a nível lógico baixo (**0 V**).

Bit = 1: A saída será levada a nível lógico alto (**+Vcc**).

REGISTRADOR DE DIREÇÃO (Direction Registers PxDIR)

Este registrador indicará se um pino de I/O será utilizado como entrada ou saída, de acordo com a seguinte configuração:

Bit = 0: A porta será configurada como **entrada**.

Bit = 1: A porta será configurada como **saída**.

REGISTRADORES DE PULL UP / PULL DOWN (Pullup/Pulldown Resistor Enable Registers PxREN (MSP430x47x only))

Apenas nos dispositivos das famílias MSP430x47x todas as portas tem resistores programáveis de pull-up/down, cuja habilitação ou não é feita do seguinte modo:

Bit = 0: Os resistores estarão **desabilitados**.

Bit = 1: Os resistores estarão **habilitados**.

REGISTRADORES DE SELEÇÃO DE FUNÇÃO (Function Select Registers PxSEL)

Como os pinos de um microcontrolador pode ter múltiplas funções, hora funcionando como terminal de I/O, hora como interface de algum periférico, é necessário ajustar qual função será realizada a cada momento. Isto é feito de acordo com os seguintes ajustes:

Bit = 0: O terminal funcionará como um **I/O**.

Bit = 1: O terminal funcionará como um **periférico**.

Nota: as interrupções por P1 e P2 são desabilitadas quando PxSEL = 1

REGISTRADOR DE SINALIZAÇÃO DE INTERRUPÇÃO (Flag Register PxIFG)

Sinalizam que uma interrupção aconteceu. A mudança de nível zero para nível um acontece por hardware. O contrário (de 1 → 0) deve ser executada por software:

Bit = 0: **não há** interrupção pendente neste pino (**0 V**).

Bit = 1: **há** uma interrupção pendente neste pino (**+Vcc**).

REGISTRADOR DE SELEÇÃO DE BORDA DE INTERRUPÇÃO (Edge Select PxIES)

Este registrador fará a seleção entre as bordas de transição no sinal dos pinos que acionarão a interrupção:

Bit = 0: PxIFG → 1 quando houver uma transição de **zero para um** no pino (**0 V** → **+Vcc**).

Bit = 1: PxIFG → 1 quando houver uma transição de **um para zero** no pino (**+Vcc** → **0 V**).

REGISTRADOR HABILITAÇÃO DE INTERRUPÇÃO (Interrupt Enable Register PxIE)

Este registrador fará a habilitação de interrupção para aquele pino:

Bit = 0: interrupções estão **desabilitadas** neste pino (**0 V**).

Bit = 1: interrupções estão **habilitadas** neste pino (**+Vcc**).

Alocação de pinos nas portas de I/O

```
/*
 * STANDARD BITS FOR DIGITAL I/O
 */

#define BIT0          (0x0001); → PINO 0 DA PORTA
#define BIT1          (0x0002); → PINO 1 DA PORTA
#define BIT2          (0x0004); → PINO 2 DA PORTA
#define BIT3          (0x0008); → PINO 3 DA PORTA
#define BIT4          (0x0010); → PINO 4 DA PORTA
#define BIT5          (0x0020); → PINO 5 DA PORTA
#define BIT6          (0x0040); → PINO 6 DA PORTA
#define BIT7          (0x0080); → PINO 7 DA PORTA
```

Exemplos de configuração de portas de I/O

Configurando um botão:

```
SetupP2    bis.b    #002h,&P2DIR          ; P2.1 output - LED

SetupP1    bis.b    #001h,&P1IE          ; P1.0 Interrupt enabled
           bis.b    #001h,&P1IES        ; P1.0 hi/low edge
           bic.b    #001h,&P1IFG        ; P1.0 IFG Cleared
```

Configurando dois botões:

```
SetupP2    bis.b    #006h,&P2DIR          ; P2.1 & P2.2 output - LED

SetupP1    bis.b    #003h,&P1IE          ; P1.0 & P1.1 Interrupt enabled
           bis.b    #003h,&P1IES        ; P1.0 & P1.1 hi/low edge
           bic.b    #003h,&P1IFG        ; P1.0 & P1.1 IFG Cleared
```

Exercício 2: Economizando energia para piscar o LED

Ao resolver os exercícios propostos anteriormente, no mesmo estilo em que foi apresentado no **item 9**, não fizemos uso da principal característica do MSP430, que é o **baixo consumo de energia**. Escreva o programa mostrado abaixo e o carregue em seu kit. Veja o seu funcionamento.

```

;*****
; MSP430xG46x Demo - Basic Timer, Toggle P5.1 Inside ISR, 32kHz ACLK
; Description: Toggles P5.1 by xor'ing P5.1 inside of a basic timer ISR.
; ACLK provides the basic timer clock source. LED toggles every 125ms.
; ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = default DCO = 32 x ACLK = 1048576Hz
; /** An external watch crystal between XIN & XOUT is required for ACLK */
;
;           MSP430xG461x
;
;           -----
;           /|\|           XIN|-
;           | |           | 32kHz
;           --|RST       XOUT|-
;           |           |
;           |           P5.1|-->LED
; K. Venkat
; Texas Instruments Inc.
; Dec 2006
; Built with IAR Embedded Workbench Version: 3.41A
;*****
#include <msp430xG46x.h>
;-----
;           RSEG    CSTACK                               ; Define stack segment
;-----
;           RSEG    CODE                                 ; Assemble to Flash memory
;-----
RESET      mov.w    #SFE(CSTACK),SP                     ; Initialize stackpointer
StopWDT    mov.w    #WDTPW+WDTHOLD,&WDTCTL              ; Stop WDT
SetupFLL   bis.b    #XCAP14PF,&FLL_CTL0                 ; Configure load caps
           bis.b    #BIT1,&P5DIR                       ; Set P5.1 as Output
SetupBT    mov.b    #BTDIV+BT_fCLK2_DIV16, & BTCTL    ; ACLK/(256*16)
           bis.b    #BTIE,&IE2                         ; Enable BT interrupt
Mainloop   bis.w    #LPM3+GIE,SR                       ; Enter LPM3, enable interrupts
           nop                                          ; Required for Debugger

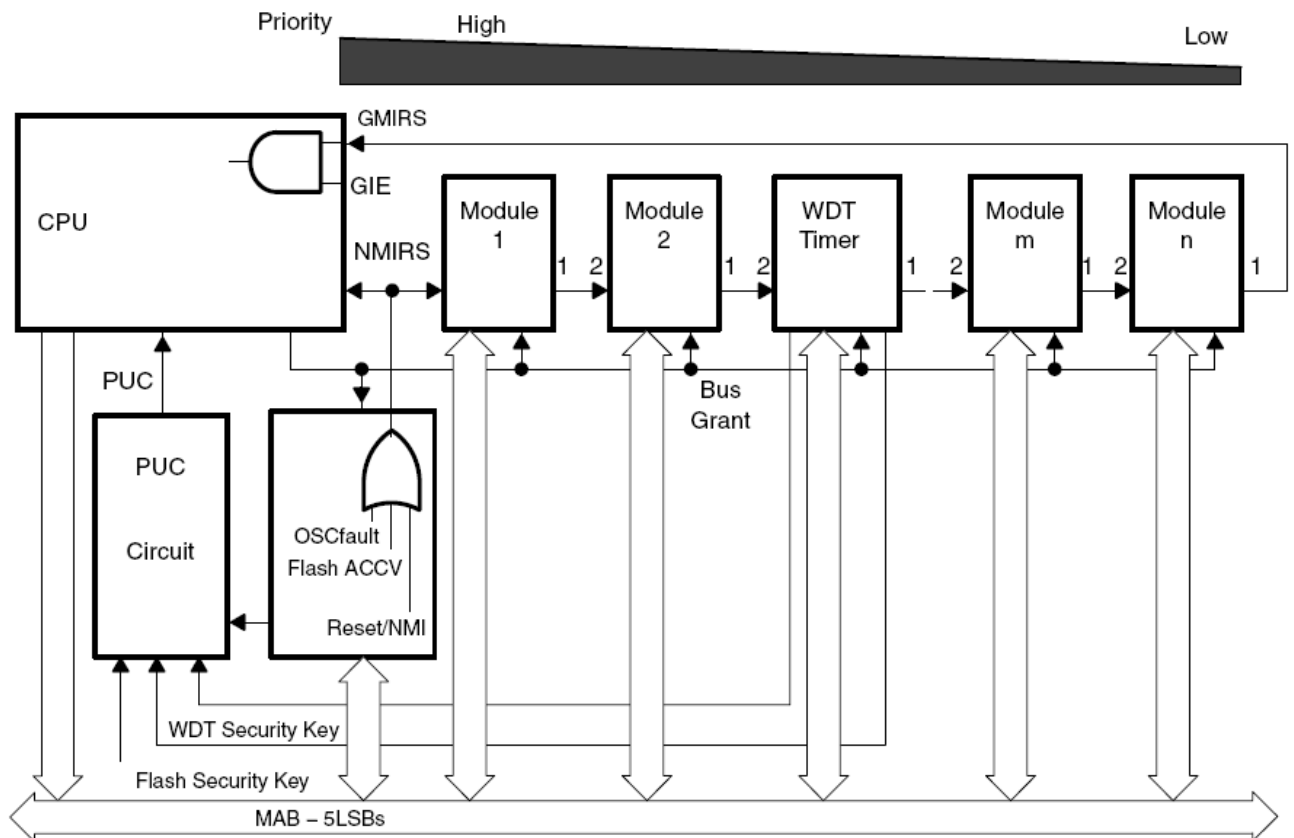
```

```
-----  
Basic_Timer_ISR ;// Basic Timer Interrupt Service Routine  
-----  
    xor.b  #BIT1,&P5OUT ; Toggle P5.1 (LED)  
    reti  
-----  
    COMMON INTVEC ; Interrupt Vectors  
-----  
    ORG    RESET_VECTOR ; MSP430 RESET Vector  
    DW    RESET ;  
    ORG    BASICTIMER_VECTOR ; MSP430 Basic Timer Interrupt Vector  
    DW    Basic_Timer_ISR  
    END
```

Para entender as vantagens conseguidas com o programa anterior, precisamos entender de dois assuntos: **interrupções** e **temporizadores** no MSP430.

Interrupções

As interrupções no MSP430 são fixas e definidas pelo arranjo modular mostrado na figura abaixo.



Quanto mais próximo for um módulo da CPU/NMIRS, maior a sua prioridade em interromper o sistema. Esta prioridade define quem será tratado primeiramente caso ocorram duas interrupções simultaneamente.

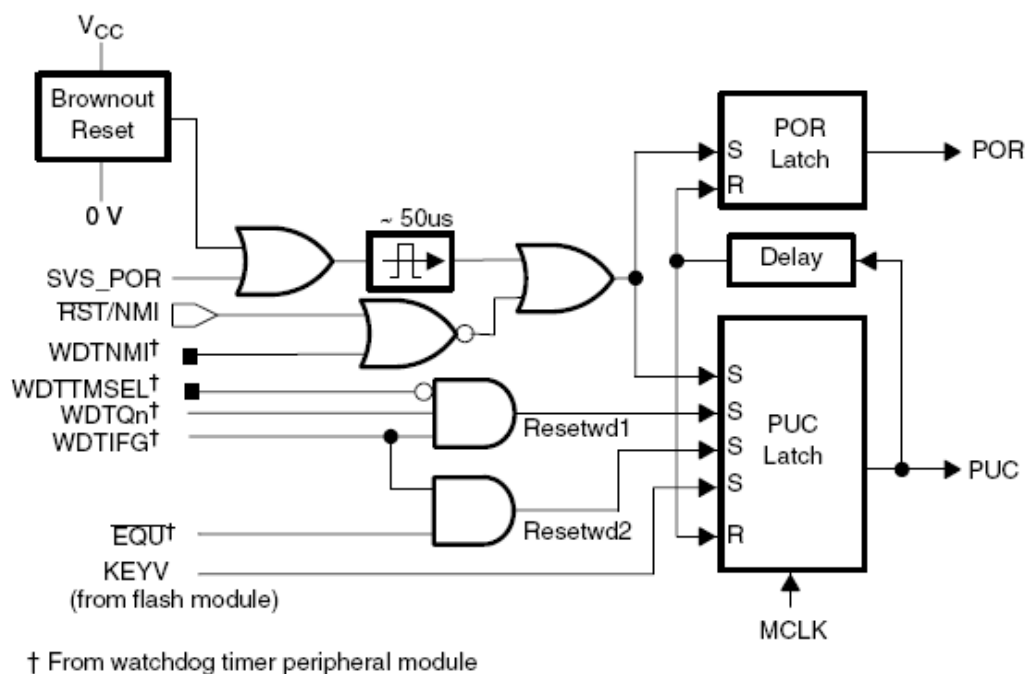
Para o MSP430 são considerados três tipos de interrupção:

- Reset do sistema;
- NMI: interrupções não mascaráveis;
- Interrupções mascaráveis.

E funcionam de acordo com a lógica mostrada na figura a seguir.

Reset do sistema

O circuito de reset do sistema é mostrado na figura a seguir. Ele fornece como saída duas fontes de reset: o **power-on reset (POR)** e o **power-up clear (PUC)**. Diferentes fontes de eventos e sinais podem disparar o circuito de reset do sistema.



O **POR** é um reset do dispositivo. Um **POR** só pode ser gerado caso ocorra um destes três eventos:

- O dispositivo é alimentado (**Powering up**);
- Um nível baixo (0 V) é inserido no terminal **RST/NMI**, quando este pino está configurado para o modo de reset;
- Um baixo nível de tensão é detectado pelo módulo **SVS**, quando **PORON = 1**.

Um **PUC** sempre é gerado quando um **POR** é gerado, porém um **POR** não é gerado quando acontece um **PUC**. Um **PUC** pode ser disparado pelos seguintes eventos:

- Um sinal **POR**;
- Expirou o tempo estipulado pelo **Watchdog timer** quando estava em modo **watchdog**;
- Houve uma violação de segurança do **Watchdog timer**;
- Houve uma violação de segurança no acesso a memória Flash.

O **Brownout Reset (BOR)** é um hardware capaz de detectar se existem variações na tensão de alimentação do dispositivo, resetando-o a fim de evitar problemas de funcionamento.

NMI – Interrupções não mascaráveis

Estas três interrupções não podem ser desabilitadas pelo bit de controle geral de interrupções (GIE). Elas têm bits específicos para isto (ACCVIE, NMIIE, OFIE). Quando uma destas interrupções é aceita, todas as demais tem o seu bit de habilitação resetado.

O programa dará um salto em sua execução para o endereço armazenado no vetor de interrupção não mascarável (**0FFFCh**). Será necessário que o software do usuário set novamente os bits destas interrupções para que elas possam ocorrer novamente.

As interrupções não mascaráveis são:

- Um borda de descida/subida no pino RST/NMI pin, quando este é configurado para o modo NMI;
- Ocorre um falha no oscilador;
- Ocorre uma violação de acesso a memória flash.

Interrupções mascaráveis

As interrupções mascaráveis são geradas por todos os periféricos que possuem capacidade de interrupção, incluindo o *watchdog timer*. Cada fonte de interrupção mascarável tem um bit que habilita ou desabilita seu funcionamento, sendo que existe ainda um bit de controle geral, chamado de **GIE** (*General Interrupt Enable*), presente no status **register** (**SR**).

O processamento de uma interrupção

Quando uma interrupção é solicitada por um periférico, é necessário que os bits de habilitação individual e geral (GIE) estejam setados. Isto fará com que o programa sofra um desvio para uma rotina de interrupção.

Entrada em uma interrupção

A latência para o tratamento de um interrupção dura 6 ciclos de máquina, começando pela aceitação de uma interrupção e terminando com a execução da primeira instrução contida na rotina de atendimento de interrupção. A execução lógica é descrita a seguir:

- Qualquer instrução que esteja sendo executada é finalizada;
- O valor do PC, que aponta para a próxima instrução que deveria ser executada, é salvo na pilha (pushed onto the stack);
- O valor do SR é salvo na pilha (pushed onto the stack);
- A interrupção que tiver a maior prioridade é selecionada, caso múltiplas interrupções ocorram simultaneamente;
- Uma requisição de interrupção reseta o flag de interrupção dos periféricos que tem fonte única

de interrupção (o conversor AD de 12 bits, por exemplo). Se o periférico tiver múltiplas flags de interrupção (portas P1 e P2, por exemplo), então o reset destas flags deverá acontecer por software.

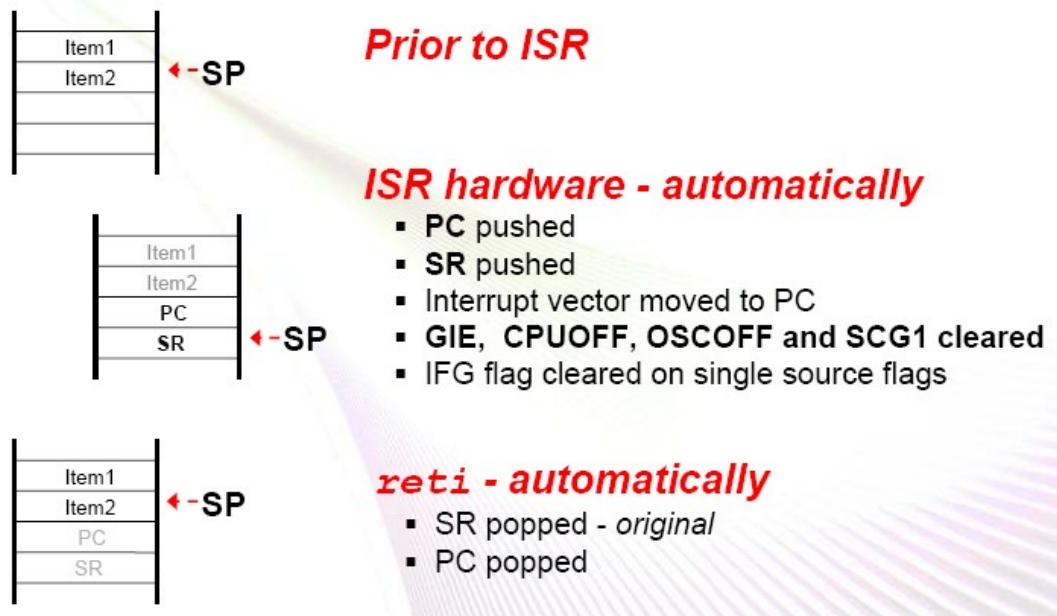
- O registrador SR tem todos os seus bits zerados, com exceção do SCG0, que não é alterado. Isto retira o microcontrolador de qualquer LPM em que ele se encontra. Como o GIE também é zerado, as próximas interrupções ficam desabilitadas.
- O conteúdo do vetor de interrupção correspondente a quem fez a solicitação é carregado no PC, que passará a executar o programa a partir do que houver escrito ali.

Saída de uma interrupção

Toda interrupção é terminada quando uma instrução **RETI** (return from an interrupt service routine) é encontrada. Este retorno gasta 5 ciclos de máquina para executar as seguintes ações:

- 1) O registrador SR é restaurado da pilha (pops from the stack) com todos os valores ajustados previamente (GIE, CPUOFF, etc.);
- 2) O valor do PC é restaurado da pilha (pops from the stack) e inicia a execução no ponto exato em que parou ao ser interrompido.

Interrupt Processing



O **aninhamento** de interrupções é habilitado quando, durante o tratamento de uma interrupção, o usuário seta o bit GIE. Neste caso, havendo uma nova interrupção os procedimentos mostrados ocorrerão novamente, seguindo a ordem de prioridades de atendimento de interrupções.

Os vetores de interrupção

Os vetores de interrupção tem um endereçamento fixo e conhecido em todos os dispositivos das famílias MSP430, indo de **0FFFFh até 0FFC0h**. Nestes dados, que tem 16 bits, são salvos os endereços onde estarão os programas que farão o atendimento as rotinas de interrupção. Dependendo do número de periféricos, os dispositivos MSP430 podem ter mais ou menos vetores de interrupção, por isto é sempre necessária uma consulta ao datasheet de cada dispositivo.

Vetores de interrupção na família 2 (MSP430F2013)

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-up External reset Watchdog Timer+ Flash key violation PC out-of-range (see Note 1)	PORIFG RSTIFG WDTIFG KEYV (see Note 2)	Reset	0FFFEh	31, highest
NMI Oscillator fault Flash memory access violation	NMIIFG OFIFG ACCVIFG (see Notes 2 and 4)	(non)-maskable, (non)-maskable, (non)-maskable	0FFFCh	30
			0FFFAh	29
			0FFF8h	28
Comparator_A+ (MSP430x20x1 only)	CAIFG (see Note 3)	maskable	0FFF6h	27
Watchdog Timer+	WDTIFG	maskable	0FFF4h	26
Timer_A2	TACCR0 CCIFG (see Note 3)	maskable	0FFF2h	25
Timer_A2	TACCR1 CCIFG. TAIFG (see Notes 2 and 3)	maskable	0FFF0h	24
			0FFEEh	23
			0FFECCh	22
ADC10 (MSP430x20x2 only)	ADC10IFG (see Note 3)	maskable		
SD16_A (MSP430x20x3 only)	SD16CCTL0 SD16OVIFG, SD16CCTL0 SD16IFG (see Notes 2 and 3)	maskable	0FFEAh	21
USI (MSP430x20x2, MSP430x20x3 only)	USIIFG, USISTTIFG (see Notes 2 and 3)	maskable	0FFE8h	20
I/O Port P2 (two flags)	P2IFG.6 to P2IFG.7 (see Notes 2 and 3)	maskable	0FFE6h	19
I/O Port P1 (eight flags)	P1IFG.0 to P1IFG.7 (see Notes 2 and 3)	maskable	0FFE4h	18
			0FFE2h	17
			0FFE0h	16
(see Note 5)			0FFDEh ... 0FFC0h	15 ... 0, lowest

NOTAS:

- Um reset é gerado toda vez que a CPU tenta carregar um endereço no PC que esteja fora do range de memória disponível no dispositivo.
- Fonte de múltiplos flags.
- A flag de interrupção é alocada no módulo.
- Não mascarável: o bit de interrupção individual pode ser desabilitado por um evento de interrupção, mas geralmente isto não é possível.
- Os endereços de vetores de interrupção entre 0FFDEh e 0FFC0h não são utilizados neste dispositivo, podendo ser utilizado como área de código regular, onde será armazenado o programa.

Declaração dos Vetores de interrupção no MSP430F2013

```

/*****
* Interrupt Vectors (offset from 0xFFE0)
*****/

#define PORT1_VECTOR      (2 * 2u) /* 0xFFE4 Port 1 */
#define PORT2_VECTOR      (3 * 2u) /* 0xFFE6 Port 2 */
#define USI_VECTOR        (4 * 2u) /* 0xFFE8 USI */
#define SD16_VECTOR       (5 * 2u) /* 0xFFEA Sigma Delta ADC */
#define TIMERA1_VECTOR    (8 * 2u) /* 0xFFFF0 Timer A CC1, TA */
#define TIMERA0_VECTOR    (9 * 2u) /* 0xFFFF2 Timer A CC0 */
#define WDT_VECTOR        (10 * 2u) /* 0xFFFF4 Watchdog Timer */
#define NMI_VECTOR        (14 * 2u) /* 0xFFFFC Non-maskable */
#define RESET_VECTOR      (15 * 2u) /* 0xFFFFE Reset [Highest Priority] */

```

Vetores de interrupção na família 4 (MSP430FG4618)

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-Up External Reset Watchdog Flash Memory	WDTIFG KEYV (see Note 1 and 5)	Reset	0xFFEh	31, highest
NMI Oscillator Fault Flash Memory Access Violation	NMIIFG (see Notes 1 and 3) OFIFG (see Notes 1 and 3) ACCVIFG (see Notes 1, 2, and 5)	(Non)maskable (Non)maskable (Non)maskable	0xFFCCh	30
Timer_B7	TBCCR0 CCIFG0 (see Note 2)	Maskable	0xFFACh	29
Timer_B7	TBCCR1 CCIFG1 ... TBCCR6 CCIFG6, TBIFG (see Notes 1 and 2)	Maskable	0xFFF8h	28
Comparator_A	CAIFG	Maskable	0xFF6h	27
Watchdog Timer+	WDTIFG	Maskable	0xFF4h	26
USCI_A0/USCI_B0 Receive	UCA0RXIFG, UCB0RXIFG (see Note 1)	Maskable	0xFF2h	25
USCI_A0/USCI_B0 Transmit	UCA0TXIFG, UCB0TXIFG (see Note 1)	Maskable	0xFF0h	24
ADC12	ADC12IFG (see Notes 1 and 2)	Maskable	0xFFEh	23
Timer_A3	TACCR0 CCIFG0 (see Note 2)	Maskable	0xFFCCh	22
Timer_A3	TACCR1 CCIFG1 and TACCR2 CCIFG2, TAIFG (see Notes 1 and 2)	Maskable	0xFFEAh	21
I/O Port P1 (Eight Flags)	P1IFG.0 to P1IFG.7 (see Notes 1 and 2)	Maskable	0xFFE8h	20
USART1 Receive	URXIFG1	Maskable	0xFFE6h	19
USART1 Transmit	UTXIFG1	Maskable	0xFFE4h	18
I/O Port P2 (Eight Flags)	P2IFG.0 to P2IFG.7 (see Notes 1 and 2)	Maskable	0xFFE2h	17
Basic Timer1/RTC	BTIFG	Maskable	0xFFE0h	16
DMA	DMA0IFG, DMA1IFG, DMA2IFG (see Notes 1 and 2)	Maskable	0xFFDEh	15
DAC12	DAC12.0IFG, DAC12.1IFG (see Notes 1 and 2)	Maskable	0xFFDCh	14
Reserved	Reserved (see Note 4)		0xFFDAh	13
		
			0xFFC0h	0, lowest

NOTAS:

1. Fonte de múltiplos flags.
2. A flag de interrupção é alocada no módulo.
3. Um reset é gerado toda vez que a CPU tenta carregar um endereço no PC que esteja fora do range de memória disponível no dispositivo. (Não mascarável: o bit de interrupção individual pode ser desabilitado por um evento de interrupção, mas geralmente isto não é possível).
4. Os endereços de vetores de interrupção entre 0FFDAh e 0FFC0h não são utilizados neste dispositivo, podendo ser utilizado como área de código regular, onde será armazenado o programa.
5. Violação na chave de acesso (KEYV e ACCVIFG), somente são aplicáveis aos dispositivos do tipo F.

Declaração dos Vetores de interrupção no MSP430FG4618

```

/*****
* Interrupt Vectors (offset from 0xFFC0)
*****/
#define DAC12_VECTOR      (14 * 2u) /* 0xFFDC DAC 12 */
#define DMA_VECTOR        (15 * 2u) /* 0xFFDE DMA */
#define BASICTIMER_VECTOR (16 * 2u) /* 0xFFE0 Basic Timer / RTC */
#define PORT2_VECTOR      (17 * 2u) /* 0xFFE2 Port 2 */
#define USART1TX_VECTOR   (18 * 2u) /* 0xFFE4 USART 1 Transmit */
#define USART1RX_VECTOR   (19 * 2u) /* 0xFFE6 USART 1 Receive */
#define PORT1_VECTOR      (20 * 2u) /* 0xFFE8 Port 1 */
#define TIMERA1_VECTOR     (21 * 2u) /* 0xFFEA Timer A CC1-2, TA */
#define TIMERA0_VECTOR     (22 * 2u) /* 0xFFEC Timer A CC0 */
#define ADC12_VECTOR       (23 * 2u) /* 0xFFEE ADC */
#define USCIAB0TX_VECTOR   (24 * 2u) /* 0xFFF0 USCI A0/B0 Transmit */
#define USCIAB0RX_VECTOR   (25 * 2u) /* 0xFFF2 USCI A0/B0 Receive */
#define WDT_VECTOR         (26 * 2u) /* 0xFFF4 Watchdog Timer */
#define COMPARATORA_VECTOR (27 * 2u) /* 0xFFF6 Comparator A */
#define TIMERB1_VECTOR     (28 * 2u) /* 0xFFF8 Timer B CC1-2, TB */
#define TIMERB0_VECTOR     (29 * 2u) /* 0xFFFA Timer B CC0 */
#define NMI_VECTOR         (30 * 2u) /* 0xFFFC Non-maskable */
#define RESET_VECTOR       (31 * 2u) /* 0xFFFE Reset [Highest Priority] */

```

BASIC TIMER 1

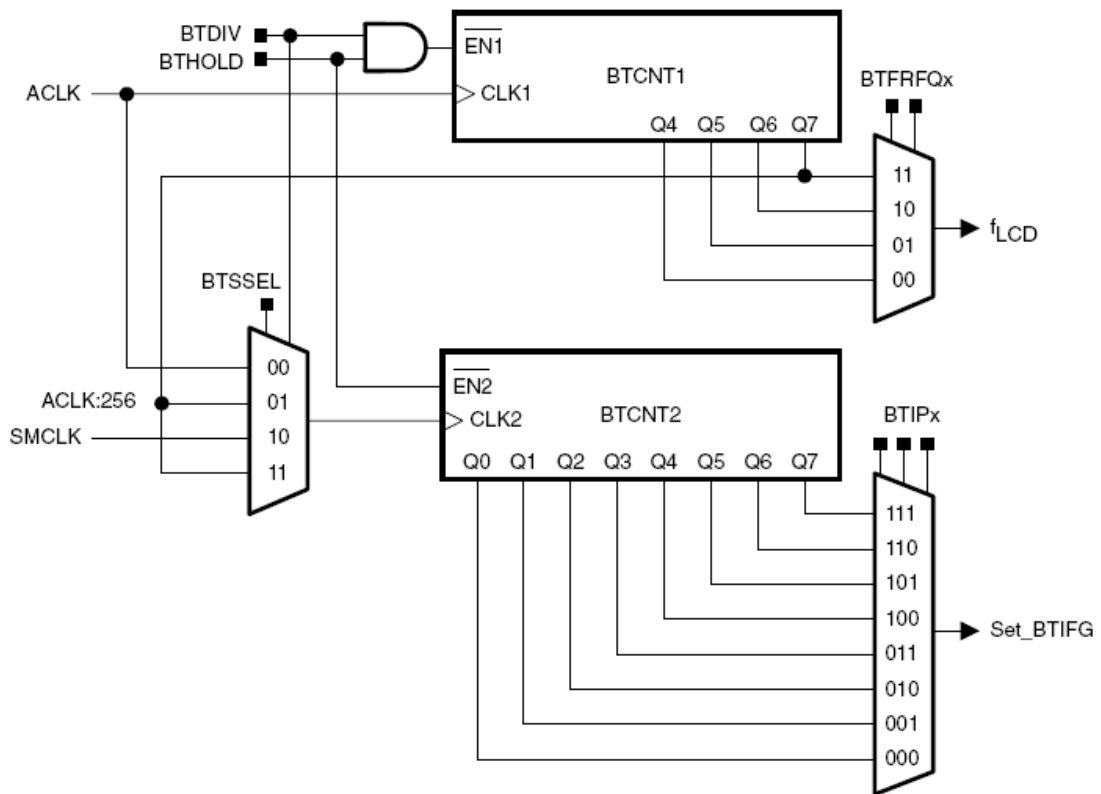
O Basic Timer 1 é dedicado a fornecer base de tempo para LCD e intervalos de tempo de baixa frequência. Ele tem dois temporizadores independentes de 8 Bits, que podem operar de forma isolada ou em cascata, compondo um temporizador de 16 bits. Alguns dos usos típicos atribuídos ao Basic Timer 1 são:

- RTC – Real time clock
- Incremento de tempo via software

Suas principais características são:

- Poder seleccionar qual será a fonte de clock a incrementar este temporizador;
- Dois registradores independentes de 8 bits, que podem ser conectados em cascata;
- Capacidade de gerar interrupção ao estourar uma contagem;
- Geração de sinais de controle para LCD

O diagrama em blocos deste temporizador é mostrado na figura na página a seguir.



Os registradores que gerenciam o funcionamento do Basic Timer 1 são mostrados a seguir.

Table 13–1. Basic Timer1 Registers

Register	Short Form	Register Type	Address	Initial State
Basic Timer1 Control	BTCTL	Read/write	040h	Unchanged
Basic Timer1 Counter 1	BTCNT1	Read/write	046h	Unchanged
Basic Timer1 Counter 2	BTCNT2	Read/write	047h	Unchanged
SFR interrupt enable register 2	IE2	Read/write	001h	Reset with PUC
SFR interrupt flag register 2	IFG2	Read/write	003h	Reset with PUC

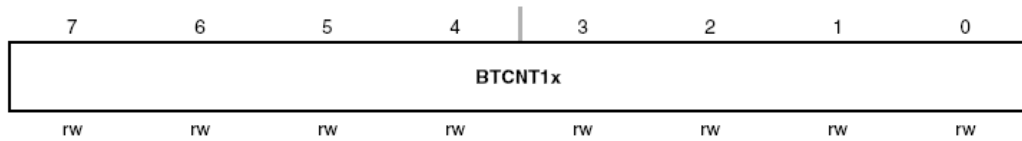
Note: The Basic Timer1 registers should be configured at power-up. There is no initial state for BTCTL, BTCNT1, or BTCNT2.

BTCTL, Basic Timer1 Control Register

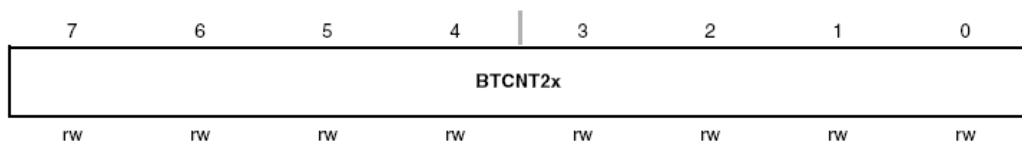
BTSSSEL	Bit 7	BTCNT2 clock select. This bit, together with the BTDIV bit, selects the clock source for BTCNT2. See the description for BTDIV.
BTHOLD	Bit 6	Basic Timer1 Hold. 0 BTCNT1 and BTCNT2 are operational 1 BTCNT1 is held if BTDIV=1 BTCNT2 is held
BTDIV	Bit 5	Basic Timer1 clock divide. This bit together with the BTSSSEL bit, selects the clock source for BTCNT2.

BTSSSEL	BTDIV	BTCNT2 Clock Source
0	0	ACLK
0	1	ACLK/256
1	0	SMCLK
1	1	ACLK/256

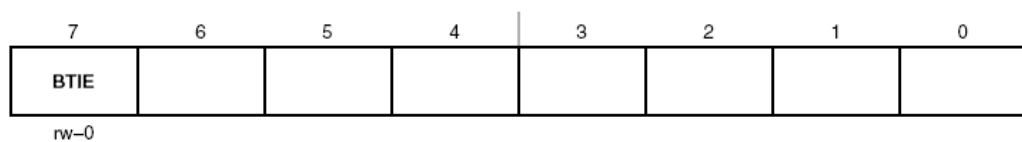
BTFRFQx	Bits 4–3	f_{LCD} frequency. These bits control the LCD update frequency. 00 $f_{ACLK}/32$ 01 $f_{ACLK}/64$ 10 $f_{ACLK}/128$ 11 $f_{ACLK}/256$
BTIPx	Bits 2–0	Basic Timer1 Interrupt Interval. 000 $f_{CLK2}/2$ 001 $f_{CLK2}/4$ 010 $f_{CLK2}/8$ 011 $f_{CLK2}/16$ 100 $f_{CLK2}/32$ 101 $f_{CLK2}/64$ 110 $f_{CLK2}/128$ 111 $f_{CLK2}/256$

BTCNT1, Basic Timer1 Counter 1

BTCNT1x Bits 7-0 BTCNT1 register. The BTCNT1 register is the count of BTCNT1.

BTCNT2, Basic Timer1 Counter 2

BTCNT2x Bits 7-0 BTCNT2 register. The BTCNT2 register is the count of BTCNT2.

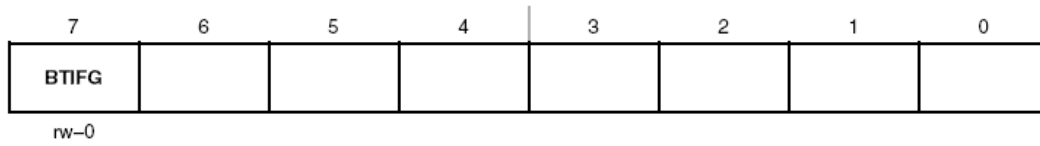
IE2, Interrupt Enable Register 2

BTIE Bit 7 Basic Timer1 interrupt enable. This bit enables the BTIFG interrupt. Because other bits in IE2 may be used for other modules, it is recommended to set or clear this bit using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions.

0 Interrupt not enabled
1 Interrupt enabled

Bits 6-1 These bits may be used by other modules. See device-specific datasheet.

IFG2, Interrupt Flag Register 2



BTIFG Bit 7 Basic Timer1 interrupt flag. Because other bits in IFG2 may be used for other modules, it is recommended to clear BTIFG automatically by servicing the interrupt, or by using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions.

0 No interrupt pending

1 Interrupt pending

Bits 6-1 These bits may be used by other modules. See device-specific datasheet.

• Exercício 3: Botões e LEDs em Power Mode

Já foi identificado no diagrama elétrico da Experimenter Board, mostrado no **item 9**, as seguintes conexões, envolvendo pinos do MSP430FG4618 e hardwares externos:

- a) Botão S1 → pino P1.0;
- b) Botão S2 → pino P1.1;
- c) LED1 → pino P2.1;
- d) LED2 → pino P2.2;
- e) LED4 → pino P5.1.

Com estas informações faça modificações na sequência de exercícios 1, **de modo que o programa fique em Low Power Mode e apenas saia deste estado para executar ações**, fazendo **economia de energia** e realizando as mesmas atividades anteriores.

Lembre-se que para obter o tempo de **500 ms** no Basic Timer você deverá observar qual o valor do cristal conectado na Experimenter Board. Veja as seguintes informações retiradas do manual da placa:

MSP430F2013 Clock Sources

The MSP430F2013 uses the **internal VLO** operating at ~12kHz for an ultra-low power standby wake up time base. The integrated DCO is internally programmable at frequencies up to 16MHz for high speed CPU and system clocking.

MSP430FG4618 Clock Sources

A standard **32.768kHz** watch crystal is populated at footprint X2 and sources source ACLK of the MSP430FG4618 for low frequency, ultra-low power standby operation and RTC functionality. The integrated FLL+ clock module provides a programmable internal high frequency clock source for the CPU and other peripherals on-chip. In addition to the FLL+, an external high frequency crystal or resonator up to 8MHz can be added via footprint X1.

Exercício 3a: 1 botão e 1 Led

Ao pressionar o botão **S1** deve acender o **LED1**. Se o botão não estiver pressionado, o **LED1** deve se manter apagado.

Exercício 3b: 2 botões e 2 Leds

Ao **pressionar** o botão **S1** deve **apagar** o **LED1**. Se o **botão S1** não estiver **pressionado**, o **LED1** deve se **manter aceso**. Ao mesmo tempo se o **botão S2** for **pressionado**, o **LED2** deve se **apagar**. Se o **botão S2** não estiver **pressionado**, o **LED2** deve se **manter aceso**.

Exercício 3c: 1 botão, 1 Led e temporização com Basic Timer 1

Ao **pressionar** o **botão S2** deve acender o **LED1**, que deve permanecer aceso por **500 ms**. Se o **botão** não estiver pressionado, o **LED1** deve se manter **apagado**.

Exercício 3d: 2 botões, 2 Leds e temporização Basic Timer 1

Ao **pressionar** o botão **S2** deve **apagar** o **LED1**, o que deve acontecer por **500 ms**. Se o botão **não estiver pressionado**, o **LED1** deve se manter **aceso**. Ao mesmo tempo, se o botão **S1** estiver **pressionado** o **LED2** deve ficar **apagado**, o que também deve acontecer por **500 ms**. Se o botão **não estiver pressionado**, o **LED2** deve se manter **aceso**

TIMER A

O *timer A* é um hardware que contém um contador de 16 bits, tendo de 3 a 5 registradores de captura/comparação. Esta estrutura suporta o funcionamento de múltiplos comparadores, captura de sinais, geração de sinais de **PWM** e geração de sinais de base de tempo, além de gerar sinais de interrupção para todos os eventos controlados de hardware. As funcionalidades deste hardware são:

1. Temporizador / contador assíncrono de 16 bits, com quatro modos de operação;
2. Fonte de **clock** selecionável e configurável;
3. De 3 a 5 registradores de captura e comparação;
4. Saídas de sinais configuráveis, com capacidade de gerar sinal de **PWM**;
5. Vetor de interrupção para rápida decodificação das múltiplas interrupções geradas pelo hardware.

O Hardware principal pode ser visto na figura da página 92.

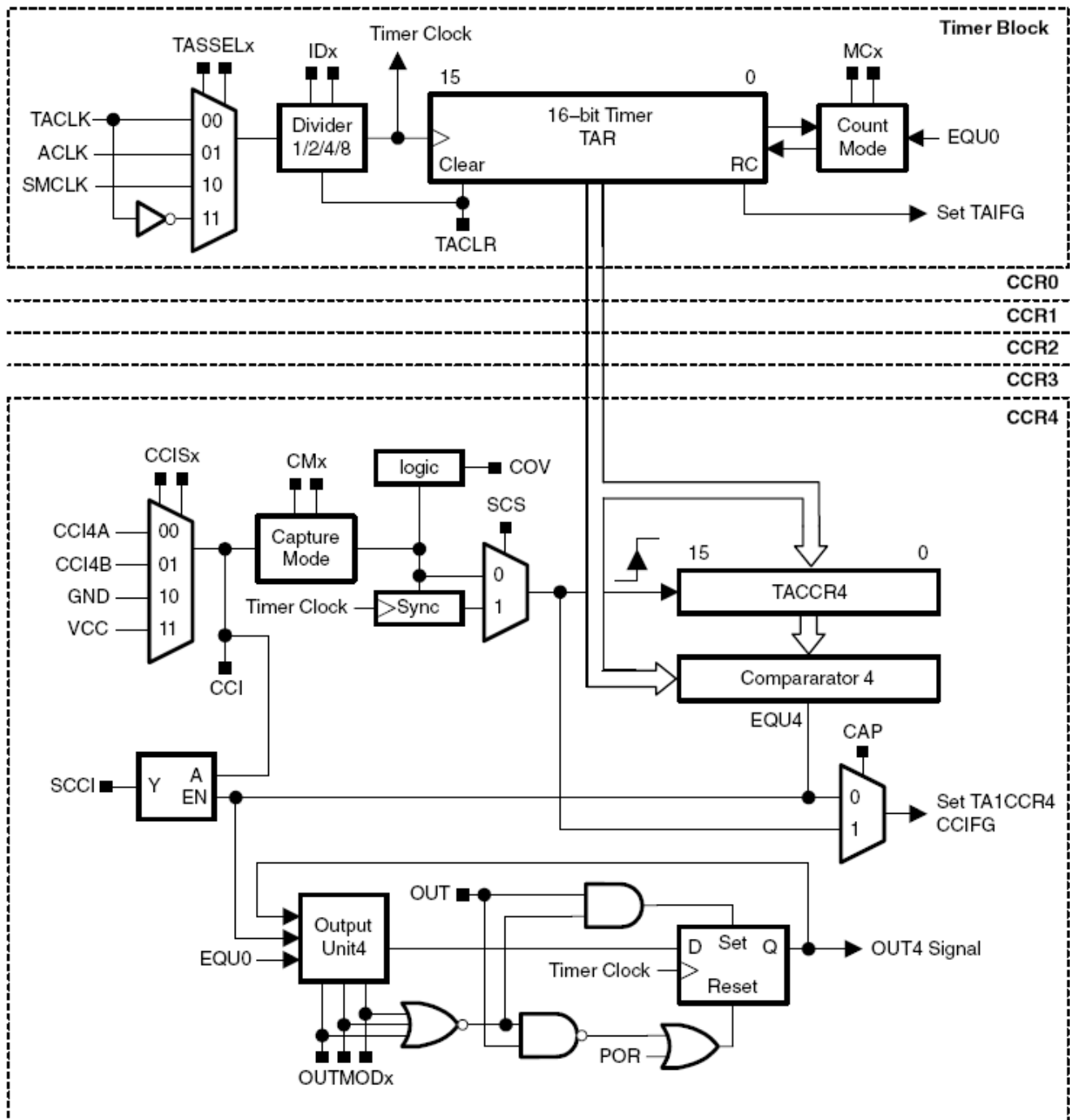
Inicializando o Timer A

O timer A pode ser inicializado ou resetado dos seguintes modos:

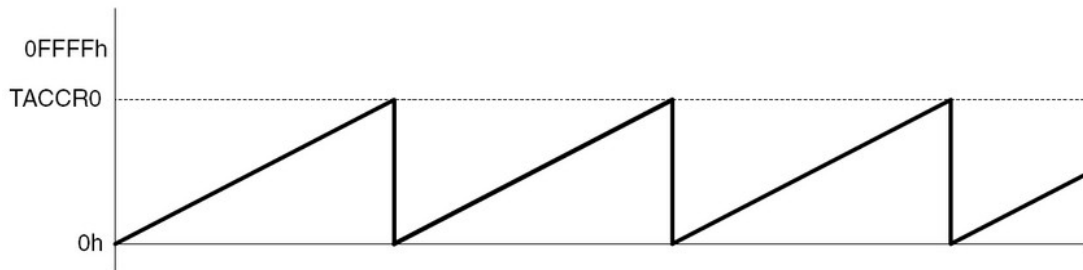
- Através do contador quando $MCx > 0$ e a fonte de clock está ativa;
- Se o timer está nos modos UP ou UP/DOWN, ele pode ser parado quando o programa escreve 0 no registrador TACCR0. Ele pode voltar a funcionar ao escrever um valor diferente de zero no mesmo registrador. Neste cenário o temporizador sempre iniciará incrementando a contagem.

O controle do Timer A

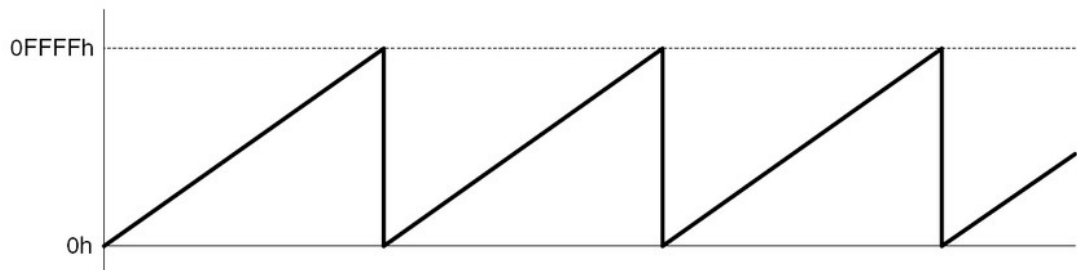
MCx	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TACCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh.
11	Up/down	The timer repeatedly counts from zero up to the value of TACCR0 and back down to zero.



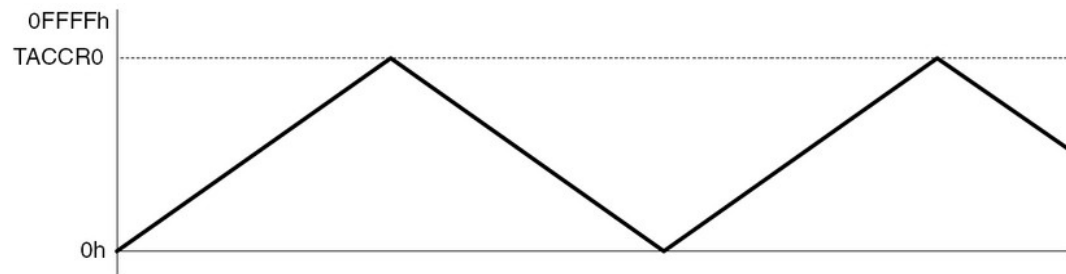
UP MODE



CONTINUOUS MODE



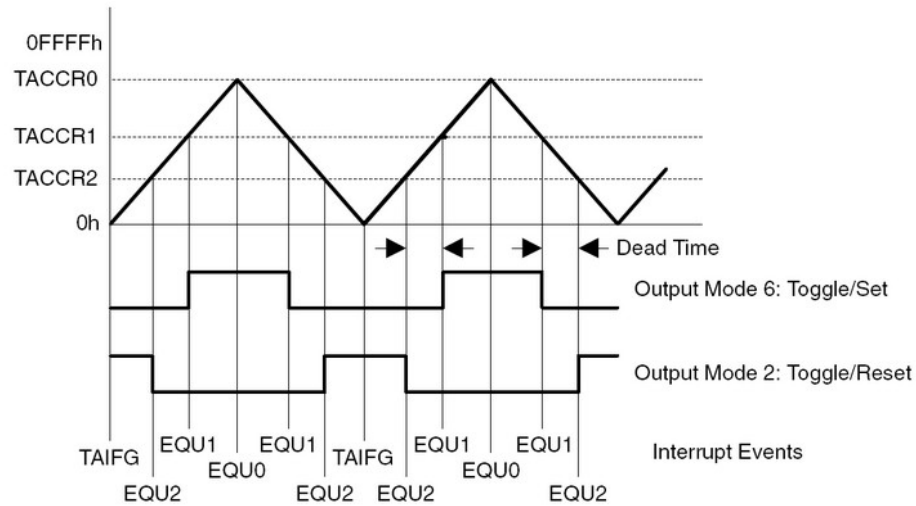
UP/DOWN MODE



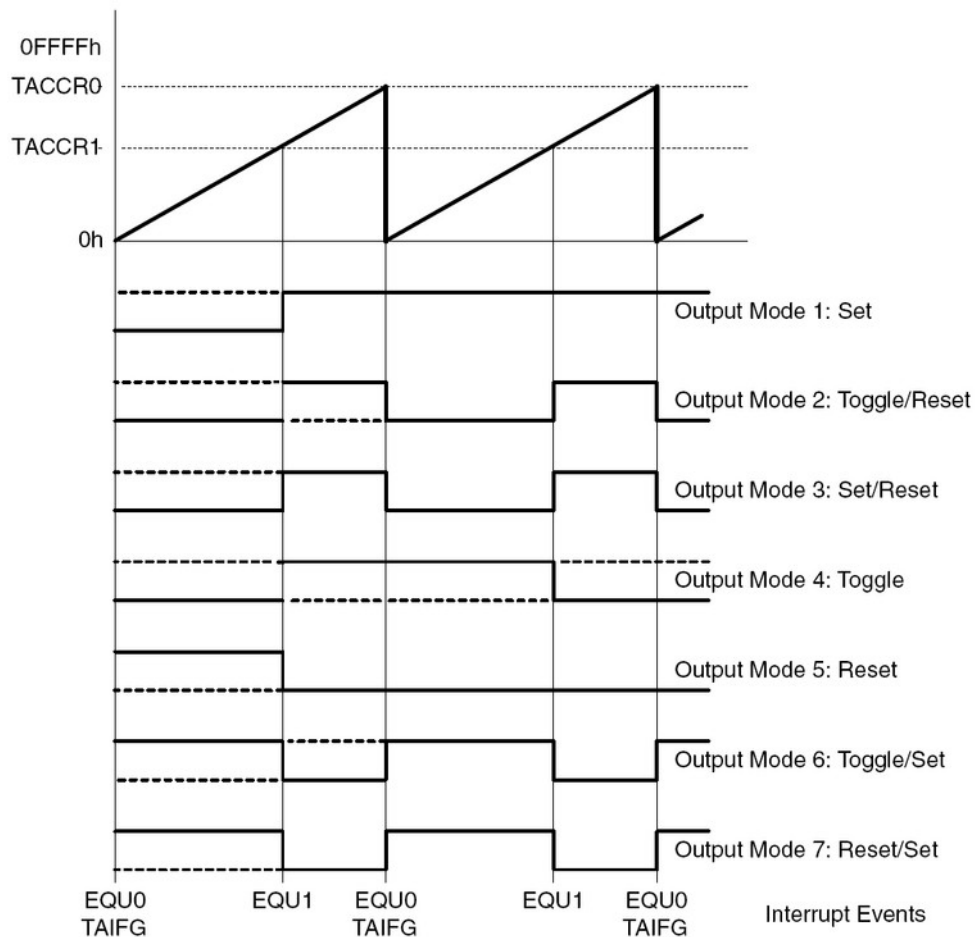
Modos de saída do Timer A

OUTMODx	Mode	Description
000	Output	The output signal OUTx is defined by the OUTx bit. The OUTx signal updates immediately when OUTx is updated.
001	Set	The output is set when the timer <i>counts</i> to the TACCRx value. It remains set until a reset of the timer, or until another output mode is selected and affects the output.
010	Toggle/Reset	The output is toggled when the timer <i>counts</i> to the TACCRx value. It is reset when the timer <i>counts</i> to the TACCR0 value.
011	Set/Reset	The output is set when the timer <i>counts</i> to the TACCRx value. It is reset when the timer <i>counts</i> to the TACCR0 value.
100	Toggle	The output is toggled when the timer <i>counts</i> to the TACCRx value. The output period is double the timer period.
101	Reset	The output is reset when the timer <i>counts</i> to the TACCRx value. It remains reset until another output mode is selected and affects the output.
110	Toggle/Set	The output is toggled when the timer <i>counts</i> to the TACCRx value. It is set when the timer <i>counts</i> to the TACCR0 value.
111	Reset/Set	The output is reset when the timer <i>counts</i> to the TACCRx value. It is set when the timer <i>counts</i> to the TACCR0 value.

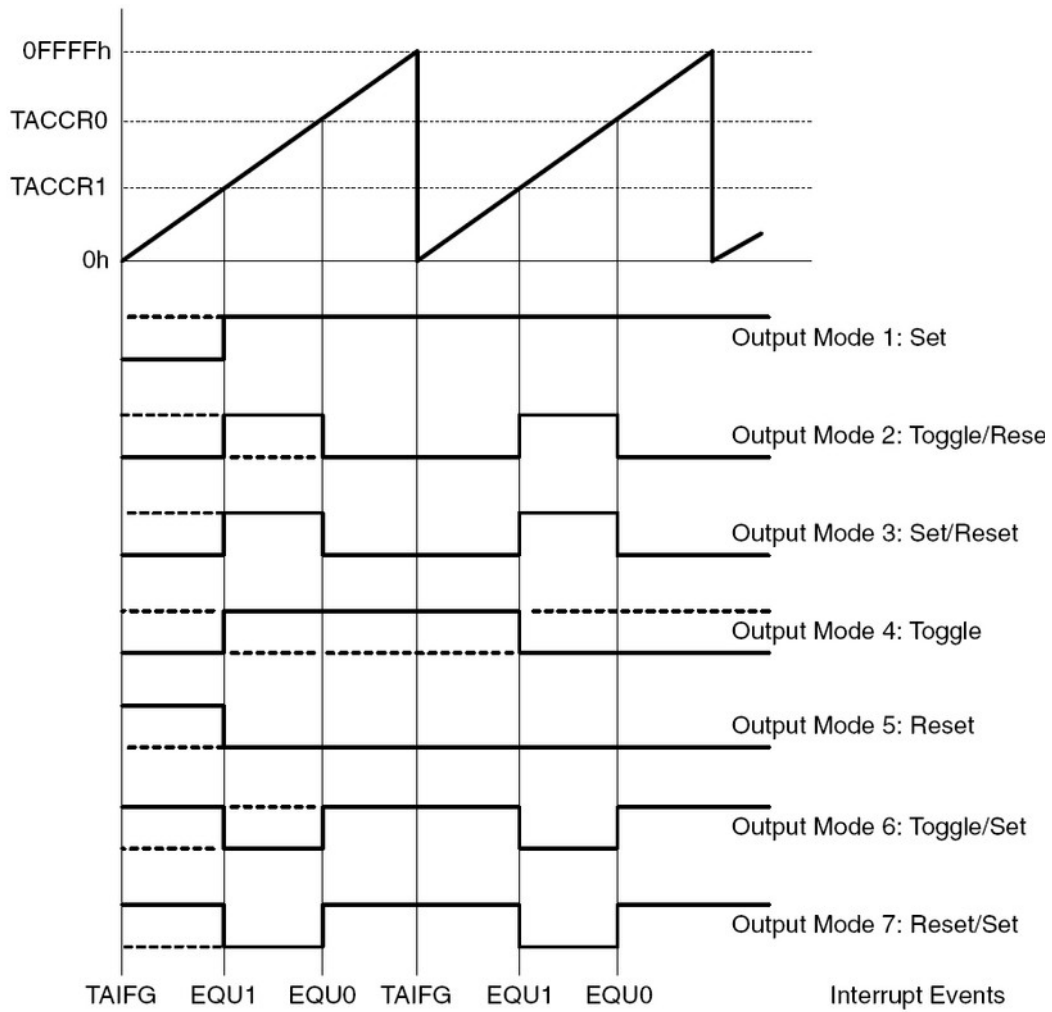
Exemplos de saída em UP/DOWN MODE



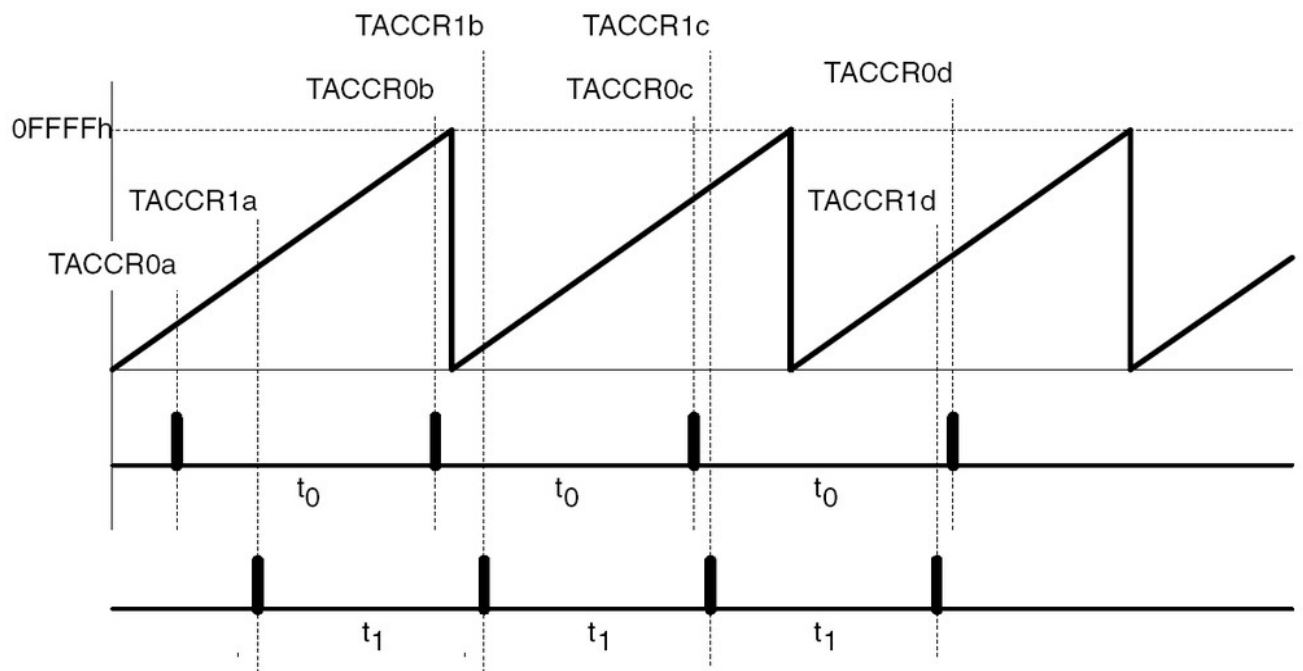
Exemplos de saída em UP MODE



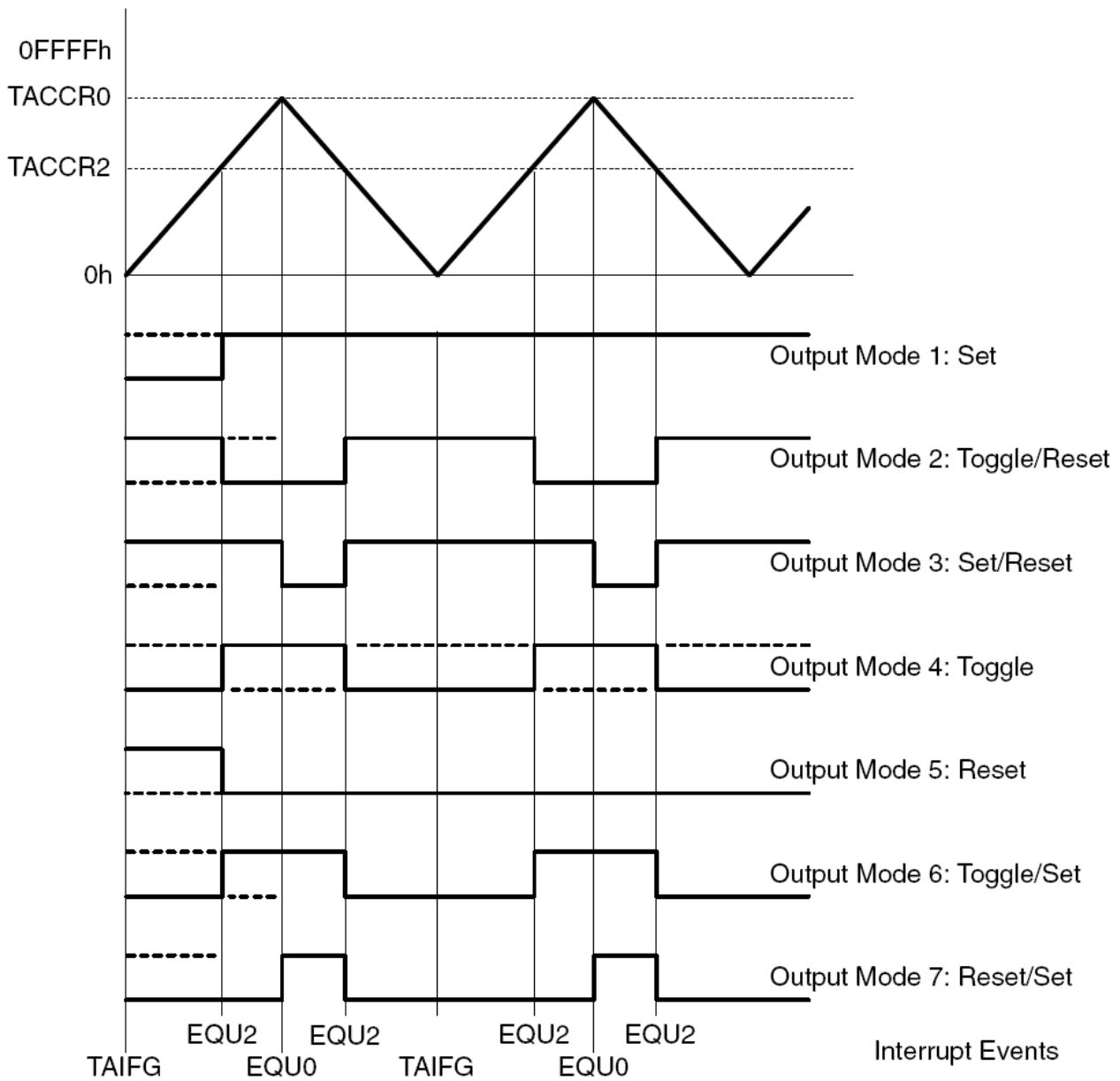
Exemplos de saída em CONTINUOS MODE



Continuous Mode Time Intervals

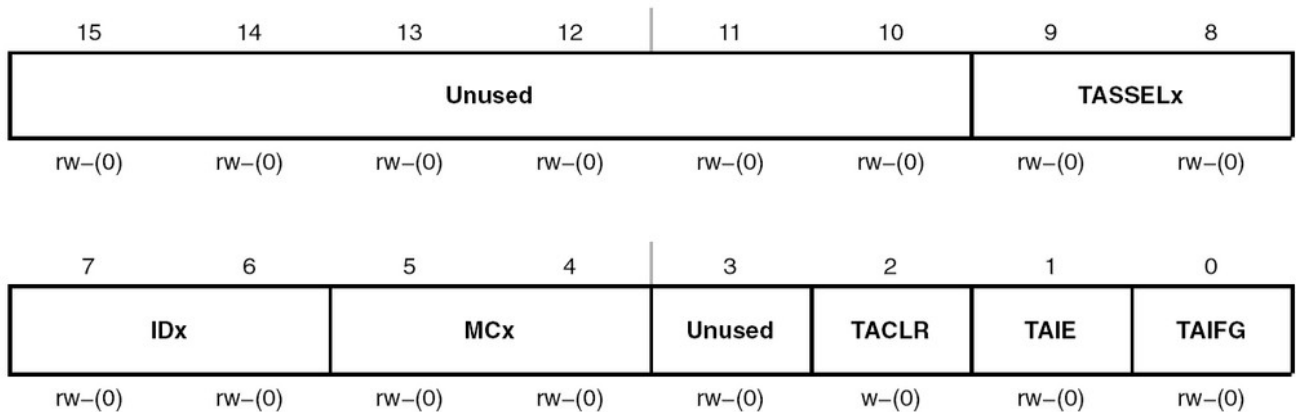


Exemplos de saída em UP/DOWN MODE

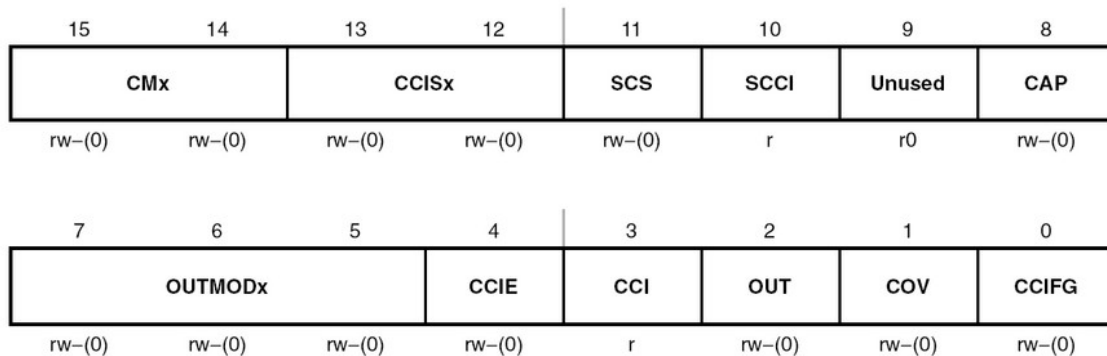


Os registradores de controle do Timer A

TACTL, Timer_A Control Register



Unused	Bits 15-10	Unused
TASSELx	Bits 9-8	Timer_A clock source select 00 TACLK 01 ACLK 10 SMCLK 11 Inverted TACLK
IDx	Bits 7-6	Input divider. These bits select the divider for the input clock. 00 /1 01 /2 10 /4 11 /8
MCx	Bits 5-4	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power. 00 Stop mode: the timer is halted 01 Up mode: the timer counts up to TACCR0 10 Continuous mode: the timer counts up to 0FFFFh 11 Up/down mode: the timer counts up to TACCR0 then down to 0000h
Unused	Bit 3	Unused
TACLx	Bit 2	Timer_A clear. Setting this bit resets TAR, the clock divider, and the count direction. The TACLx bit is automatically reset and is always read as zero.
TAIE	Bit 1	Timer_A interrupt enable. This bit enables the TAIFG interrupt request. 0 Interrupt disabled 1 Interrupt enabled
TAIFG	Bit 0	Timer_A interrupt flag 0 No interrupt pending 1 Interrupt pending

TACCTLx, Capture/Compare Control Register

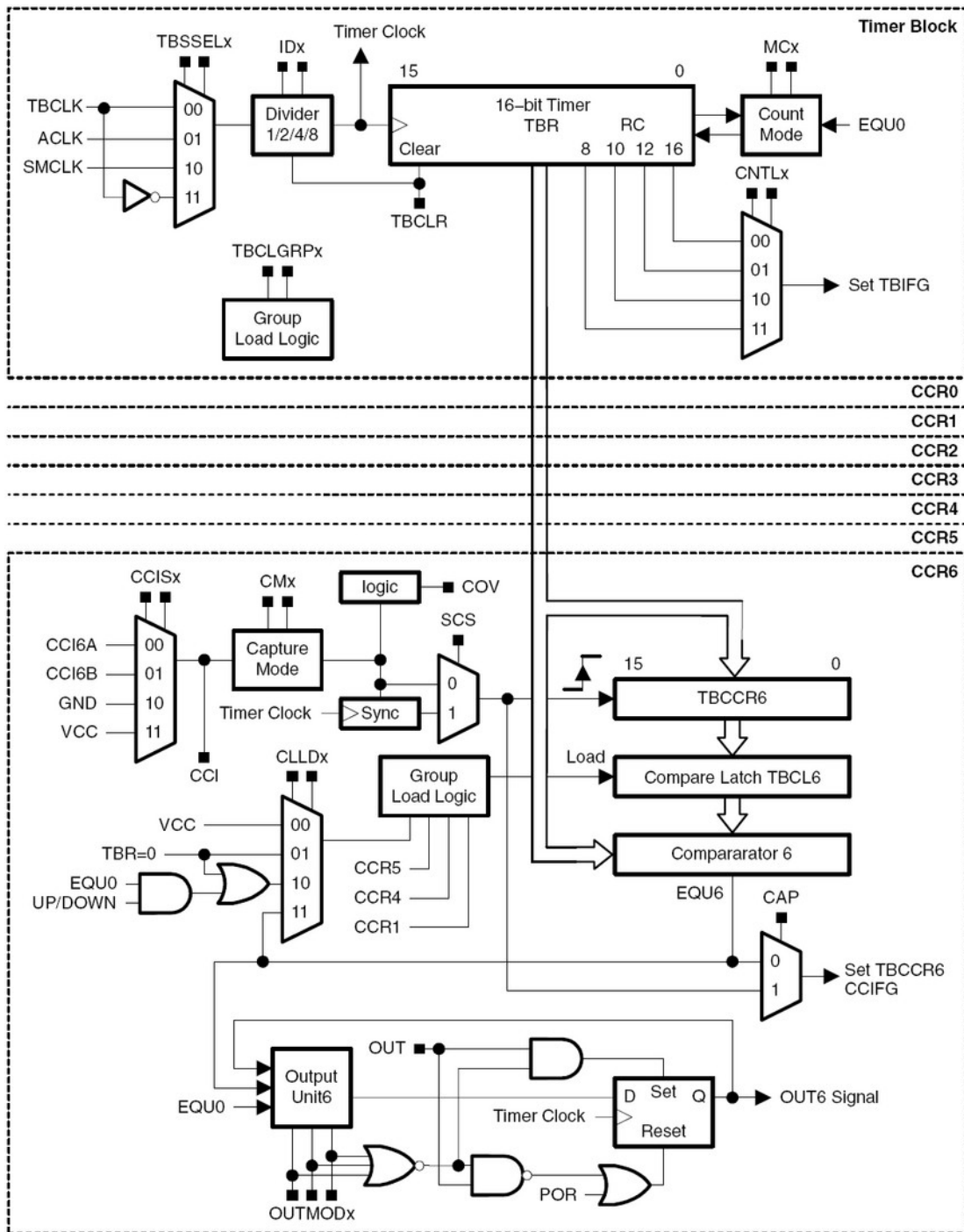
CMx	Bit 15-14	Capture mode 00 No capture 01 Capture on rising edge 10 Capture on falling edge 11 Capture on both rising and falling edges
CCISx	Bit 13-12	Capture/compare input select. These bits select the TACCRx input signal. See the device-specific data sheet for specific signal connections. 00 CCIxA 01 CCIxB 10 GND 11 V _{CC}
SCS	Bit 11	Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock. 0 Asynchronous capture 1 Synchronous capture
SCCI	Bit 10	Synchronized capture/compare input. The selected CCI input signal is latched with the EQUx signal and can be read via this bit
Unused	Bit 9	Unused. Read only. Always read as 0.
CAP	Bit 8	Capture mode 0 Compare mode 1 Capture mode
OUTMODx	Bits 7-5	Output mode. Modes 2, 3, 6, and 7 are not useful for TACCR0 because EQUx = EQU0. 000 OUT bit value 001 Set 010 Toggle/reset 011 Set/reset 100 Toggle 101 Reset 110 Toggle/set 111 Reset/set

CCIE	Bit 4	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag. 0 Interrupt disabled 1 Interrupt enabled
CCI	Bit 3	Capture/compare input. The selected input signal can be read by this bit.
OUT	Bit 2	Output. For output mode 0, this bit directly controls the state of the output. 0 Output low 1 Output high
COV	Bit 1	Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software. 0 No capture overflow occurred 1 Capture overflow occurred
CCIFG	Bit 0	Capture/compare interrupt flag 0 No interrupt pending 1 Interrupt pending

TIMER B

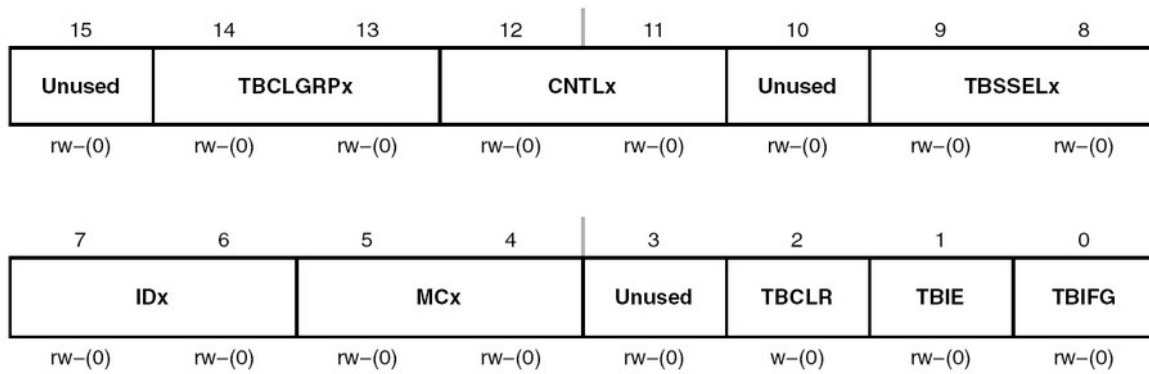
O temporizador Timer B é idêntico ao Timer A, com as seguintes exceções:

- O tamanho do contador pode ser ajustado para 8, 10, 12 ou 16 bits;
- Os registradores TBCCR_x são duplamente buferizados e podem ser agrupados;
- As saídas do Timer B podem ser colocadas em alta impedância;
- O bit SCCI não é implementado neste hardware.

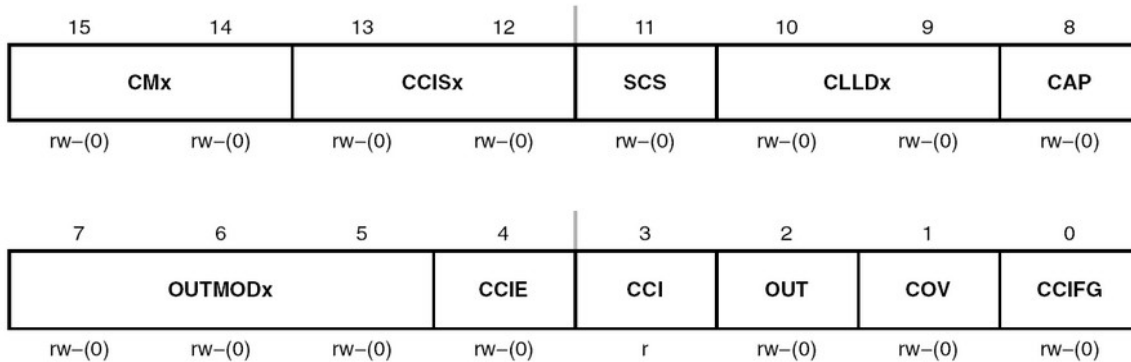


Os registradores de controle do Timer B

Timer_B Control Register TBCTL



Unused	Bit 15	Unused
TBCLGRP	Bit 14-13	TBCLx group
		00 Each TBCLx latch loads independently
		01 TBCL1+TBCL2 (TBCCR1 CLLDx bits control the update) TBCL3+TBCL4 (TBCCR3 CLLDx bits control the update) TBCL5+TBCL6 (TBCCR5 CLLDx bits control the update) TBCL0 independent
		10 TBCL1+TBCL2+TBCL3 (TBCCR1 CLLDx bits control the update) TBCL4+TBCL5+TBCL6 (TBCCR4 CLLDx bits control the update) TBCL0 independent
		11 TBCL0+TBCL1+TBCL2+TBCL3+TBCL4+TBCL5+TBCL6 (TBCCR1 CLLDx bits control the update)
CNTLx	Bits 12-11	Counter Length
		00 16-bit, $TBR_{(max)} = 0FFFFh$
		01 12-bit, $TBR_{(max)} = 0FFFh$
		10 10-bit, $TBR_{(max)} = 03FFh$
		11 8-bit, $TBR_{(max)} = 0FFh$
Unused	Bit 10	Unused
TBSSELx	Bits 9-8	Timer_B clock source select.
		00 TBCLK
		01 ACLK
		10 SMCLK
		11 Inverted TBCLK
IDx	Bits 7-6	Input divider. These bits select the divider for the input clock.
		00 /1
		01 /2
		10 /4
		11 /8
MCx	Bits 5-4	Mode control. Setting MCx = 00h when Timer_B is not in use conserves power.
		00 Stop mode: the timer is halted
		01 Up mode: the timer counts up to TBCL0
		10 Continuous mode: the timer counts up to the value set by TBCNTLx
		11 Up/down mode: the timer counts up to TBCL0 and down to 0000h
Unused	Bit 3	Unused
TBCLR	Bit 2	Timer_B clear. Setting this bit resets TBR, the clock divider, and the count direction. The TBCLR bit is automatically reset and is always read as zero.
TBIE	Bit 1	Timer_B interrupt enable. This bit enables the TBIFG interrupt request.
		0 Interrupt disabled
		1 Interrupt enabled
TBIFG	Bit 0	Timer_B interrupt flag.
		0 No interrupt pending
		1 Interrupt pending

TBCCTLx, Capture/Compare Control Register

CMx	Bit 15-14	Capture mode 00 No capture 01 Capture on rising edge 10 Capture on falling edge 11 Capture on both rising and falling edges
CCISx	Bit 13-12	Capture/compare input select. These bits select the TBCCR _x input signal. See the device-specific datasheet for specific signal connections. 00 CCI _x A 01 CCI _x B 10 GND 11 V _{CC}
SCS	Bit 11	Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock. 0 Asynchronous capture 1 Synchronous capture
CLLDx	Bit 10-9	Compare latch load. These bits select the compare latch load event. 00 TBCL _x loads on write to TBCCR _x 01 TBCL _x loads when TBR <i>counts</i> to 0 10 TBCL _x loads when TBR <i>counts</i> to 0 (up or continuous mode) TBCL _x loads when TBR <i>counts</i> to TBCL ₀ or to 0 (up/down mode) 11 TBCL _x loads when TBR <i>counts</i> to TBCL _x
CAP	Bit 8	Capture mode 0 Compare mode 1 Capture mode
OUTMODx	Bits 7-5	Output mode. Modes 2, 3, 6, and 7 are not useful for TBCL ₀ because EQU _x = EQU ₀ . 000 OUT bit value 001 Set 010 Toggle/reset 011 Set/reset 100 Toggle 101 Reset 110 Toggle/set 111 Reset/set

CCIE	Bit 4	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag. 0 Interrupt disabled 1 Interrupt enabled
CCI	Bit 3	Capture/compare input. The selected input signal can be read by this bit.
OUT	Bit 2	Output. For output mode 0, this bit directly controls the state of the output. 0 Output low 1 Output high
COV	Bit 1	Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software. 0 No capture overflow occurred 1 Capture overflow occurred
CCIFG	Bit 0	Capture/compare interrupt flag 0 No interrupt pending 1 Interrupt pending

• **Exercício 4: Acender o LED em Low Power Mode com PWM**

Já foi identificado no diagrama elétrico da Experimenter Board, mostrado no **item 9**, as conexões abaixo, envolvendo pinos do MSP430FG4618 e hardwares externos. Procure no datasheet do dispositivo a correlação entre os temporizadores A e B e os pinos dos LEDs.

- Botão S1 → pino P1.0 → **TA0**;
- Botão S2 → pino P1.1 → **TA1**;
- LED1 → pino P2.1 → **TB0**;
- LED2 → pino P2.2 → **TB1**;
- LED4 → pino P5.1.

Com estas informações escreva programas em Assembly de modo que sejam executadas as seguintes ações, em **Low Power Mode**:

Conexões do Timer A no MSP430FG4618

Timer_A3 Signal Connections					
Input Pin Number	Device Input Signal	Module Input Name	Module Block	Module Output Signal	Output Pin Number
PZ/QW					PZ/QW
82/B9 - P1.5	TACLK	TACLK	Timer	NA	
	ACLK	ACLK			
	SMCLK	SMCLK			
82/B9 - P1.5	TACLK	INCLK			
87/A7 - P1.0	TA0	CCI0A	CCR0	TA0	87/A7 - P1.0
86/E7 - P1.1	TA0	CCI0B			
	DV _{SS}	GND			
	DV _{CC}	V _{CC}			
85/D7 - P1.2	TA1	CCI1A	CCR1	TA1	85/D7 - P1.2
	CAOUT (internal)	CCI1B			ADC12 (internal)
	DV _{SS}	GND			
	DV _{CC}	V _{CC}			
79/A10 - P2.0	TA2	CCI2A	CCR2	TA2	79/A10 - P2.0
	ACLK (internal)	CCI2B			
	DV _{SS}	GND			
	DV _{CC}	V _{CC}			

Conexões do Timer B no MSP430FG4618

Timer_B7 Signal Connections					
Input Pin Number	Device Input Signal	Module Input Name	Module Block	Module Output Signal	Output Pin Number
PZ/QZW					PZ/QZW
83/B8 - P1.4	TBCLK	TBCLK	Timer	NA	
	ACLK	ACLK			
	SMCLK	SMCLK			
83/B8 - P1.4	TBCLK	INCLK			
78/D8 - P2.1	TB0	CCI0A	CCR0	TB0	78/D8 - P2.1
78/D8 - P2.1	TB0	CCI0B			ADC12 (internal)
	DV _{SS}	GND			
	DV _{CC}	V _{CC}			
77/E8 - P2.2	TB1	CCI1A	CCR1	TB1	77/E8 - P2.2
77/E8 - P2.2	TB1	CCI1B			ADC12 (internal)
	DV _{SS}	GND			
	DV _{CC}	V _{CC}			
76/A11 - P2.3	TB2	CCI2A	CCR2	TB2	76/A11 - P2.3
76/A11 - P2.3	TB2	CCI2B			
	DV _{SS}	GND			
	DV _{CC}	V _{CC}			
67/E12 - P3.4	TB3	CCI3A	CCR3	TB3	67/E12 - P3.4
67/E12 - P3.4	TB3	CCI3B			
	DV _{SS}	GND			
	DV _{CC}	V _{CC}			
66/G9 - P3.5	TB4	CCI4A	CCR4	TB4	66/G9 - P3.5
66/G9 - P3.5	TB4	CCI4B			
	DV _{SS}	GND			
	DV _{CC}	V _{CC}			
65/F11 - P3.6	TB5	CCI5A	CCR5	TB5	65/F11 - P3.6
65/F11 - P3.6	TB5	CCI5B			
	DV _{SS}	GND			
	DV _{CC}	V _{CC}			
64/F12 - P3.7	TB6	CCI6A	CCR6	TB6	64/F12 - P3.7
	ACLK (internal)	CCI6B			
	DV _{SS}	GND			
	DV _{CC}	V _{CC}			

Exercício 4a: 1 Led com PWM em 50%

Fazer o **LED2** → **pino P2.2** → **TB1** piscar utilizando um **PWM** com as seguintes características:

- Largura total do ciclo de trabalho (duty cycle) = **0,5 s**.
- Funcionamento em **50 %** do duty cycle: **0,25 s aceso e 0,25 s apagado**.

Para testar a interrupção do Timer B, faça piscar o **LED1** → **pino P2.1** cada vez que uma interrupção acontecer.

Exercício 4b: 1 Led com PWM ajustável por 2 botões - pressionando

Fazer o **LED2** → **pino P2.2** → **TB1** piscar utilizando um **PWM** com as seguintes características:

- Largura total do ciclo de trabalho (duty cycle) = **1 ms**.
- Funcionamento **inicial** em **50 %** do duty cycle: **0,5 ms aceso e 0,5 ms apagado**.
- Cada vez que o **Botão S1** → **pino P1.0** for pressionado o duty cycle deve ser **incrementado** em **1 %**, até o limite de **99 % aceso e 1 % apagado**.
- Cada vez que o **Botão S2** → **pino P1.2** for pressionado o duty cycle deve ser **decrementado** em **1 %**, até o limite de **1 % aceso e 99 % apagado**.

Exercício 4c: 1 Led com PWM ajustável por 2 botões - contínuo

Fazer o **LED2** → **pino P2.2** → **TB1** piscar utilizando um **PWM** com as seguintes características:

- Largura total do ciclo de trabalho (duty cycle) = **1 ms**.
- Funcionamento **inicial** em **50 %** do duty cycle: **0,5 ms aceso e 0,5 ms apagado**.

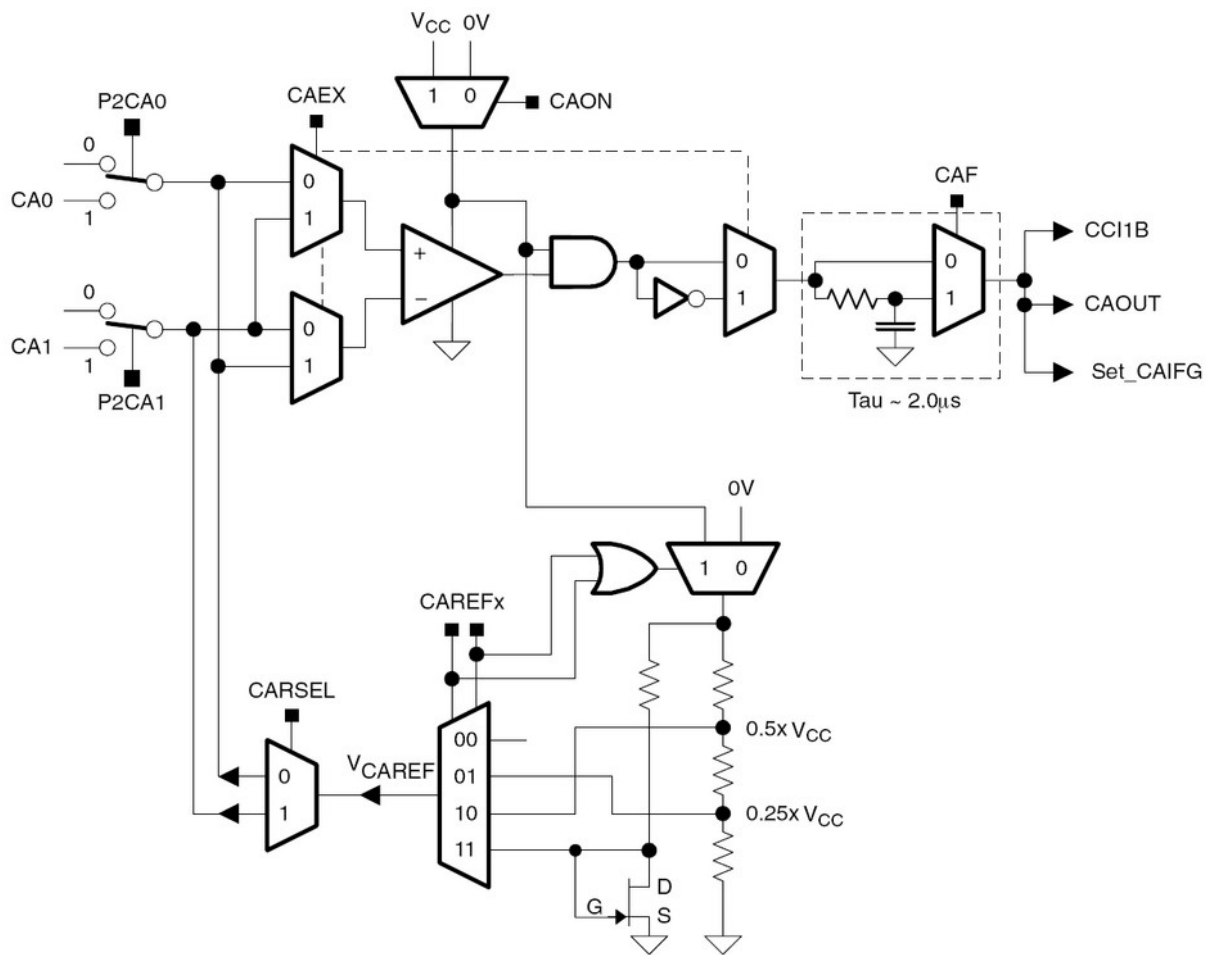
- Cada vez que o **Botão S1 → pino P1.0** for pressionado o duty cycle deve ser **incrementado** em **1 %**, até o limite de **99 % aceso** e **1 % apagado**. Se o botão for mantido pressionado, o incremento deve ser contínuo, até o limite estabelecido.
- Cada vez que o **Botão S2 → pino P1.2** for pressionado o duty cycle deve ser **decrementado** em **1 %**, até o limite de **1 % aceso** e **99 % apagado**. Se o botão for mantido pressionado, o decremento deve ser contínuo, até o limite estabelecido.

Comparador A

O modulo Comparador A permite a comparação entre dois valores de tensões analógicas, inseridas em suas entradas. Por isto ele pode ser utilizado, entre outras coisas, como uma fonte de conversão precisa de sinais analógicos para digital, utilizando o método slope/interception, monitor de tensões analógicas, etc.

Algumas de suas características são:

- Terminais multiplexados de entrada inversor e não inversor;
- Seleção de filtro RC por software, para a saída de comparação;
- A saída pode ser utilizada na entrada de captura do TIMER A;
- Controle por software do buffer da porta de entrada;
- Capacidade para gerar interrupções;
- Gerador de voltagem de referência selecionável;
- O comparador e o gerador de sinal de referência podem ser desligados.



Os Registradores de controle do Comparador

A

CACTL1, Comparator_A Control Register 1

7	6	5	4	3	2	1	0
CAEX	CARSEL	CAREF _x		CAON	CAIES	CAIE	CAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

CAEX	Bit 7	Comparator_A exchange. This bit exchanges the comparator inputs and inverts the comparator output.
CARSEL	Bit 6	Comparator_A reference select. This bit selects which terminal the V_{CAREF} is applied to. When CAEX = 0: 0 V_{CAREF} is applied to the + terminal 1 V_{CAREF} is applied to the - terminal When CAEX = 1: 0 V_{CAREF} is applied to the - terminal 1 V_{CAREF} is applied to the + terminal
CAREF	Bits 5-4	Comparator_A reference. These bits select the reference voltage V_{CAREF} . 00 Internal reference off. An external reference can be applied. 01 $0.25 \cdot V_{CC}$ 10 $0.50 \cdot V_{CC}$ 11 Diode reference is selected
CAON	Bit 3	Comparator_A on. This bit turns on the comparator. When the comparator is off it consumes no current. The reference circuitry is enabled or disabled independently. 0 Off 1 On
CAIES	Bit 2	Comparator_A interrupt edge select 0 Rising edge 1 Falling edge
CAIE	Bit 1	Comparator_A interrupt enable 0 Disabled 1 Enabled
CAIFG	Bit 0	The Comparator_A interrupt flag 0 No interrupt pending 1 Interrupt pending

CACTL2, Comparator_A Control Register 2

7	6	5	4	3	2	1	0
Unused				P2CA1	P2CA0	CAF	CAOUT
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)

Unused	Bits 7-4	Unused.
P2CA1	Bit 3	Pin to CA1. This bit selects the CA1 pin function. 0 The pin is not connected to CA1 1 The pin is connected to CA1
P2CA0	Bit 2	Pin to CA0. This bit selects the CA0 pin function. 0 The pin is not connected to CA0 1 The pin is connected to CA0
CAF	Bit 1	Comparator_A output filter 0 Comparator_A output is not filtered 1 Comparator_A output is filtered
CAOUT	Bit 0	Comparator_A output. This bit reflects the value of the comparator output. Writing this bit has no effect.

CAPD, Comparator_A Port Disable Register

7	6	5	4	3	2	1	0
CAPD7	CAPD6	CAPD5	CAPD4	CAPD3	CAPD2	CAPD1	CAPD0
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

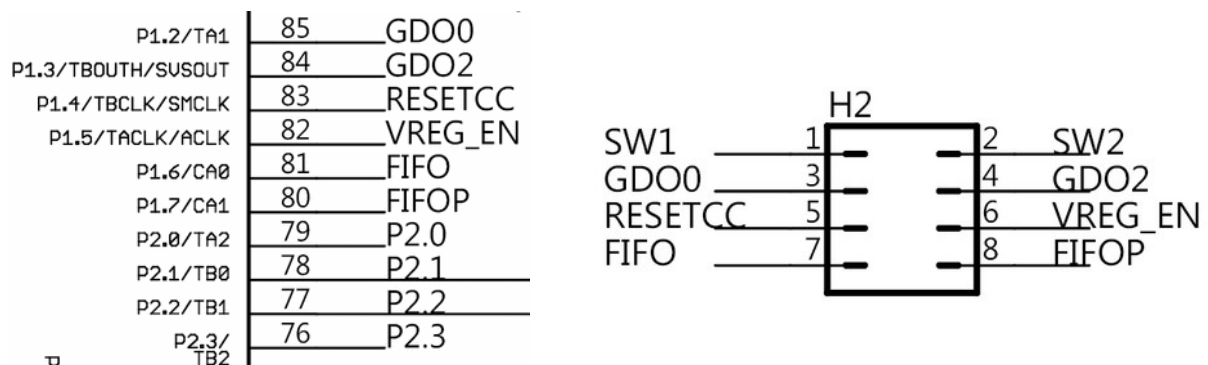
CAPDx	Bits 7-0	Comparator_A port disable. These bits individually disable the input buffer for the pins of the port associated with Comparator_A. For example, the CAPDx bits can be used to individually enable or disable each P1.x pin buffer. CAPD0 disables P1.0, CAPD1 disables P1.1, etc. 0 The input buffer is enabled. 1 The input buffer is disabled.
--------------	-------------	--

Exercício 5: Comparar as tensões de entrada com um referencial

Já foi identificado no diagrama elétrico da Experimenter Board, mostrado no [item 9](#), as conexões abaixo, envolvendo pinos do MSP430FG4618 e hardwares externos. Procure no datasheet do dispositivo a correlação entre o módulo comparador e os pinos dos LEDs.

- | | |
|--------------------------|-----------------------|
| a) Botão S1 → pino P1.0; | d) LED2 → pino P2.2; |
| b) Botão S2 → pino P1.1; | e) LED4 → pino P5.1. |
| c) LED1 → pino P2.1; | f) Buzzer → pino P3.5 |

Note que as duas entradas do módulo comparador estão nos **pinos 81 (P1.6 → CA0) e 80 (P1.7 → CA1)** do dispositivo. É possível acessar e inserir níveis de tensão no comparador através dos pinos **7 (FIFO → CA0) e 8 (FIFOP → CA1)** do **conector H2**, localizado ao lado do conector JTAG do MSP430FG4618, na Experimenter Board, como pode ser visto nos detalhes abaixo.



Como já vem sendo explorado ao longo do treinamento, estes exercícios devem ser resolvidos em LPM, de modo a fazer a máxima economia de energia possível.

Exercício 5a: Acender o Led toda vez que a tensão passar de um patamar

Escreva um programa em *Assembly* que monitore a tensão presente na entrada CA0 → P1.6 → conector H2 → pino 7 → FIFO, em referência a metade do valor da tensão de alimentação ($0,5*V_{CC}$), e que apresente a seguinte saída:

- $V_{CA0} > 0,5*V_{CC}$ → LED1 → pino P2.1 aceso e LED2 → pino P2.2 apagado.
- $V_{CA0} < 0,5*V_{CC}$ → LED1 → pino P2.1 apagado e LED2 → pino P2.2 aceso.

Exercício 5b: Tocar o Buzzer avisando que a tensão está baixa

Escreva um programa em *Assembly* que monitore a tensão presente na entrada CA1 → P1.7 → conector H2 → pino 8 → FIFOP, em referência a $\frac{1}{4}$ do valor da tensão de alimentação ($0,25*V_{CC}$), e que apresente a seguinte saída:

- $V_{CA1} > 0,25*V_{CC}$ → Buzzer → pino P3.5 desligado e LED4 → pino P5.1 aceso.
- $V_{CA1} < 0,25*V_{CC}$ → Buzzer → pino P3.5 ligado e LED4 → pino P5.1 apagado.

Amplificadores Operacionais

O MSP430FG4618 tem três módulos idênticos de Amplificadores Operacionais construídos internamente, com seus pontos de entrada e saída acessíveis através de pinos do uC. O uso de cada um destes módulos pode ser feito de modo independente ou agrupados, de acordo com as configurações que o usuário programa nos registradores de controle.

Os pinos associados aos módulos podem ser vistos na tabela abaixo.

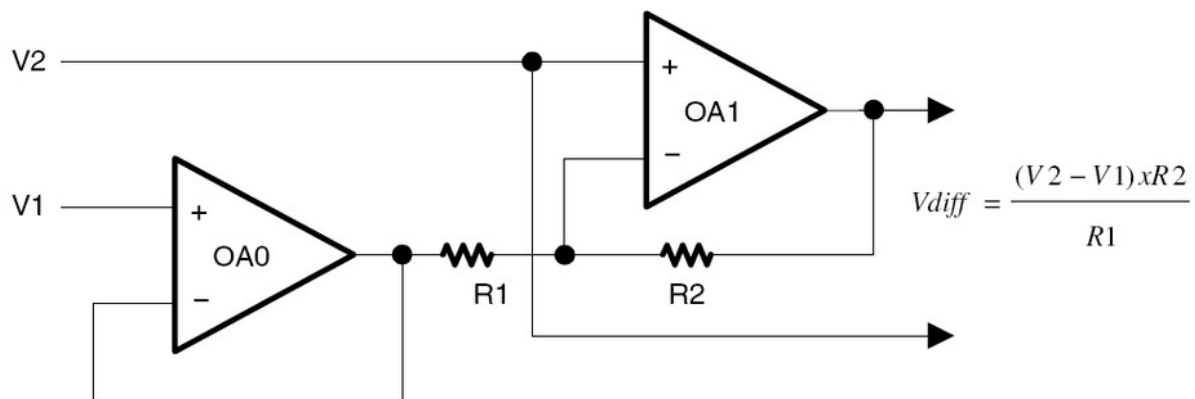
OA Signal Connections						
Input Pin Number	Device Input Signal	Module Input Name	Module Block	Module Output Signal	Device Output Signal	Output Pin Number
PZ						PZ
95 - P6.0	OA0I0	OA0I0	OA0	OA0OUT	OA0O	96 - P6.1
97 - P6.2	OA0I1	OA0I1			OA0O	ADC12 (internal)
	DAC12_0OUT (internal)	DAC12_0OUT				
	DAC12_1OUT (internal)	DAC12_1OUT				
3 - P6.4	OA1I0	OA1I0	OA1	OA1OUT	OA1O	2 - P6.3
13 - P5.0	OA1I1	OA1I1			OA1O	13- P5.0
	DAC12_0OUT (internal)	DAC12_0OUT			OA1O	ADC12 (internal)
	DAC12_1OUT (internal)	DAC12_1OUT				
5 - P6.6	OA2I0	OA2I0	OA2	OA2OUT	OA2O	4 - P6.5
14 - P10.7	OA2I1	OA2I1			OA2O	14 - P10.7
	DAC12_0OUT (internal)	DAC12_0OUT			OA2O	ADC12 (internal)
	DAC12_1OUT (internal)	DAC12_1OUT				

Na figura da próxima página é possível encontrar o diagrama de cada um dos três módulos de AO.

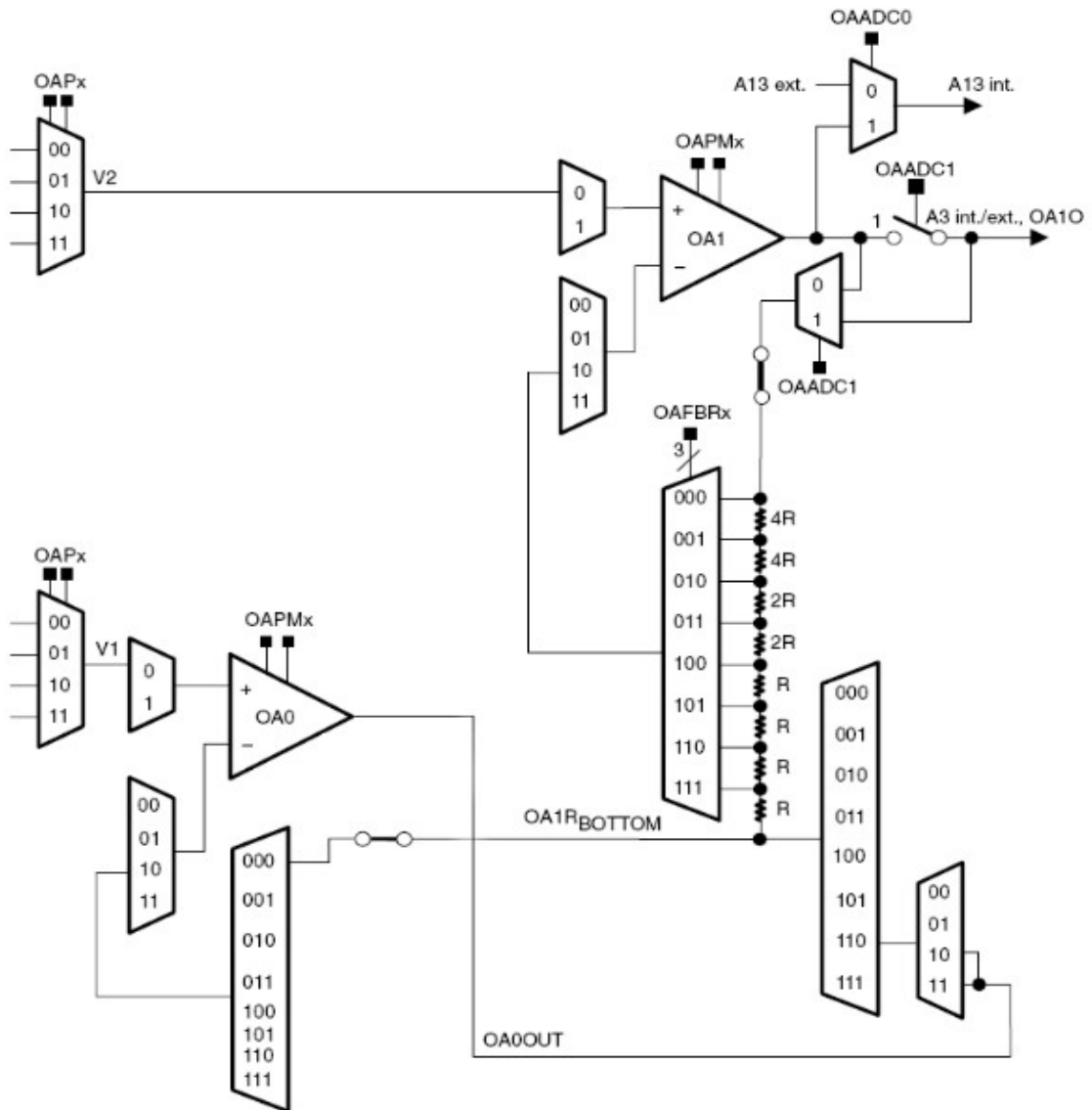
Estes módulos podem ser configurados para funcionar em até seis modos de operação diferentes, de acordo com o valor programado nos bits OAFCx, presentes no registrador OAxCTL1:

OAFCx	OA Mode
000	General-purpose op amp
001	Unity gain buffer
010	Reserved
011	Comparator
100	Non-inverting PGA amplifier
101	Reserved
110	Inverting PGA amplifier
111	Differential amplifier

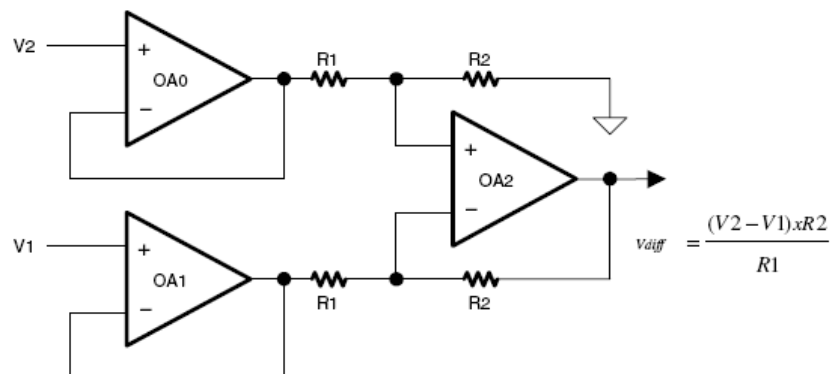
Quando os módulos são configurados para funcionar no modo Amplificador Diferencial, podem ser consideradas conexões com dois AOs ou três AOs. Na figura a seguir, tem-se a opção em modo Amplificador Diferencial com dois AOs:



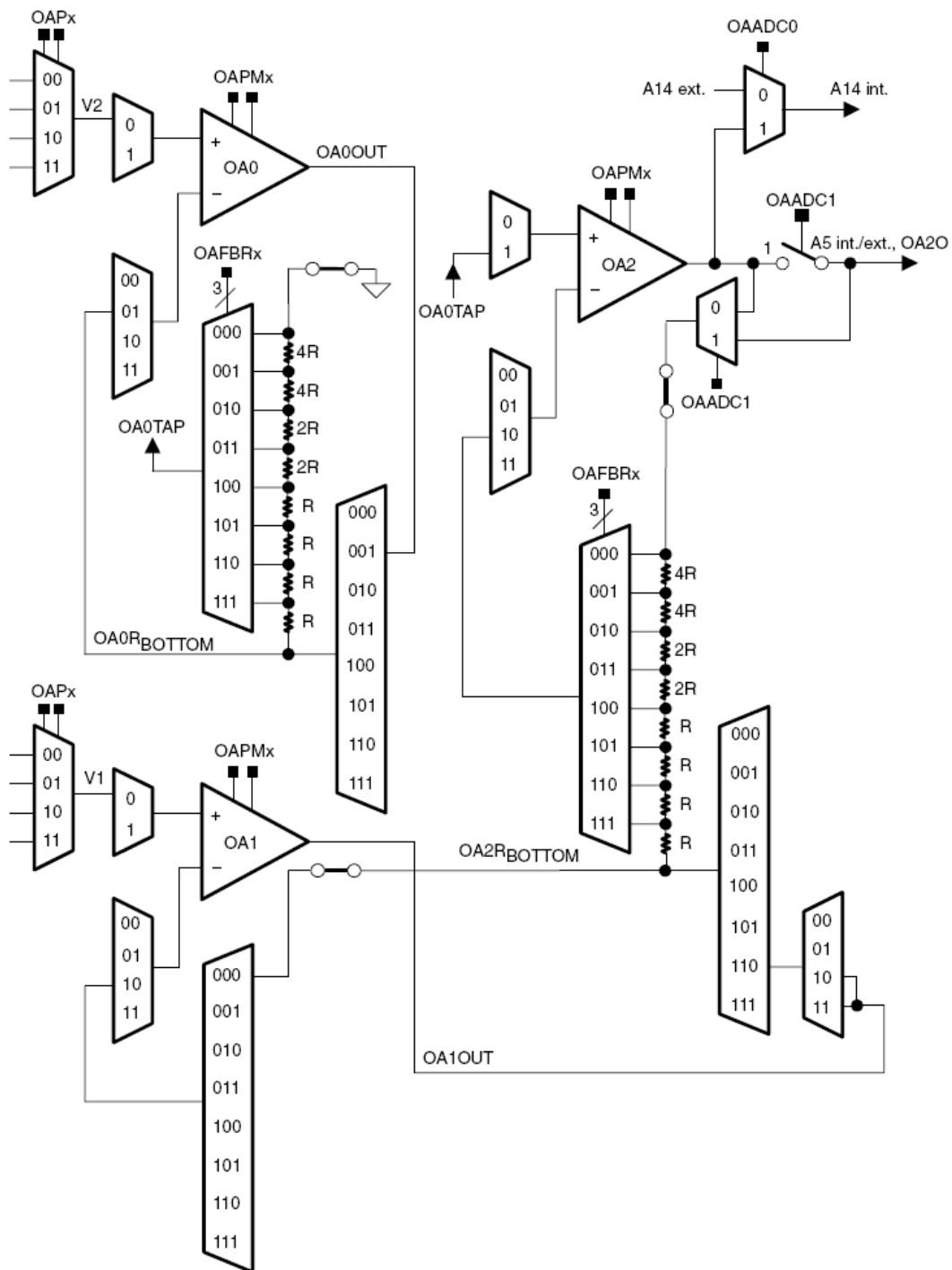
E o diagrama de conexão interna em modo Amplificador Diferencial com dois Aos:



Na figura a seguir, tem-se a opção em modo Amplificador Diferencial com três AOs:



E o diagrama de conexão interna em modo Amplificador Diferencial com dois AOs:



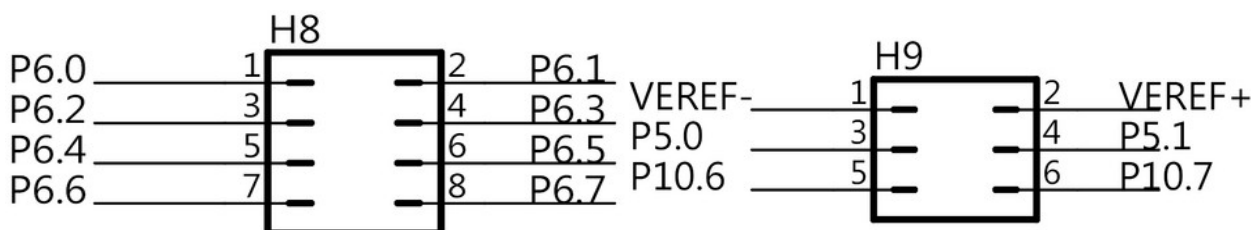
Exercício 6: Funcionamento do A.O. no MSP430

Já foi identificado no diagrama elétrico da Experimenter Board, mostrado no **item 9**, as conexões abaixo, envolvendo pinos do MSP430FG4618 e hardwares externos. Procure no datasheet do dispositivo a correlação entre o módulo comparador e os pinos dos LEDs.

- Botão S1 → pino P1.0;
- Botão S2 → pino P1.1;
- LED1 → pino P2.1;
- LED2 → pino P2.2;
- LED4 → pino P5.1.
- Buzzer → pino P3.5

Os pinos associados aos módulos de Amp. Op. No MSP430FG4618 podem ser vistos na tabela abaixo, cujo acesso é obtido através dos conectores H8 e H9.

OA Signal Connections						
Input Pin Number	Device Input Signal	Module Input Name	Module Block	Module Output Signal	Device Output Signal	Output Pin Number
PZ						PZ
95 - P6.0	OA0I0	OA0I0	OA0	OA0OUT	OA0O	96 - P6.1
97 - P6.2	OA0I1	OA0I1			OA0O	ADC12 (internal)
	DAC12_0OUT (internal)	DAC12_0OUT				
	DAC12_1OUT (internal)	DAC12_1OUT				
3 - P6.4	OA1I0	OA1I0	OA1	OA1OUT	OA1O	2 - P6.3
13 - P5.0	OA1I1	OA1I1			OA1O	13- P5.0
	DAC12_0OUT (internal)	DAC12_0OUT			OA1O	ADC12 (internal)
	DAC12_1OUT (internal)	DAC12_1OUT				
5 - P6.6	OA2I0	OA2I0	OA2	OA2OUT	OA2O	4 - P6.5
14 - P10.7	OA2I1	OA2I1			OA2O	14 - P10.7
	DAC12_0OUT (internal)	DAC12_0OUT			OA2O	ADC12 (internal)
	DAC12_1OUT (internal)	DAC12_1OUT				



Exercício 6a: Módulo AO0 como comparador

Escreva um programa em *Assembly* que **configure o módulo AO0 como comparador**. Após realizar as configurações necessárias, o MSP430 deve ser levado a LPM3, e passa a operar apenas com o AO. Note que serão utilizadas as seguintes conexões:

Entrada:

- AO0_{+input} → P6.0 → conector H8 → pino 1.

Saída:

- $AO0_{output} \rightarrow P6.1 \rightarrow \text{conector H8} \rightarrow \text{pino 2}$

Este programa deverá monitorar a tensão presente na entrada **positiva** do módulo. O valor de referência que deve estar presente na entrada **negativa** deste módulo será **um quarto do valor da tensão de alimentação ($0,25 \cdot V_{CC}$)**.

Caso o valor na entrada positiva do AO0 seja maior que a referência ($V_{AO0 + input} > 0,25 \cdot V_{CC}$), a saída do módulo deve ficar em nível lógico 1, o que fará o **LED4 \rightarrow pino P5.1** ficar **aceso**.

Será necessário usar um jumper para este programa ser testado na placa.

Exercício 6b: Módulo AO1 como buffer de ganho unitário

Escreva um programa em *Assembly* que **configure o módulo AO0 como buffer de ganho unitário**. Após realizar as configurações necessárias, o MSP430 deve ser levado a LPM3, e passa a operar apenas com o AO. Note que serão utilizadas as seguintes conexões:

Entrada:

- $AO1_{+input} \rightarrow P6.4 \rightarrow \text{conector H8} \rightarrow \text{pino 5}$

Saída:

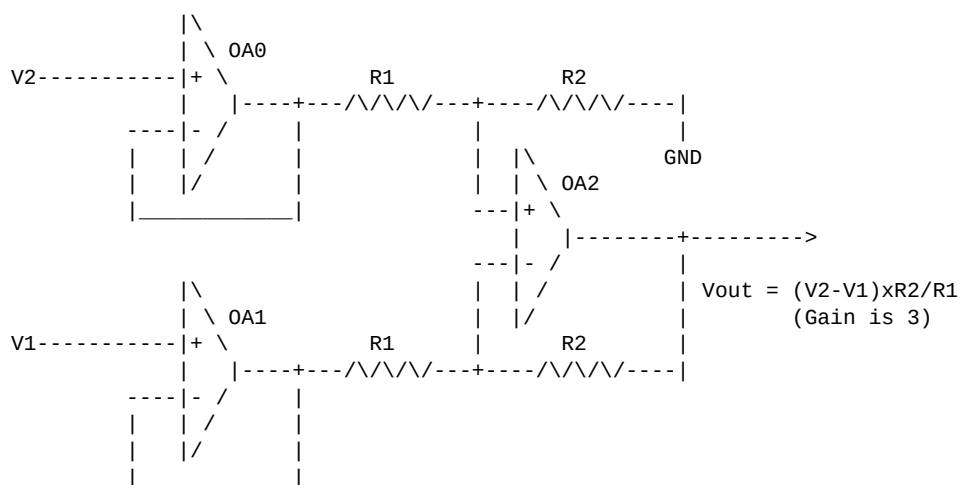
- $AO1_{output} \rightarrow P6.3 \rightarrow \text{conector H8} \rightarrow \text{pino 4}$

Este programa deverá monitorar a tensão presente na entrada **positiva** do módulo. O valor de tensão na saída deverá copiar, com ganho unitário, o valor da tensão aplicada na entrada.

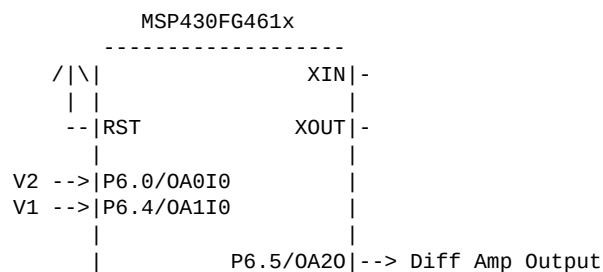
Será necessário usar um osciloscópio digital, para este programa ser testado na placa.

Exercício 6c: Módulos AO0, AO1 e AO2 como Amplificador diferencial

Escreva um programa em Assembly que configure os **módulos AO0, AO1 e AO2 como amplificador diferencial**, de acordo com o seguinte diagrama:



Após realizar as configurações necessárias, o MSP430 deve ser levado a LPM3, e passa a operar apenas com o AO. Note que serão utilizadas as seguintes conexões:



Entrada:

- AO0_{+input} → P6.0 → conector H8 → pino 1.
- AO1_{+input} → P6.4 → conector H8 → pino 5.

Saída:

- AO2_{output} → P6.5 → conector H8 → pino 6

Este programa deverá monitorar as tensões presentes nas entradas **positivas** dos módulos AO0 e AO1. O valor de tensão na saída deverá ter um ganho igual a 3, em relação a diferença entre as tensões na entrada.

Será necessário usar um osciloscópio digital, para este programa ser testado na placa.

Anexos

Para você conhecer os outros livros sobre eletrônica do Instituto Newton C. Braga.

Acesse :

www.newtoncbraga.com.br/index.php/livros-tecnicos

Ou fotografe o QR abaixo:

