■ MSP430 FRAM的优势介绍

■ 使用MSP430 FRAM 的注意事项

■ 如何灵活分配FRAM在数据存储和程序存储的应用。

■ MSP430 FRAM 的IPE模块应用

——陈冰

MSP430 FRAM的优势

1. FRAM 的优势？ 至少3个耳熟能详的特性！

★擦写次数特别多 10^15

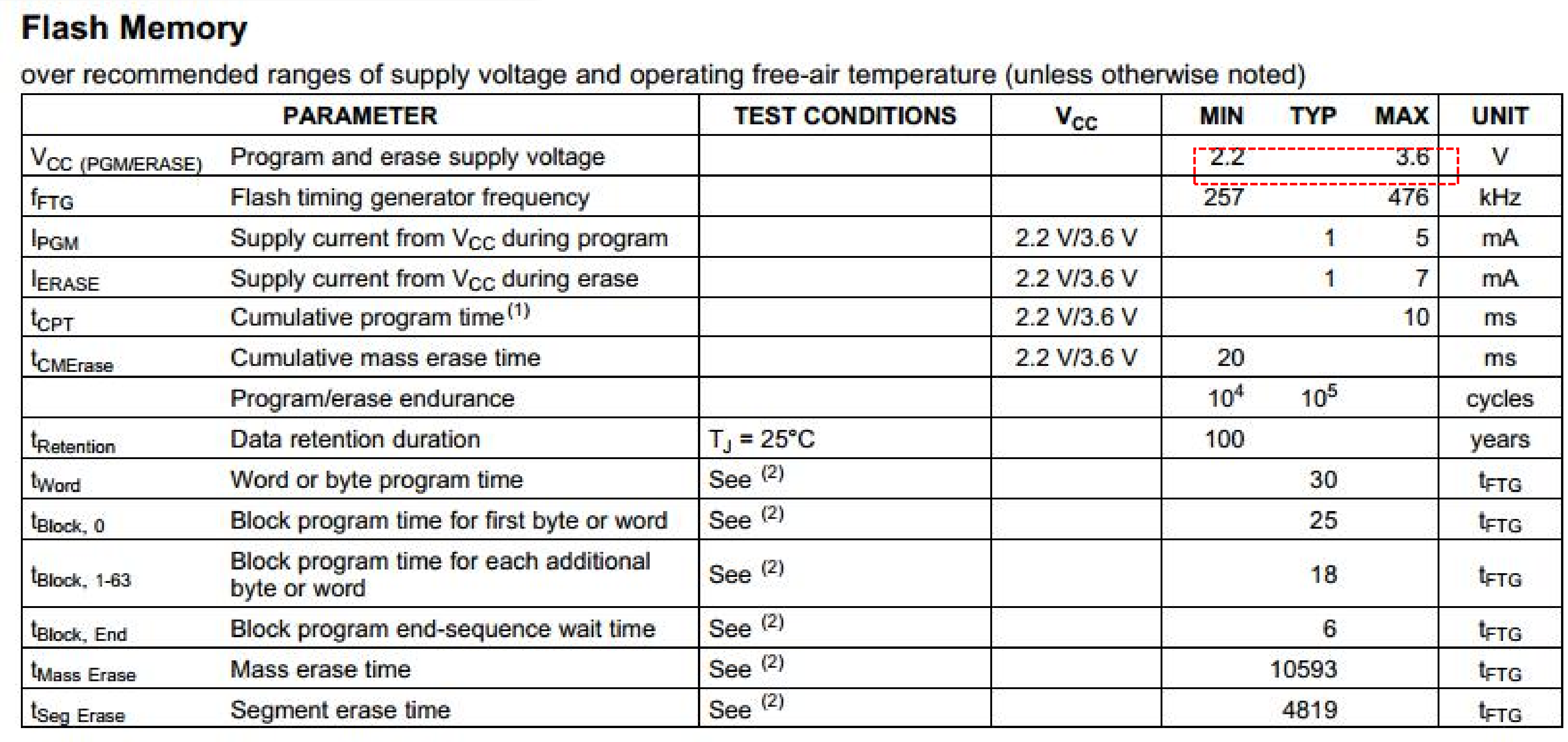
★容易赋值操作，像RAM？

★ 掉电不丢失

2. MSP430FRxx俗称”金刚狼“，业界第一款以FRAM做程序存储的MCU。怎么使用优势？

◆编程电压：

大部分Flash的编程电压都是5V以上。



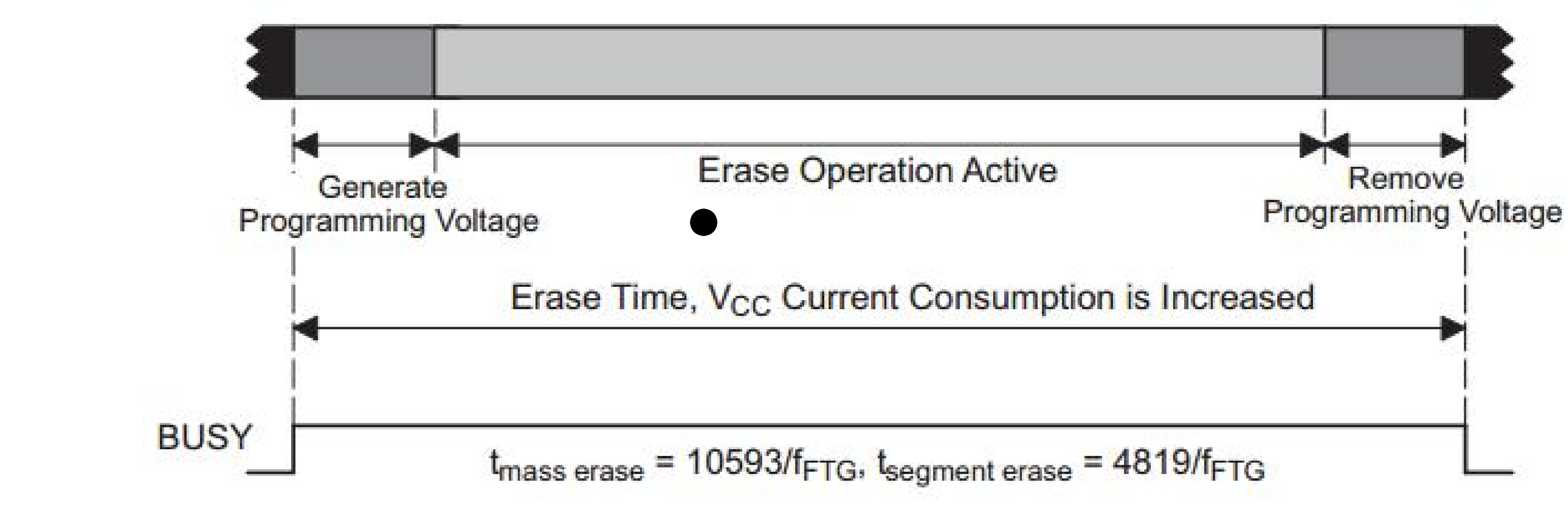
Flash Memory

over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER		TEST CONDITIONS	V_{CC}	MIN	TYP	MAX	UNIT
$V_{CC\ (PGM/ERASE)}$	Program and erase supply voltage			2.2		3.6	V
f_{FTG}	Flash timing generator frequency			257		476	kHz
I_{PGM}	Supply current from V_{CC} during program		2.2 V/3.6 V		1	5	mA
I_{ERASE}	Supply current from V_{CC} during erase		2.2 V/3.6 V		1	7	mA
t_{CPT}	Cumulative program time[1]		2.2 V/3.6 V			10	ms
$t_{CMErase}$	Cumulative mass erase time		2.2 V/3.6 V	20			ms
	Program/erase endurance			10^4	10^5		cycles
$t_{Retention}$	Data retention duration	T_J = 25°C		100			years
t_{Word}	Word or byte program time	See [2]			30		t_{FTG}
$t_{Block,\ 0}$	Block program time for first byte or word	See [2]			25		t_{FTG}
$t_{Block,\ 1-63}$	Block program time for each additional byte or word	See [2]			18		t_{FTG}
$t_{Block,\ End}$	Block program end-sequence wait time	See [2]			6		t_{FTG}
$t_{Mass\ Erase}$	Mass erase time	See [2]			10593		t_{FTG}
$t_{Seg\ Erase}$	Segment erase time	See [2]			4819		t_{FTG}

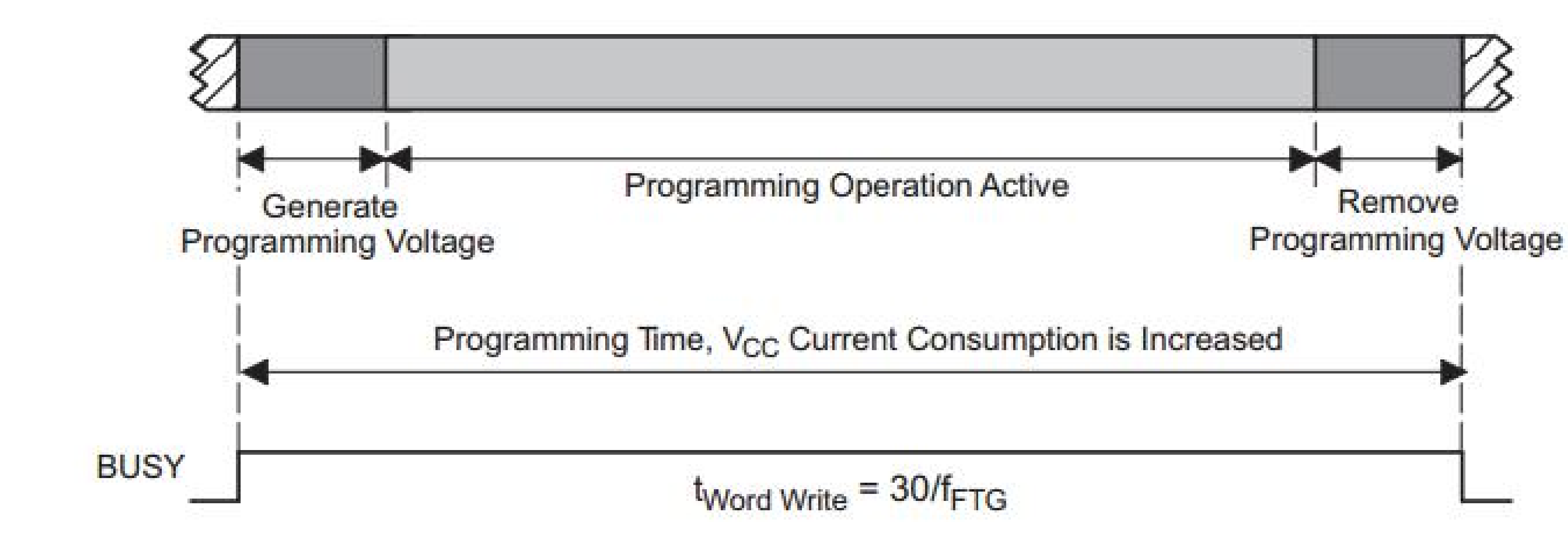
MSP430F2xx 为例

- Erase


Generate Programming Voltage
Erase Operation Active
Remove Programming Voltage
Erase Time, V_{CC} Current Consumption is Increased
BUSY
$t_{mass\ erase} = 10593/f_{FTG}$, $t_{segment\ erase} = 4819/f_{FTG}$

```
  Void Flash_Erase(u16_t EraseAdr)
  {
     u16_t* Flash_ptrA;
     Flash_ptrA = (u16_t *) EraseAdr;     // Point to beginning of seg
     FCTL2 = FWKEY + FSSEL0 + FN1;        // 配置时钟，使得FLASH时钟在
257Khz~476KHz
  FCTL3 = FWKEY;                    //FLASH解锁
  FCTL1 = FWKEY + ERASE;            // Set Erase bit
  *Flash_ptrA = 0x00;               // Dummy write to erase Flash seg
  while(FCTL3 & BUSY);              //等待擦除操作完成，变空闲
  FCTL3 = FWKEY + LOCK;             // 重新上锁
  }
```

- Write


Generate
Programming Voltage
Programming Operation Active
Remove
Programming Voltage
Programming Time, V_{CC} Current Consumption is Increased
BUSY
$t_{Word\ Write} = 30/f_{FTG}$

```
   Void Flash_Write(u16_t WriteAdr, u16_t* DataBuf,u16_t WriteByts)
   {
   u16_t i, Flash_ptrA*;
   Flash_ptrA = (u16_t *) WriteAdr; // Point to beginning of seg
   FCTL2 = FWKEY + FSSEL0 + FN1; // 配置时钟，使得FLASH时钟在
257Khz~476KHz
   FCTL3 = FWKEY;                //FLASH解锁
   FCTL1 = FWKEY + WRT;          // Set Write bit
   for(i=0;i< WriteByts;i++)
       {
       *Flash_ptrA++ = *DataBuf++;
       while( !(FCTL3 & WAIT) );
       }
   while(FCTL3 & BUSY);               //等待擦除操作完成，变空闲
   FCTL3 = FWKEY + LOCK;               // 重新上锁
   }
```

MSP430 FRAM的优势

从这里我们可以看到需要的操作有：

1） FLASH时钟设定

2） FLASH的解锁

3） 选择FLASH操作，是擦除还是写

4） 启动擦除或者写操作（启动Charge Pump）

5） 等待操作完成

6） FLASH重新加锁，并退出。

MSP430 FRAM的优势

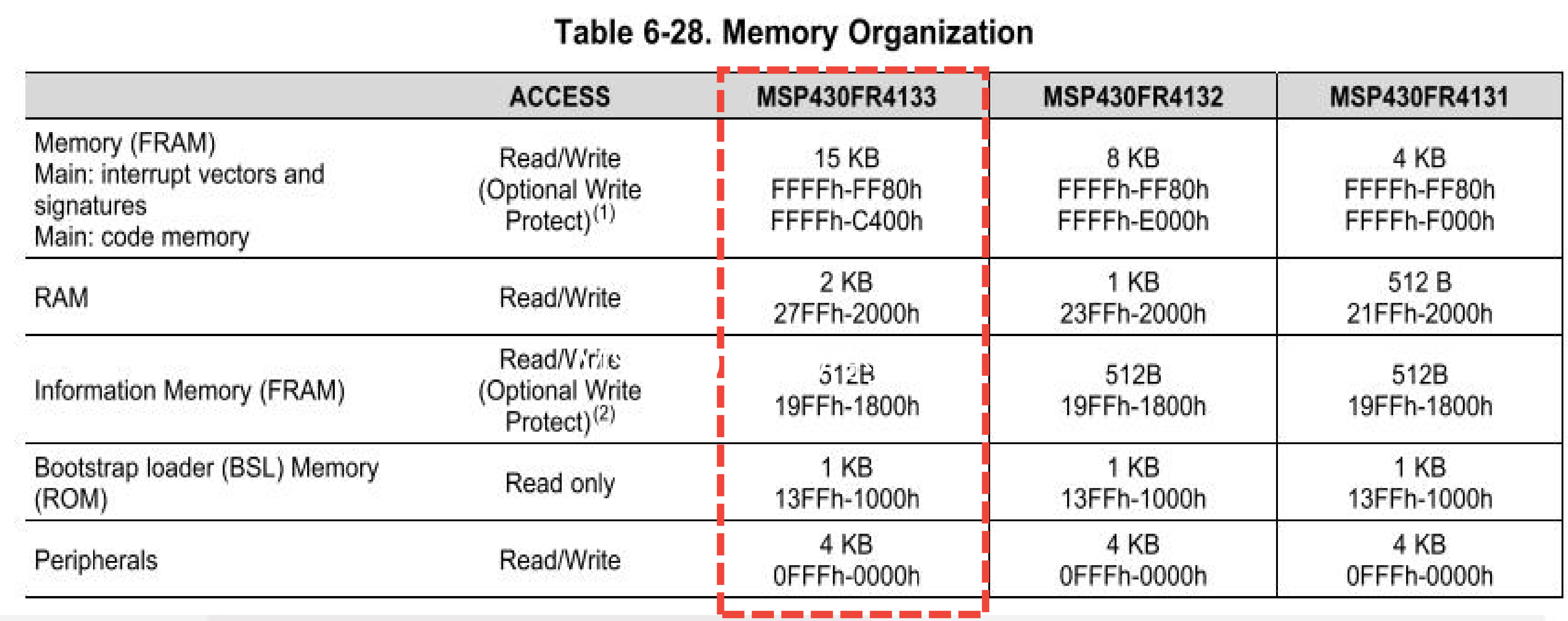


Table 6-28. Memory Organization

	ACCESS	MSP430FR4133	MSP430FR4132	MSP430FR4131
Memory (FRAM) Main: interrupt vectors and signatures Main: code memory	Read/Write (Optional Write Protect)[1]	15 KB FFFFh-FF80h FFFFh-C400h	8 KB FFFFh-FF80h FFFFh-E000h	4 KB FFFFh-FF80h FFFFh-F000h
RAM	Read/Write	2 KB 27FFh-2000h	1 KB 23FFh-2000h	512 B 21FFh-2000h
Information Memory (FRAM)	Read/Write (Optional Write Protect)[2]	512B 19FFh-1800h	512B 19FFh-1800h	512B 19FFh-1800h
Bootstrap loader (BSL) Memory (ROM)	Read only	1 KB 13FFh-1000h	1 KB 13FFh-1000h	1 KB 13FFh-1000h
Peripherals	Read/Write	4 KB 0FFFh-0000h	4 KB 0FFFh-0000h	4 KB 0FFFh-0000h

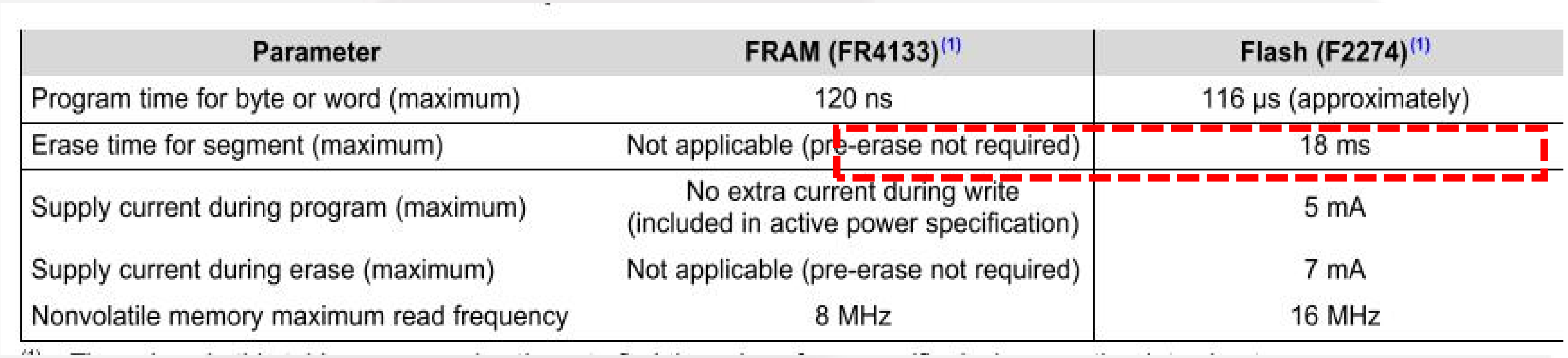
以MSP430FR4133为例：

0xF000这个地址在MSP430FR4133内属于程序存储区，它是FRAM。如果需要将这个内存单元“擦除”，只要

```
void Fram_Erase(void )
{
   u16_t* ErasePtr = (u16_t *) (0xF000);
  *ErasePtr = 0xFFFF;
}
```

MSP430 FRAM的优势

FLASH只能从1“写”0， 要将数据从“0”恢复到“1”时，只能选择“擦除”操作，擦除操作才是将“0”改成“1”的操作。此外，擦除的最小单位是一个Segment,这就导致我们即使误写了一个bit就需要擦除整个segment。从MSP430的角度来讲的话，只要一个bit出错（主FLASH内），即需要擦除512个字节。这种情况不仅导致不必要的擦除操作（相比FRAM、RAM而言）导致FLASH寿命降低，另外，在数据频繁存储的应用中它还将明显的增加功耗



Parameter	FRAM (FR4133)[1]	Flash (F2274)[1]
Program time for byte or word (maximum)	120 ns	116 µs (approximately)
Erase time for segment (maximum)	Not applicable (pre-erase not required)	18 ms
Supply current during program (maximum)	No extra current during write (included in active power specification)	5 mA
Supply current during erase (maximum)	Not applicable (pre-erase not required)	7 mA
Nonvolatile memory maximum read frequency	8 MHz	16 MHz

MSP430 FRAM的优势

数据更新 （寿命长 10^15方）

程序更新（读写速度快）

掉电丢失保存？（电压低、读写速度快）

MSP430 FRAM 的注意事项

★擦写次数特别多 10^15

★容易赋值操作，像RAM？

如何防止“非蓄意”的数据 or 程序更改？

★ 掉电不丢失。

MSP430 FRAM 的注意事项

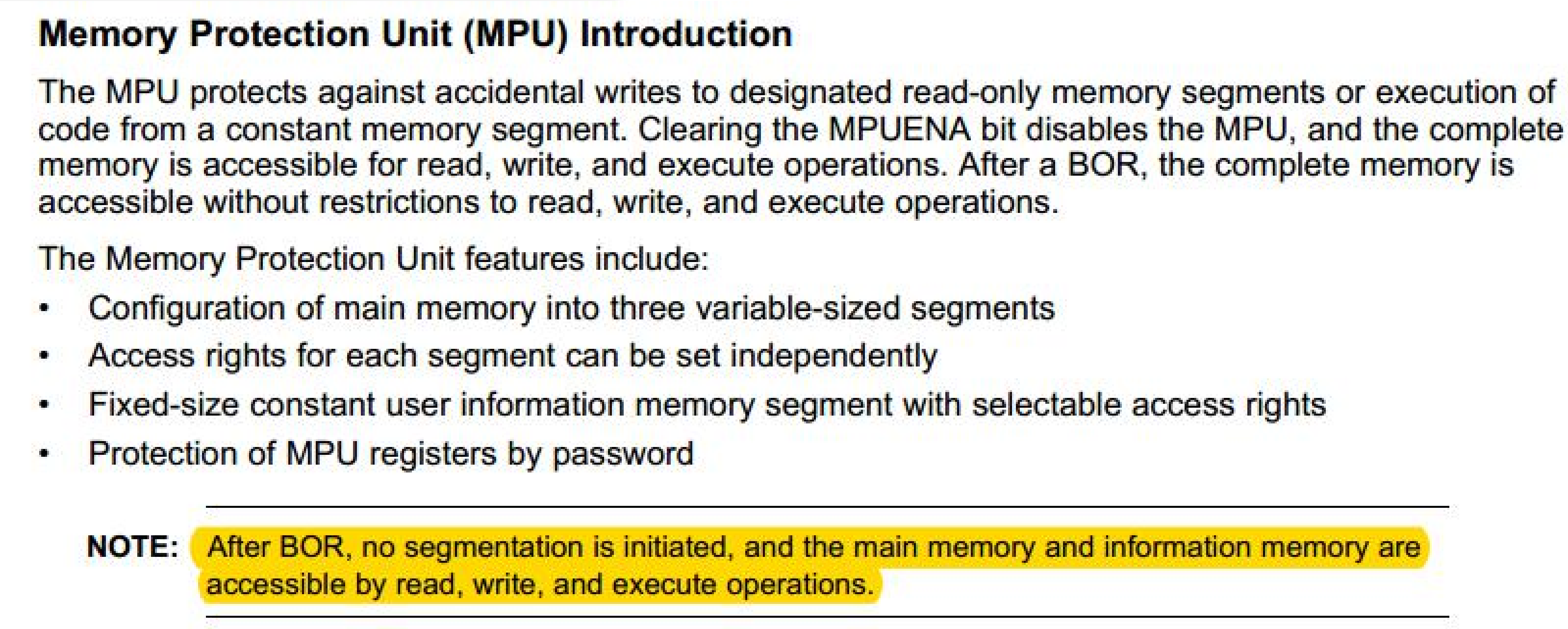
MSP430FRxx

- 带MPU模块， 如FR57xx,FR69xx
- 不带MPU模块,如FR2xx、FR4xx

MSP430 FRAM 的注意事项

MPU = Memory Protect Unit

1> MPU 用还是不用？



Memory Protection Unit (MPU) Introduction

The MPU protects against accidental writes to designated read-only memory segments or execution of code from a constant memory segment. Clearing the MPUENA bit disables the MPU, and the complete memory is accessible for read, write, and execute operations. After a BOR, the complete memory is accessible without restrictions to read, write, and execute operations.

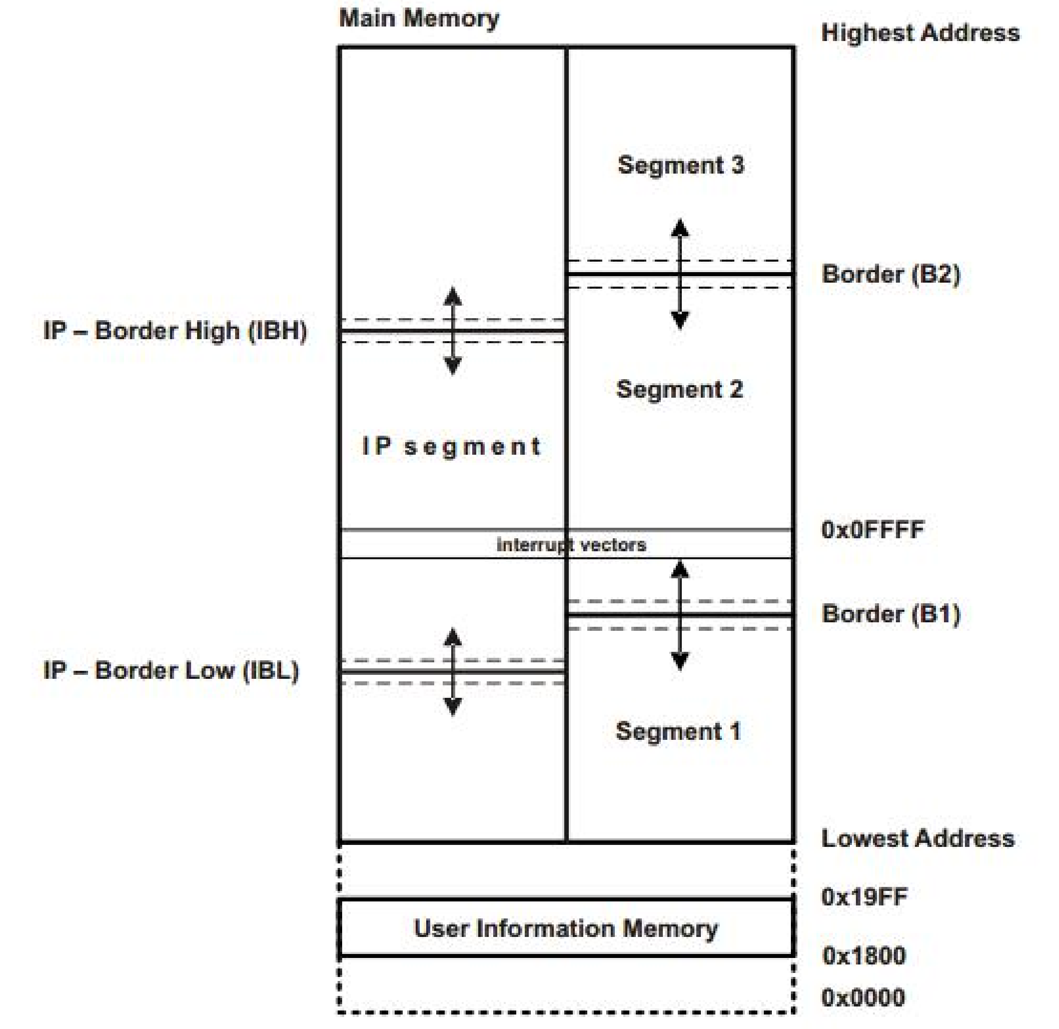
The Memory Protection Unit features include:

- Configuration of main memory into three variable-sized segments
- Access rights for each segment can be set independently
- Fixed-size constant user information memory segment with selectable access rights
- Protection of MPU registers by password

NOTE: After BOR, no segmentation is initiated, and the main memory and information memory are accessible by read, write, and execute operations.

MSP430 FRAM 的注意事项

2> MPU 怎么用？



MSP430 FRAM 的注意事项

```
void Mpu_Setting(void)
{

   //MPU模块使能，并且密码要正确
   MPUCTL0 = MPUPW;
   MPUSEGB1 = 0x0FF0;
   MPUSEGB2 = 0x1000;
   MPUSAM = MPUSEGIRE  \
         |MPUSEG3RE   \
         |MPUSEG2WE|MPUSEG2RE   \
         |MPUSEG1XE|MPUSEG1RE ;


    //MPU模块使能，并且密码要正确
   MPUCTL0 = MPUPW | MPUENA;
   //故意在MPUCTL0的高字节写错，禁止MPU寄存器的再次写入
   MPUCTL0_H = 0xDD;
}
```

MSP430 FRAM 的注意事项

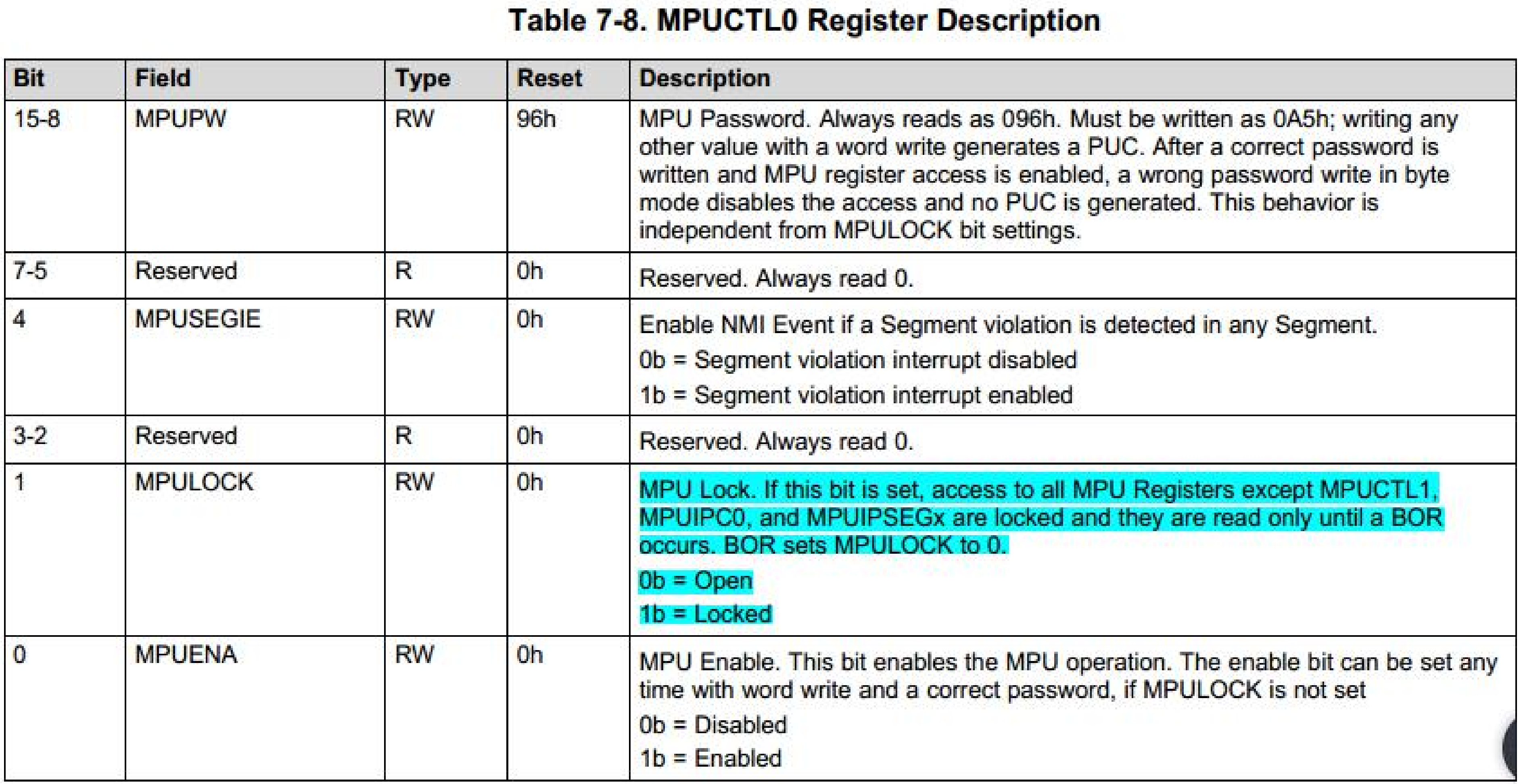
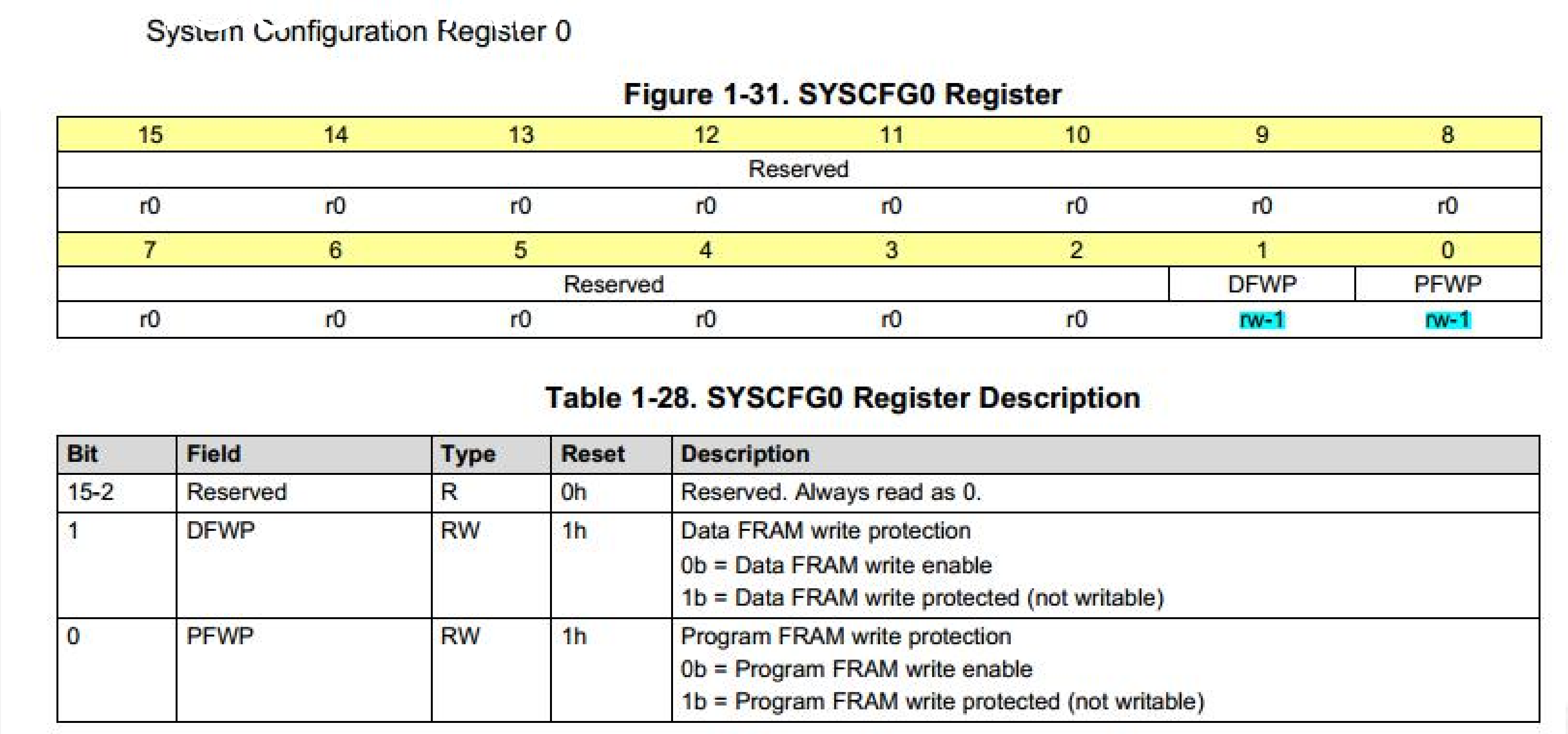


Table 7-8. MPUCTL0 Register Description

Bit	Field	Type	Reset	Description
15-8	MPUPW	RW	96h	MPU Password. Always reads as 096h. Must be written as 0A5h; writing any other value with a word write generates a PUC. After a correct password is written and MPU register access is enabled, a wrong password write in byte mode disables the access and no PUC is generated. This behavior is independent from MPULOCK bit settings.
7-5	Reserved	R	0h	Reserved. Always read 0.
4	MPUSEGIE	RW	0h	Enable NMI Event if a Segment violation is detected in any Segment. 0b = Segment violation interrupt disabled 1b = Segment violation interrupt enabled
3-2	Reserved	R	0h	Reserved. Always read 0.
1	MPULOCK	RW	0h	MPU Lock. If this bit is set, access to all MPU Registers except MPUCTL1, MPUIPC0, and MPUIPSEGx are locked and they are read only until a BOR occurs. BOR sets MPULOCK to 0. 0b = Open 1b = Locked
0	MPUENA	RW	0h	MPU Enable. This bit enables the MPU operation. The enable bit can be set any time with word write and a correct password, if MPULOCK is not set 0b = Disabled 1b = Enabled

MSP430 FRAM 的注意事项



System Configuration Register 0

Figure 1-31. SYSCFG0 Register

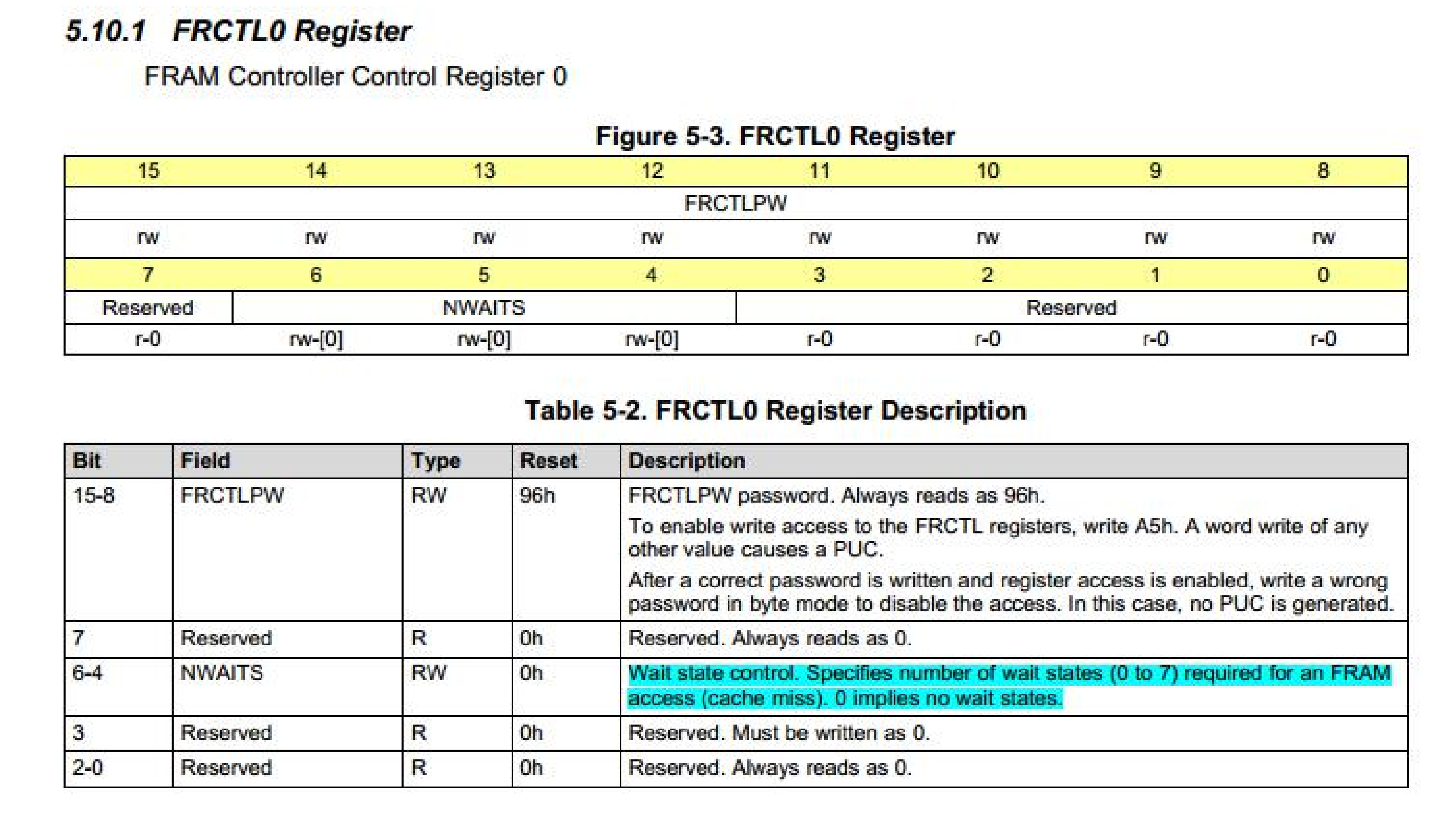
15	14	13	12	11	10	9	8
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
Reserved						DFWP	PFWP
r0	r0	r0	r0	r0	r0	rw-1	rw-1

Table 1-28. SYSCFG0 Register Description

Bit	Field	Type	Reset	Description
15-2	Reserved	R	0h	Reserved. Always read as 0.
1	DFWP	RW	1h	Data FRAM write protection 0b = Data FRAM write enable 1b = Data FRAM write protected (not writable)
0	PFWP	RW	1h	Program FRAM write protection 0b = Program FRAM write enable 1b = Program FRAM write protected (not writable)

```
void Fram_Erase(void )
{
  u16_t* ErasePtr = (u16_t *) (0xF000);
 *ErasePtr = 0xFFFF;
}
```

MSP430 FRAM 的注意事项



5.10.1 *FRCTL0 Register*

FRAM Controller Control Register 0

Figure 5-3. FRCTL0 Register

15	14	13	12	11	10	9	8
FRCTLPW							
rw	rw	rw	rw	rw	rw	rw	rw
7	6	5	4	3	2	1	0
Reserved	NWAITS			Reserved			
r-0	rw-[0]	rw-[0]	rw-[0]	r-0	r-0	r-0	r-0

Table 5-2. FRCTL0 Register Description

Bit	Field	Type	Reset	Description
15-8	FRCTLPW	RW	96h	FRCTLPW password. Always reads as 96h. To enable write access to the FRCTL registers, write A5h. A word write of any other value causes a PUC. After a correct password is written and register access is enabled, write a wrong password in byte mode to disable the access. In this case, no PUC is generated.
7	Reserved	R	0h	Reserved. Always reads as 0.
6-4	NWAITS	RW	0h	Wait state control. Specifies number of wait states (0 to 7) required for an FRAM access (cache miss). 0 implies no wait states.
3	Reserved	R	0h	Reserved. Must be written as 0.
2-0	Reserved	R	0h	Reserved. Always reads as 0.

如何灵活分配FRAM在数据存储和程序存储的应用

★擦写次数特别多 10^15

★容易赋值操作，像RAM？

怎么当RAM用？

★ 掉电不丢失。

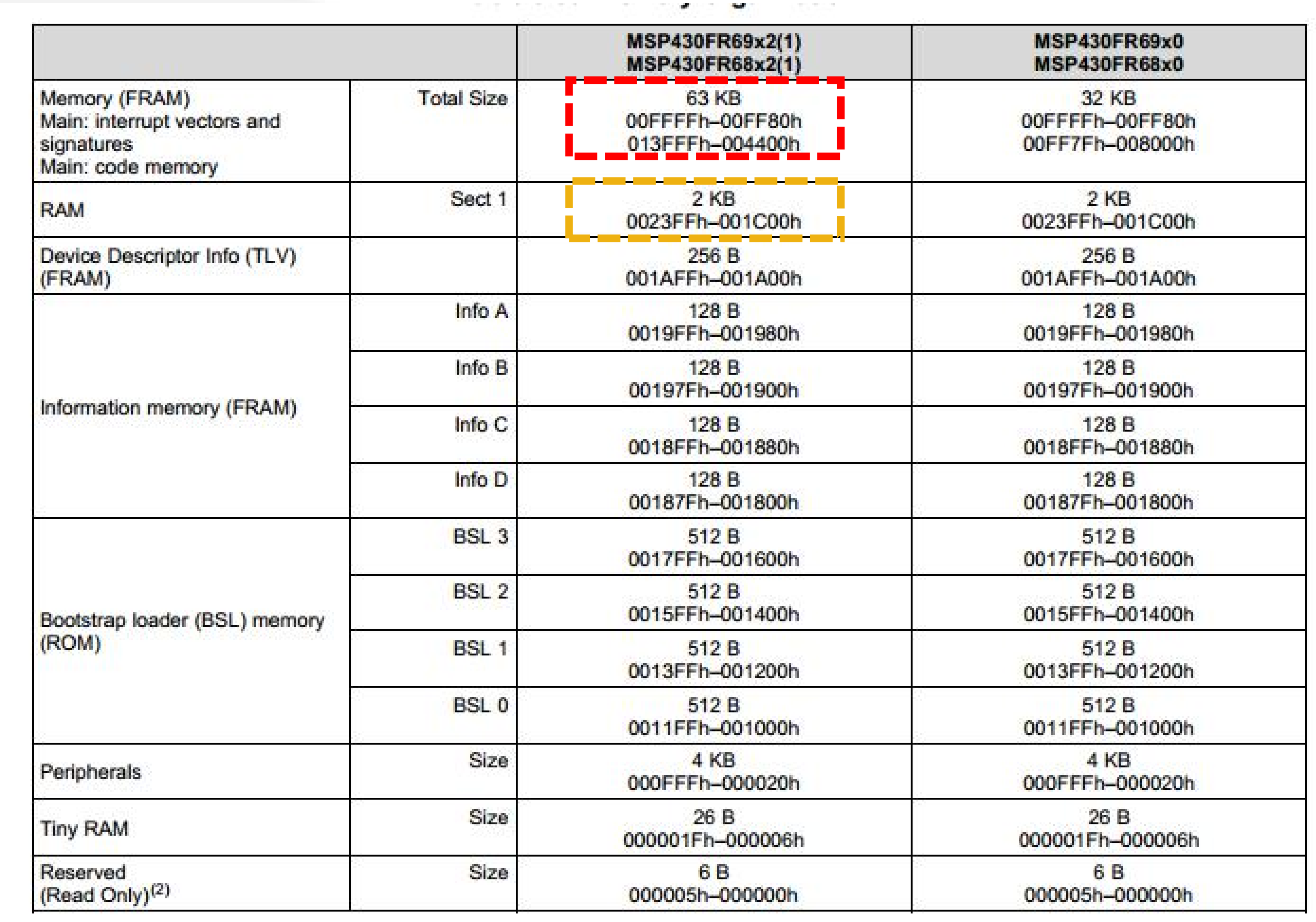
如何灵活分配FRAM在数据存储和程序存储的应用

MSP430FRxx

- 带MPU模块，如FR57xx,FR69xx
- 不带MPU模块,如FR2xx、FR4xx

如何灵活分配FRAM在数据存储和程序存储的应用

带MPU的，以FR6972为例



		MSP430FR69x2(1)
		MSP430FR68x2(1)
			MSP430FR69x0
			MSP430FR68x0

Memory (FRAM) Main: interrupt vectors and signatures Main: code memory	Total Size	63 KB 00FFFFh–00FF80h 013FFFh–004400h	32 KB 00FFFFh–00FF80h 00FF7Fh–008000h
RAM	Sect 1	2 KB 0023FFh–001C00h	2 KB 0023FFh–001C00h
Device Descriptor Info (TLV) (FRAM)		256 B 001AFFh–001A00h	256 B 001AFFh–001A00h
Information memory (FRAM)	Info A	128 B 0019FFh–001980h	128 B 0019FFh–001980h
	Info B	128 B 00197Fh–001900h	128 B 00197Fh–001900h
	Info C	128 B 0018FFh–001880h	128 B 0018FFh–001880h
	Info D	128 B 00187Fh–001800h	128 B 00187Fh–001800h
Bootstrap loader (BSL) memory (ROM)	BSL 3	512 B 0017FFh–001600h	512 B 0017FFh–001600h
	BSL 2	512 B 0015FFh–001400h	512 B 0015FFh–001400h
	BSL 1	512 B 0013FFh–001200h	512 B 0013FFh–001200h
	BSL 0	512 B 0011FFh–001000h	512 B 0011FFh–001000h
Peripherals	Size	4 KB 000FFFh–000020h	4 KB 000FFFh–000020h
Tiny RAM	Size	26 B 000001Fh–000006h	26 B 000001Fh–000006h
Reserved (Read Only)(2)	Size	6 B 000005h–000000h	6 B 000005h–000000h

如何灵活分配FRAM在数据存储和程序存储的应用

.xcl 更改前

```
// -----------------------------------------------
// RAM memory
//
-Z(DATA)TINYRAM=0006-001F
-Z(DATA)DATA16_I,DATA16_Z,DATA16_N,TLS16_I=1C00-23FF
-Z(DATA)CODE_I
-Z(DATA)DATA20_I,DATA20_Z,DATA20_N
-Z(DATA)CSTACK+_STACK_SIZE#
```

.xcl 更改后

```
// -----------------------------------------------
// RAM memory
-Z(DATA)TINYRAM=0006-001F
-Z(DATA)DATA16_I,DATA16_Z,DATA16_N,TLS16_I=1C00-23FF， 4400-47FF
-Z(DATA)CODE_I
-Z(DATA)DATA20_I,DATA20_Z,DATA20_N
-Z(DATA)CSTACK+_STACK_SIZE#
```

如何灵活分配FRAM在数据存储和程序存储的应用

```
// --------------------------
// Constant data
//
-Z(CONST)DATA16_C,DATA16_ID,TLS16_ID,DIFUNCT,CHECKSUM=4400-FF7F
// ---------------------------
// Code                                                      4800-FF7F
//
-Z(CODE)CSTART,ISR_CODE,CODE16= 4400-FF7F
                                         4800-FF7F
// ------------------------------------
// All memory 0-FFFFF
//
// ---------------------------
// Code
//                        4800-FF7F
-P(CODE)CODE=4400-FF7F,10000-13FFF
-Z(CODE)CODE_PAD
// ---------------------------
// Constant data
//
                                          4800-FF7F
-Z(CONST)DATA20_C,DATA20_ID,CODE_ID=4400-FF7F,10040-13FFF
```

如何灵活分配FRAM在数据存储和程序存储的应用

```
void Mpu_Setting(void)
{

   //MPU模块使能，并且密码要正确
   MPUCTL0 = MPUPW;
   MPUSEGB1 = 0x0480;
   MPUSEGB2 = 0x1000;
   MPUSAM |=
         MPUSEG1XE | MPUSEG1RE  | MPUSEG1WE
         | ……|……;


    //MPU模块使能，并且密码要正确
   MPUCTL0 = MPUPW | MPUENA;
   //故意在MPUCTL0的高字节写错，禁止MPU寄存器的再次写入
   MPUCTL0_H = 0xDD;
}
```

如何灵活分配FRAM在数据存储和程序存储的应用

不带MPU的，以FR4133为例

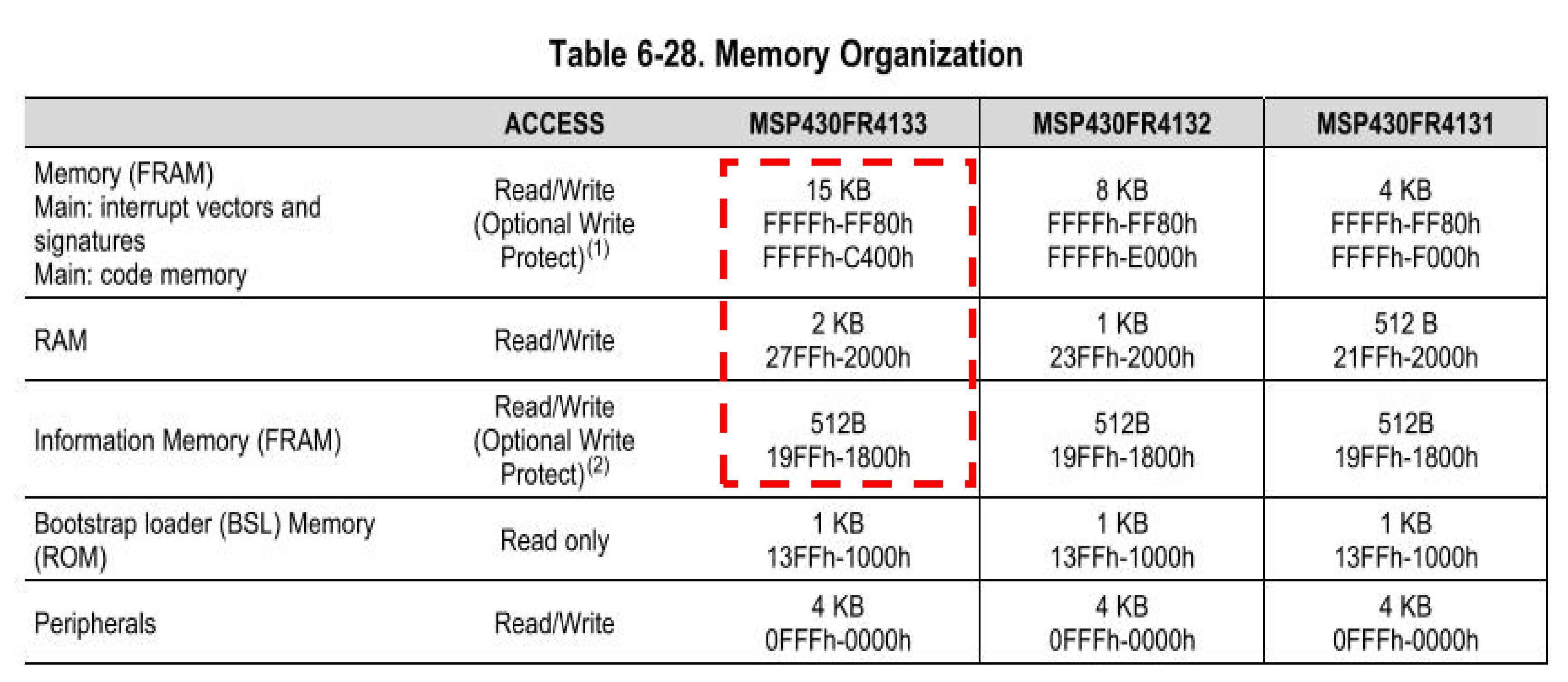


Table 6-28. Memory Organization

	ACCESS	MSP430FR4133	MSP430FR4132	MSP430FR4131
Memory (FRAM) Main: interrupt vectors and signatures Main: code memory	Read/Write (Optional Write Protect)[1]	15 KB FFFFh-FF80h FFFFh-C400h	8 KB FFFFh-FF80h FFFFh-E000h	4 KB FFFFh-FF80h FFFFh-F000h
RAM	Read/Write	2 KB 27FFh-2000h	1 KB 23FFh-2000h	512 B 21FFh-2000h
Information Memory (FRAM)	Read/Write (Optional Write Protect)[2]	512B 19FFh-1800h	512B 19FFh-1800h	512B 19FFh-1800h
Bootstrap loader (BSL) Memory (ROM)	Read only	1 KB 13FFh-1000h	1 KB 13FFh-1000h	1 KB 13FFh-1000h
Peripherals	Read/Write	4 KB 0FFFh-0000h	4 KB 0FFFh-0000h	4 KB 0FFFh-0000h

据存储和程

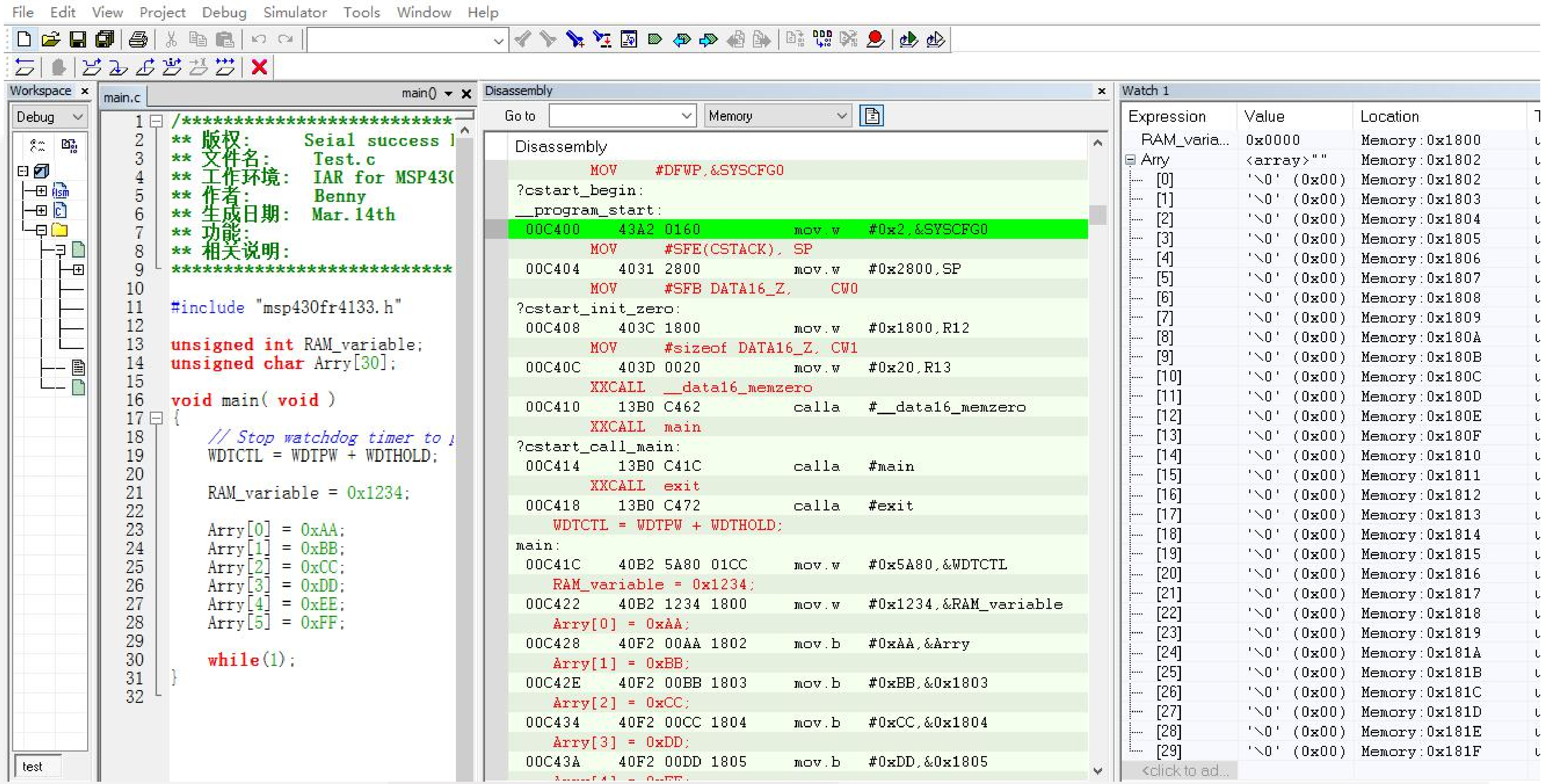
.xcl 更改前

```
// ----------------------------------------------
// RAM memory
-Z(DATA)DATA16_I,DATA16_Z,DATA16_N,TLS16_I=2000-27FF
-Z(DATA)CODE_I
-Z(DATA)DATA20_I,DATA20_Z,DATA20_N
-Z(DATA)CSTACK+_STACK_SIZE#
```

.xcl 更改后

```
// ----------------------------------------------
// RAM memory
-Z(DATA)DATA16_I,DATA16_Z,DATA16_N,TLS16_I=1800-19FF,2000-27FF
-Z(DATA)CODE_I
-Z(DATA)DATA20_I,DATA20_Z,DATA20_N
-Z(DATA)CSTACK+_STACK_SIZE#
```

如何灵活分配FRAM在数据存储和程序存储的应用

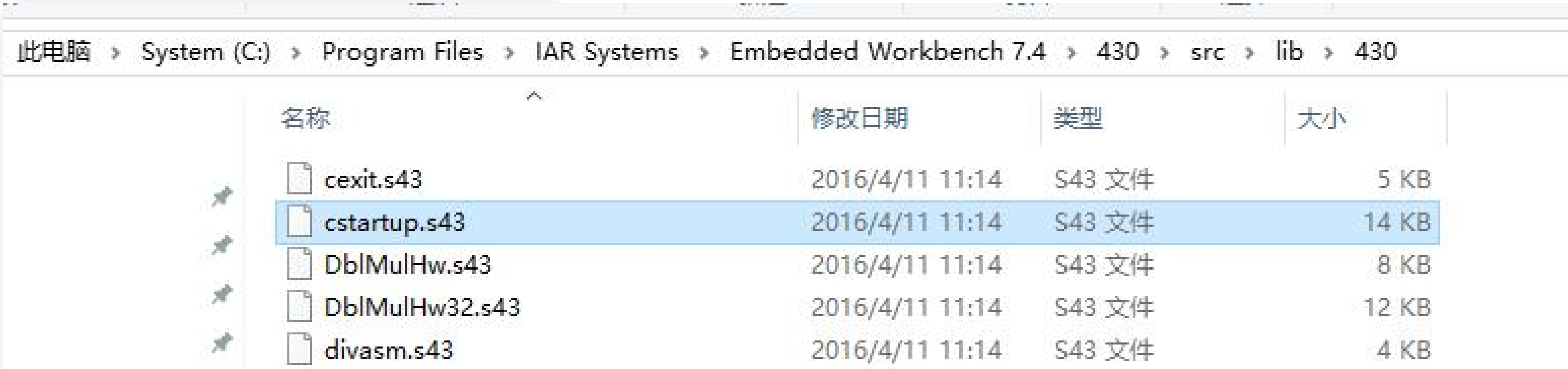


```
/*****************************
** 版权:      Seial success l
** 文件名:     Test.c
** 工作环境:   IAR for MSP43(
** 作者:       Benny
** 生成日期:   Mar.14th
** 功能:
** 相关说明:
*****************************

#include "msp430fr4133.h"

unsigned int RAM_variable;
unsigned char Arry[30];

void main( void )
{
    // Stop watchdog timer to p
    WDTCTL = WDTPW + WDTHOLD;

    RAM_variable = 0x1234;

    Arry[0] = 0xAA;
    Arry[1] = 0xBB;
    Arry[2] = 0xCC;
    Arry[3] = 0xDD;
    Arry[4] = 0xEE;
    Arry[5] = 0xFF;

    while(1);
}
```

```
Disassembly
        MOV     #DFWP,&SYSCFG0
?cstart_begin:
__program_start:
 00C400     43A2 0160           mov.w    #0x2,&SYSCFG0
        MOV     #SFE(CSTACK), SP
 00C404     4031 2800           mov.w    #0x2800,SP
        MOV     #SFB DATA16_Z,    CW0
?cstart_init_zero:
 00C408     403C 1800           mov.w    #0x1800,R12
        MOV     #sizeof DATA16_Z, CW1
 00C40C     403D 0020           mov.w    #0x20,R13
        XXCALL  __data16_memzero
 00C410     13B0 C462           calla    #__data16_memzero
        XXCALL  main
?cstart_call_main:
 00C414     13B0 C41C           calla    #main
        XXCALL  exit
 00C418     13B0 C472           calla    #exit
    WDTCTL = WDTPW + WDTHOLD;
main:
 00C41C     40B2 5A80 01CC      mov.w    #0x5A80,&WDTCTL
    RAM_variable = 0x1234;
 00C422     40B2 1234 1800      mov.w    #0x1234,&RAM_variable
    Arry[0] = 0xAA;
 00C428     40F2 00AA 1802      mov.b    #0xAA,&Arry
    Arry[1] = 0xBB;
 00C42E     40F2 00BB 1803      mov.b    #0xBB,&0x1803
    Arry[2] = 0xCC;
 00C434     40F2 00CC 1804      mov.b    #0xCC,&0x1804
    Arry[3] = 0xDD;
 00C43A     40F2 00DD 1805      mov.b    #0xDD,&0x1805
```

Expression	Value	Location
RAM_varia...	0x0000	Memory:0x1800
⊟ Arry	<array>""	Memory:0x1802
[0]	'\0' (0x00)	Memory:0x1802
[1]	'\0' (0x00)	Memory:0x1803
[2]	'\0' (0x00)	Memory:0x1804
[3]	'\0' (0x00)	Memory:0x1805
[4]	'\0' (0x00)	Memory:0x1806
[5]	'\0' (0x00)	Memory:0x1807
[6]	'\0' (0x00)	Memory:0x1808
[7]	'\0' (0x00)	Memory:0x1809
[8]	'\0' (0x00)	Memory:0x180A
[9]	'\0' (0x00)	Memory:0x180B
[10]	'\0' (0x00)	Memory:0x180C
[11]	'\0' (0x00)	Memory:0x180D
[12]	'\0' (0x00)	Memory:0x180E
[13]	'\0' (0x00)	Memory:0x180F
[14]	'\0' (0x00)	Memory:0x1810
[15]	'\0' (0x00)	Memory:0x1811
[16]	'\0' (0x00)	Memory:0x1812
[17]	'\0' (0x00)	Memory:0x1813
[18]	'\0' (0x00)	Memory:0x1814
[19]	'\0' (0x00)	Memory:0x1815
[20]	'\0' (0x00)	Memory:0x1816
[21]	'\0' (0x00)	Memory:0x1817
[22]	'\0' (0x00)	Memory:0x1818
[23]	'\0' (0x00)	Memory:0x1819
[24]	'\0' (0x00)	Memory:0x181A
[25]	'\0' (0x00)	Memory:0x181B
[26]	'\0' (0x00)	Memory:0x181C
[27]	'\0' (0x00)	Memory:0x181D
[28]	'\0' (0x00)	Memory:0x181E
[29]	'\0' (0x00)	Memory:0x181F
<click to ad...		

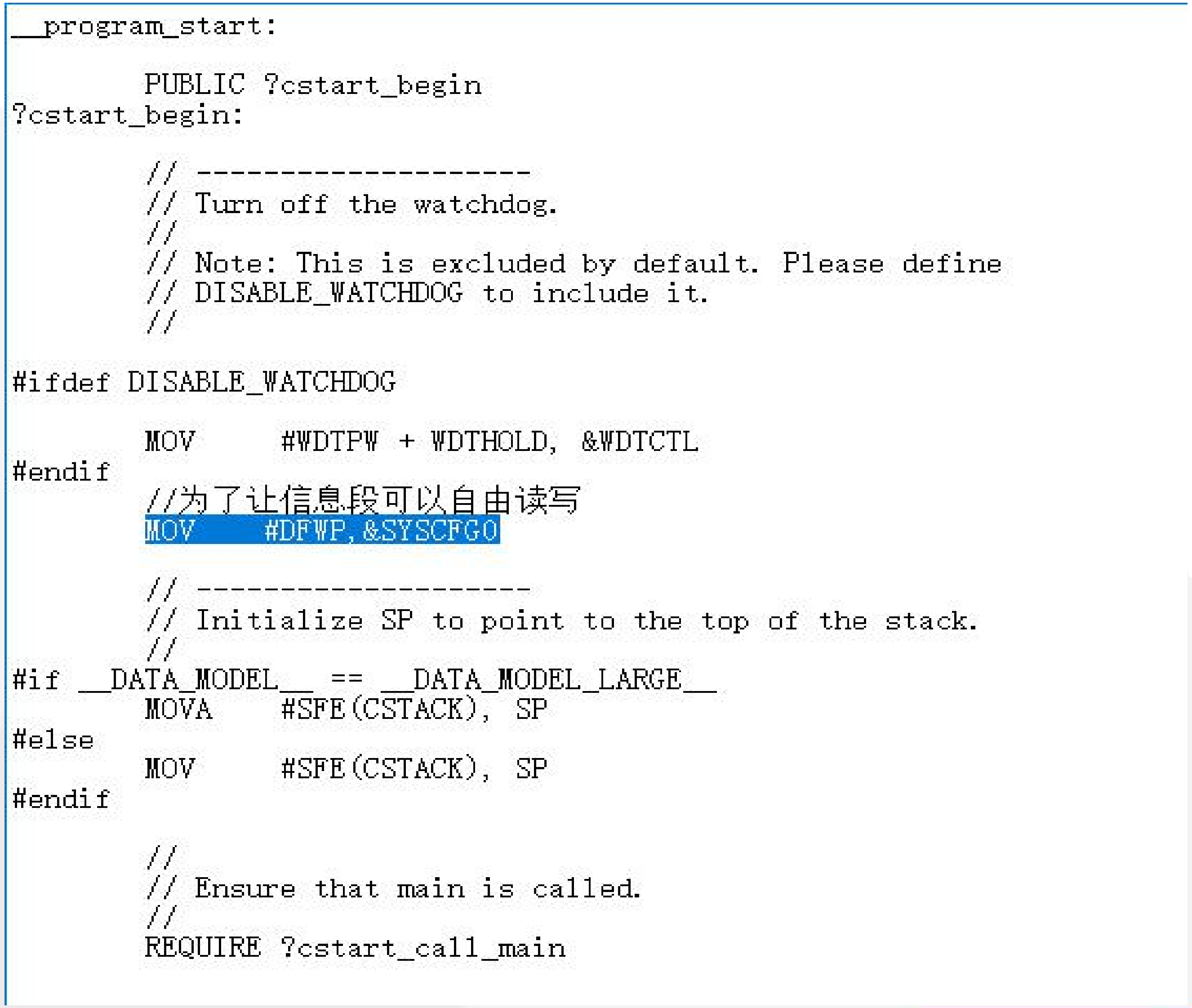
如何灵活分配FRAM在数据存储和程序存储的应用



此电脑 › System (C:) › Program Files › IAR Systems › Embedded Workbench 7.4 › 430 › src › lib › 430

名称	修改日期	类型	大小
cexit.s43	2016/4/11 11:14	S43 文件	5 KB
cstartup.s43	2016/4/11 11:14	S43 文件	14 KB
DblMulHw.s43	2016/4/11 11:14	S43 文件	8 KB
DblMulHw32.s43	2016/4/11 11:14	S43 文件	12 KB
divasm.s43	2016/4/11 11:14	S43 文件	4 KB

如何灵活分配FRAM在数据存储和程序存储的应用

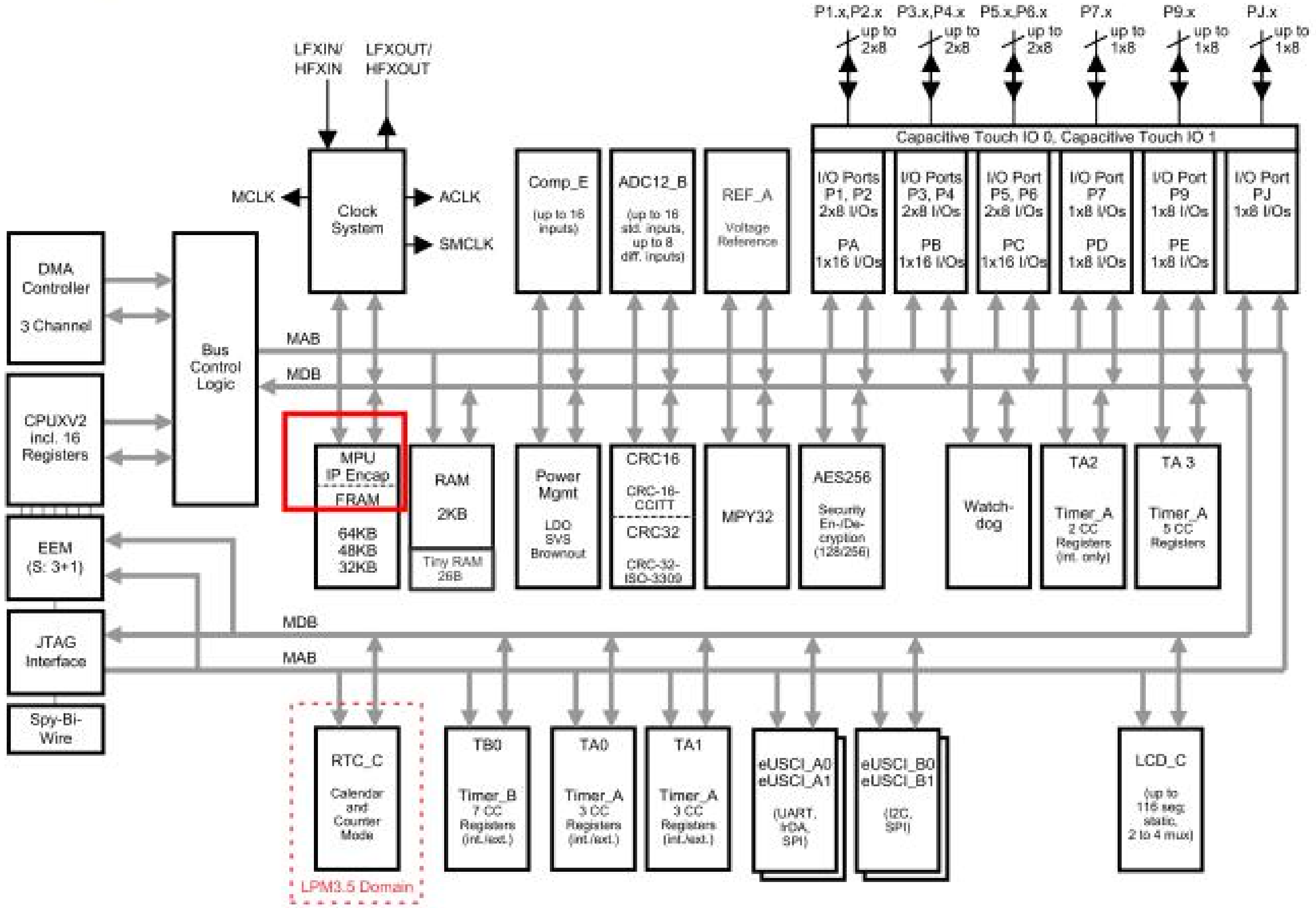


```
__program_start:

        PUBLIC ?cstart_begin
?cstart_begin:

        // --------------------
        // Turn off the watchdog.
        //
        // Note: This is excluded by default. Please define
        // DISABLE_WATCHDOG to include it.
        //

#ifdef DISABLE_WATCHDOG

        MOV     #WDTPW + WDTHOLD, &WDTCTL
#endif
        //为了让信息段可以自由读写
        MOV     #DFWP,&SYSCFG0

        // --------------------
        // Initialize SP to point to the top of the stack.
        //
#if __DATA_MODEL__ == __DATA_MODEL_LARGE__
        MOVA    #SFE(CSTACK), SP
#else
        MOV     #SFE(CSTACK), SP
#endif

        //
        // Ensure that main is called.
        //
        REQUIRE ?cstart_call_main
```

IP Encapsulation

- MSP430FR IPE Module

—IPE Module introduce
—IPE application Lab


P1.x,P2.x up to 2x8
P3.x,P4.x up to 2x8
P5.x,P6.x up to 2x8
P7.x up to 1x8
P9.x up to 1x8
PJ.x up to 1x8
LFXIN/ HFXIN
LFXOUT/ HFXOUT
MCLK
Clock System
ACLK
SMCLK
Comp_E (up to 16 inputs)
ADC12_B (up to 16 std. inputs, up to 8 diff. inputs)
REF_A Voltage Reference
Capacitive Touch IO 0, Capacitive Touch IO 1
I/O Ports P1, P2 2x8 I/Os PA 1x16 I/Os
I/O Ports P3, P4 2x8 I/Os PB 1x16 I/Os
I/O Port P5, P6 2x8 I/Os PC 1x16 I/Os
I/O Port P7 1x8 I/Os PD 1x8 I/Os
I/O Port P9 1x8 I/Os PE 1x8 I/Os
I/O Port PJ 1x8 I/Os
DMA Controller 3 Channel
Bus Control Logic
MAB
MDB
CPUXV2 incl. 16 Registers
MPU IP Encap
FRAM 64KB 48KB 32KB
RAM 2KB
Tiny RAM 26B
Power Mgmt LDO SVS Brownout
CRC16 CRC-16-CCITT
CRC32 CRC-32-ISO-3309
MPY32
AES256 Security En-/De-cryption (128/256)
Watch-dog
TA2 Timer_A 2 CC Registers (int. only)
TA 3 Timer_A 5 CC Registers
EEM (S: 3+1)
JTAG Interface
Spy-Bi-Wire
MDB
MAB
RTC_C Calendar and Counter Mode
LPM3.5 Domain
TB0 Timer_B 7 CC Registers (int./ext.)
TA0 Timer_A 3 CC Registers (int./ext.)
TA1 Timer_A 3 CC Registers (int./ext.)
eUSCI_A0 eUSCI_A1 (UART, IrDA, SPI)
eUSCI_B0 eUSCI_B1 (I2C, SPI)
LCD_C (up to 116 seg; static, 2 to 4 mux)

- What is IPE? —- intellectual property (IP) Encapsulation
- Which Devices have IPE module?—-FR58xx/FR59xx/FR69xx
- How to use IPE module?

  http://www.ti.com.cn/cn/lit/an/slaa685/slaa685.pdf

  (from Page9 to the last)

- What IPE can do ?

- IP Encapsulation allows the user to encapsulate and protect an area of the FRAM memory from readout

- When IPE is in place, any code or data in the IP Encapsulated area is protected from read or write access from anywhere outside of the IP Encapsulated area, even by JTAG

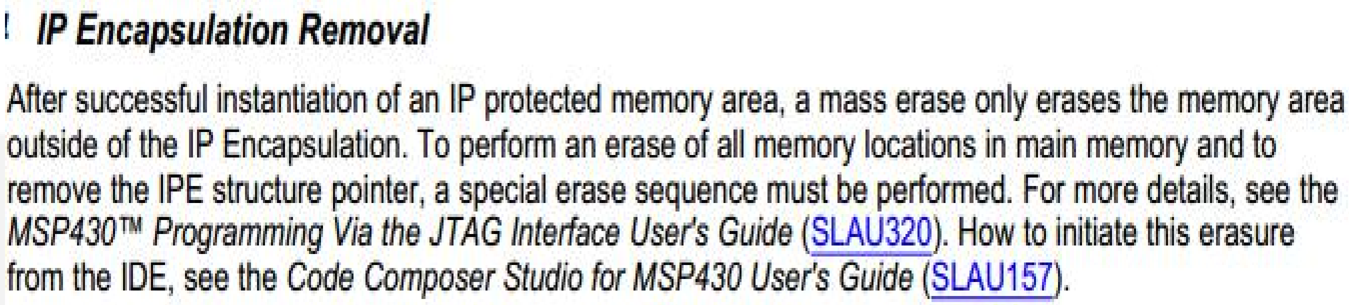
- No form of code protection is perfect, but this feature adds an additional layer on top of JTAG/SBW or bootloader security, for sensitive data like keys or for proprietary code that is the user's intellectual property (IP)

- The way to remove the IPE protection is to perform a special mass erase sequence enabled by the tool chain

被IPE保护的函数可以被整个工程的中的任意函数调用，但是IPE区域内的数据（常量）只能被IPE区域的函数读取。

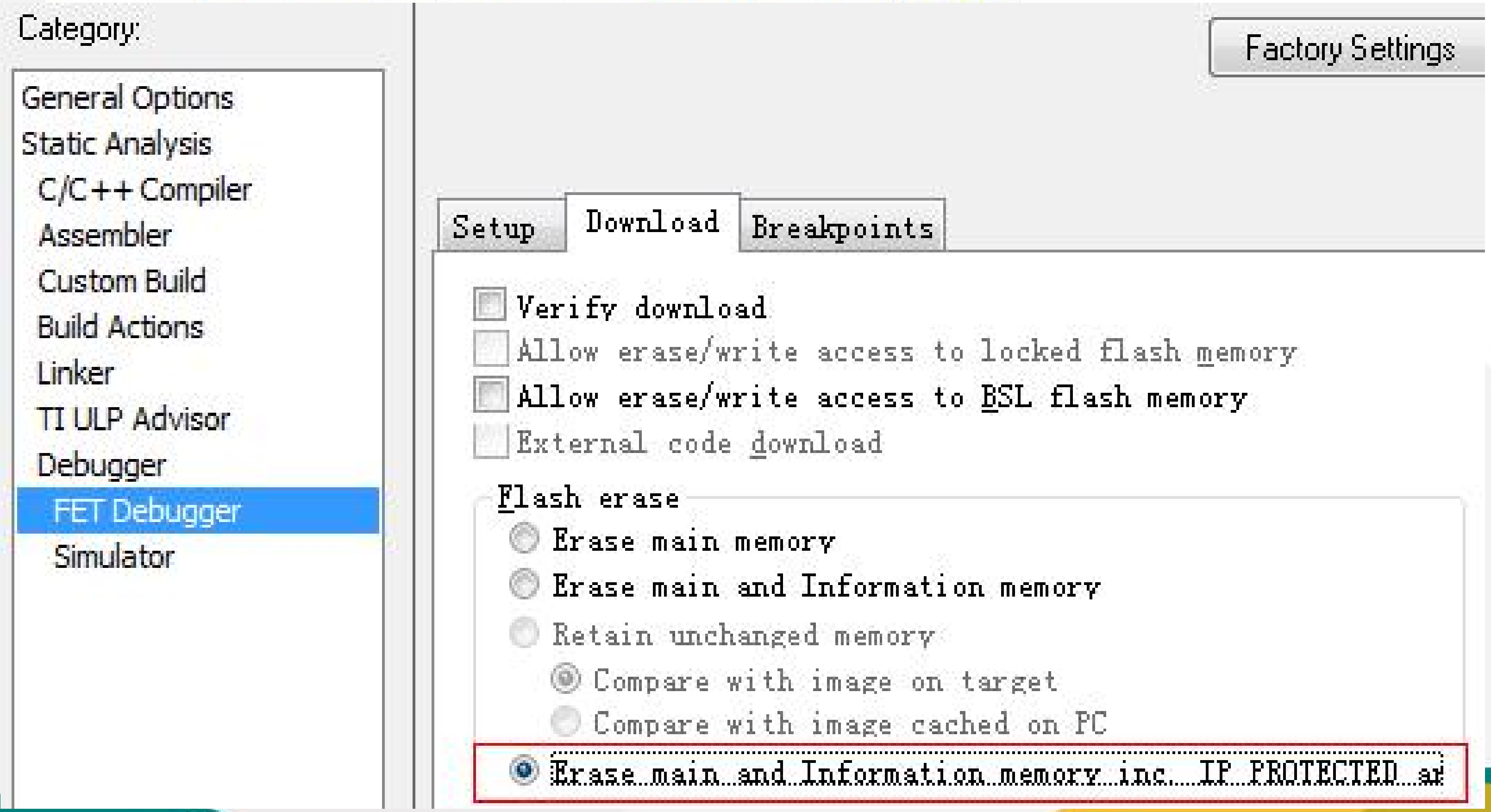
FRAM430上电后并不是马上执行用户代码，而是先执行一段固化在430内部的Boot Code，这段代码用于加载出厂校准信息并检测BSL加载时序，在这之后Boot Code先进行IPE配置以保护IPE区域的代码，最后才会执行用户代码。

IPE的主要特点



IP Encapsulation Removal

After successful instantiation of an IP protected memory area, a mass erase only erases the memory area outside of the IP Encapsulation. To perform an erase of all memory locations in main memory and to remove the IPE structure pointer, a special erase sequence must be performed. For more details, see the *MSP430™ Programming Via the JTAG Interface User's Guide* (SLAU320). How to initiate this erasure from the IDE, see the *Code Composer Studio for MSP430 User's Guide* (SLAU157).



Category:

- General Options
- Static Analysis
  - C/C++ Compiler
  - Assembler
  - Custom Build
  - Build Actions
  - Linker
  - TI ULP Advisor
  - Debugger
    - FET Debugger
    - Simulator

□ Verify download
□ Allow erase/write access to locked flash memory
□ Allow erase/write access to BSL flash memory
□ External code download

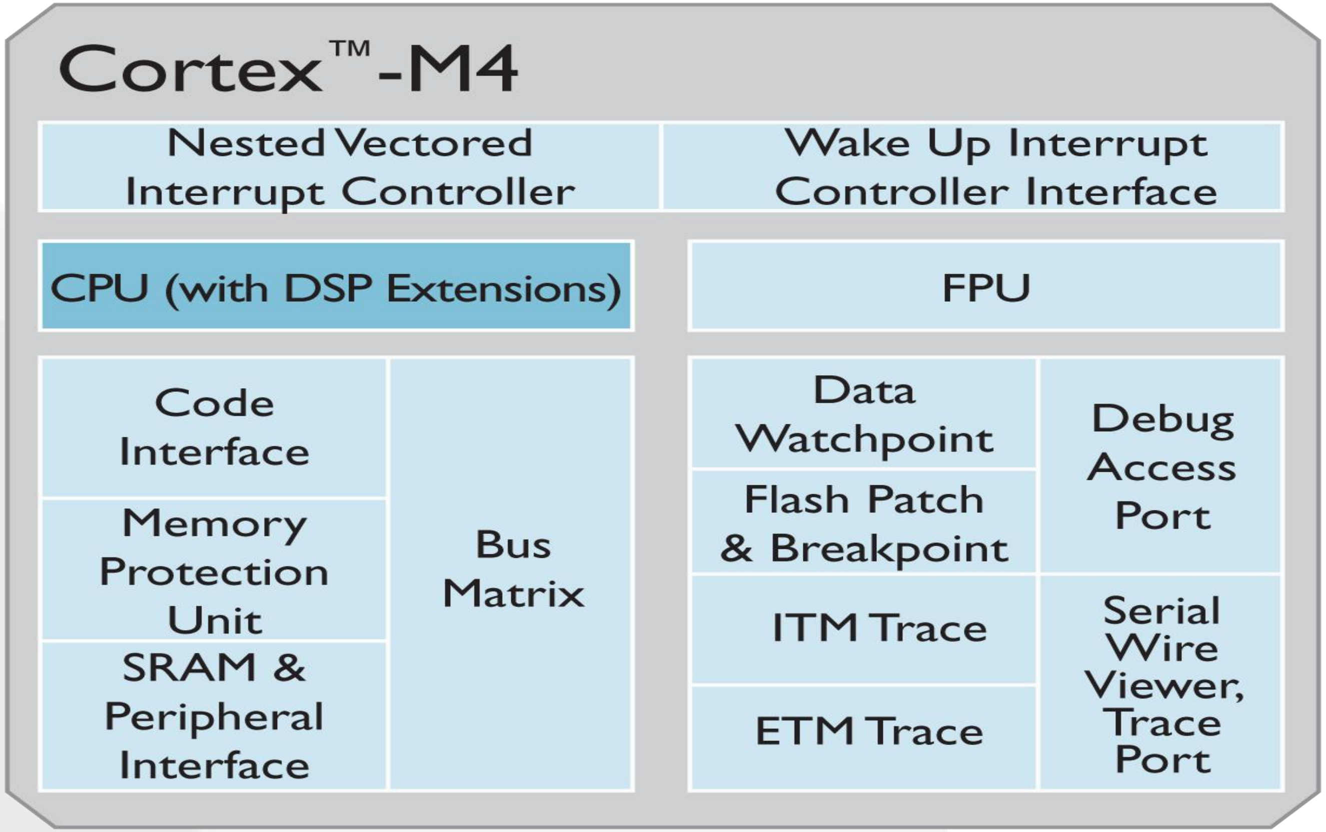
Flash erase
○ Erase main memory
○ Erase main and Information memory
○ Retain unchanged memory
● Compare with image on target
○ Compare with image cached on PC
● Erase main and Information memory inc. IP PROTECTED ar

MCUs Training

MSP432™ : Overview

MSP432 | 32-bit Cortex-M4F

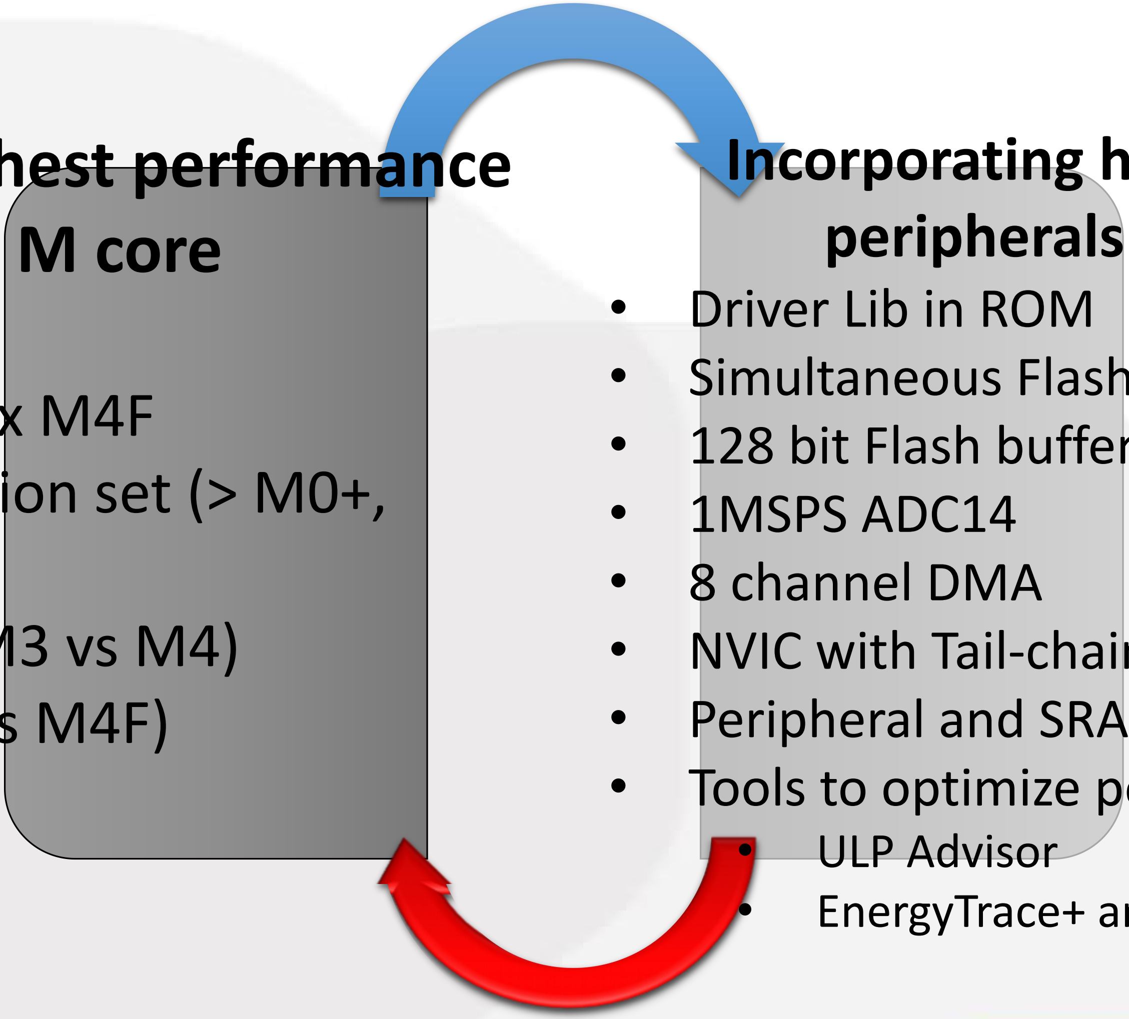
- 32-bit pipeline architecture
- Cortex-M4 with DSP extension instruction set
- Floating Point Unit
- Standard Cortex-M Debugger Module, Serial Wire Debug, ITM Trace support
- Core modules including DMA, SysTick, & Interrupt (NVIC)


Cortex™-M4
Nested Vectored Interrupt Controller
Wake Up Interrupt Controller Interface
CPU (with DSP Extensions)
FPU
Code Interface
Memory Protection Unit
SRAM & Peripheral Interface
Bus Matrix
Data Watchpoint
Flash Patch & Breakpoint
Debug Access Port
ITM Trace
ETM Trace
Serial Wire Viewer, Trace Port

Cortex-M | Core Comparison

Cortex-M	**Thumb**	**Thumb-2**	**HW MPY**	**HW DIV**	**Saturated math**	**DSP-extensions**	**FPU**	**ARM architecture**
Cortex-M0	Most	Subset	1 or 32 cycle	No	No	No	No	ARMv6-M Von Neumann
Cortex-M0+	Most	Subset	1 or 32 cycle	No	No	No	No	ARMv6-M Von Neumann
Cortex-M1	Most	Subset	3 or 33 cycle	No	No	No	No	ARMv6-M Von Neumann
Cortex-M3	Entire	Entire	1 cycle	2-12 cycles	Yes	No	No	ARMv7-M Harvard
Cortex-M4	Entire	Entire	1 cycle	2-12 cycles	Yes	Yes	Optional **Yes for MSP432**	ARMv7E-M Harvard

Increased processing capability



Selecting the highest performance Cortex M core

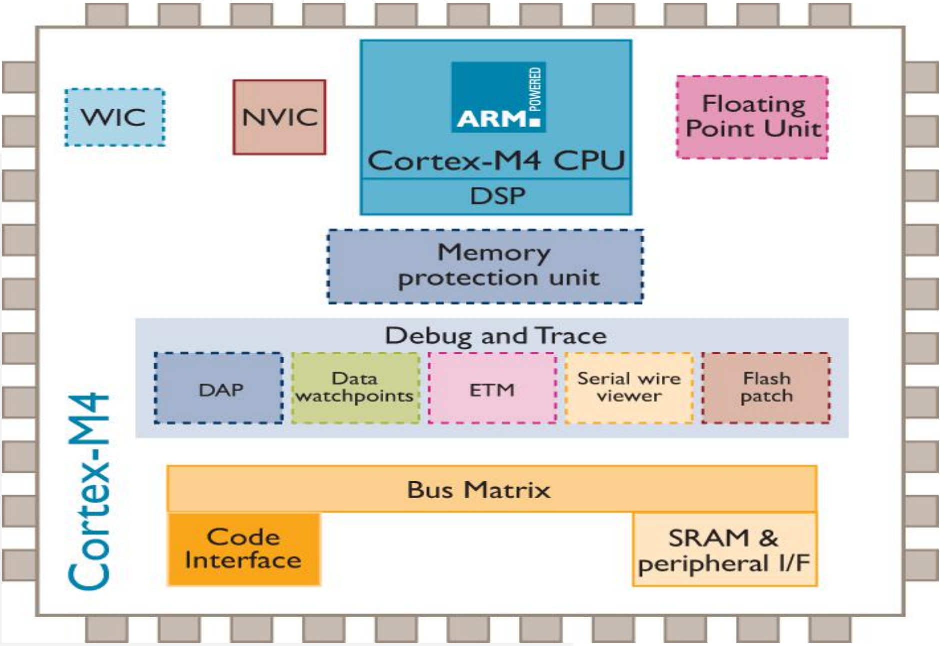
- 48MHz ARM Cotex M4F
- Full ARM instruction set (> M0+, M3,M4)
- DSP extensions (M3 vs M4)
- FPU engine (M4 vs M4F)

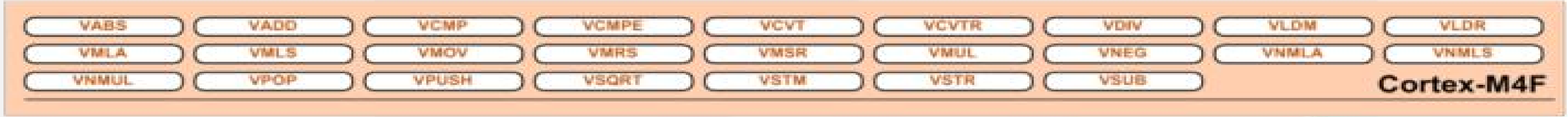
Incorporating high performance peripherals and features

- Driver Lib in ROM
- Simultaneous Flash read/write
- 128 bit Flash buffer and pre-fetch
- 1MSPS ADC14
- 8 channel DMA
- NVIC with Tail-chaining
- Peripheral and SRAM memory bit-band
- Tools to optimize power
  - ULP Advisor
  - EnergyTrace+ and Debuggers

FPU | Floating-Point Unit

- The FPU provides floating-point computation functionality that is compliant with the IEEE 754 standard
- Enables conversions between fixed-point and floa floating-point constant instructions
- The Cortex-M4F FPU fully supports single-precision:
  - Add
  - Subtract
  - Multiply
  - Divide
  - Single cycle multiply and accumulate (MAC)





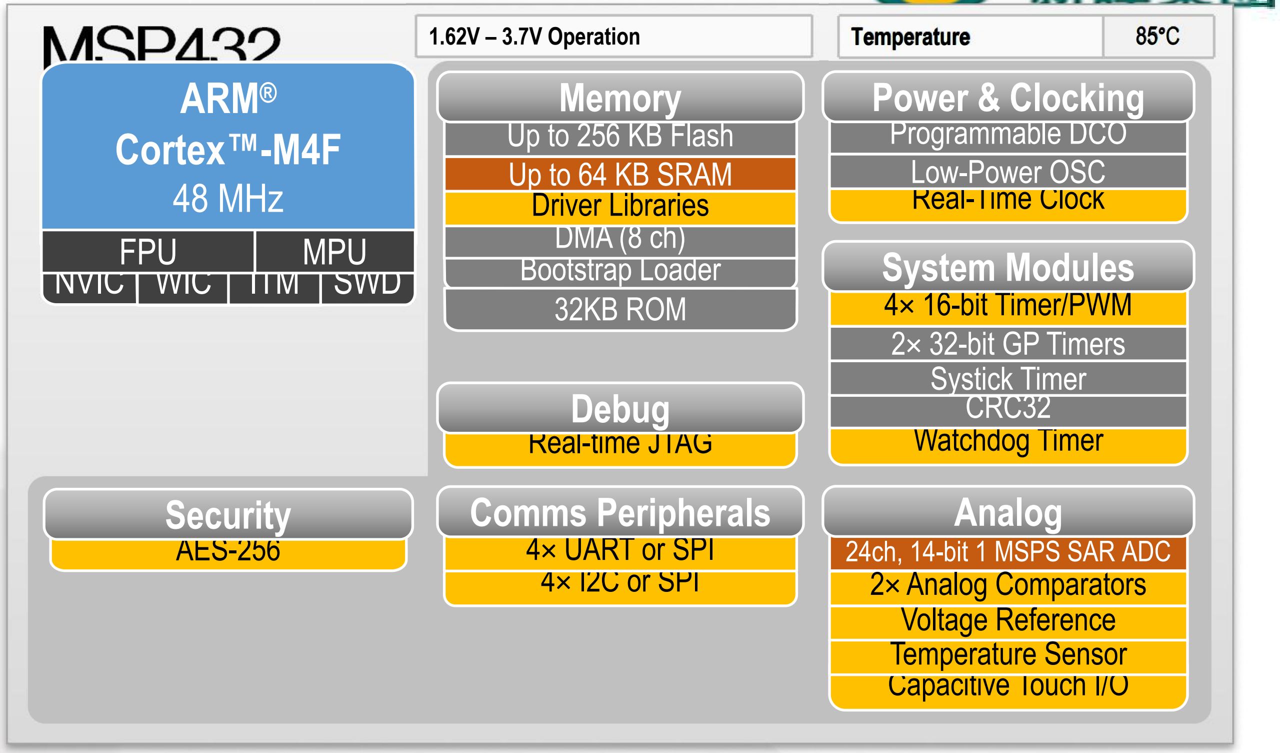
VABS	VADD	VCMP	VCMPE	VCVT	VCVTR	VDIV	VLDM	VLDR
VMLA	VMLS	VMOV	VMRS	VMSR	VMUL	VNEG	VNMLA	VNMLS
VNMUL	VPOP	VPUSH	VSQRT	VSTM	VSTR	VSUB		Cortex-M4F

Modes of Operation...

MSP432™ Microcontrollers

Differentiation

- **Ultra-low standby and active power, and fast wakeup** – 95uA/MHz active, 850nA Standby; Deep sleep to Active: <10us typ
- **Wide supply range** – 1.62-3.7V, including flash operation, enabling multiple battery technologies and eliminating external regulation
- **Integrated high-performance and low-power analog** – Including 1MSPS 14-bit ADC
- **Secure MCU environment** – Flash IP protection & integrated AES-256 encryption



MSP432

1.62V – 3.7V Operation

Temperature 85°C

ARM® Cortex™-M4F 48 MHz
FPU | MPU
NVIC | WIC | ITM | SWD

Memory
Up to 256 KB Flash
Up to 64 KB SRAM
Driver Libraries
DMA (8 ch)
Bootstrap Loader
32KB ROM

Power & Clocking
Programmable DCO
Low-Power OSC
Real-Time Clock

System Modules
4× 16-bit Timer/PWM
2× 32-bit GP Timers
Systick Timer
CRC32
Watchdog Timer

Debug
Real-time JTAG

Security
AES-256

Comms Peripherals
4× UART or SPI
4× I2C or SPI

Analog
24ch, 14-bit 1 MSPS SAR ADC
2× Analog Comparators
Voltage Reference
Temperature Sensor
Capacitive Touch I/O

🟨 Same as MSP430

Kits

LaunchPad

- Designed for evaluation and initial development
- Includes on-board emulator
- $12.99

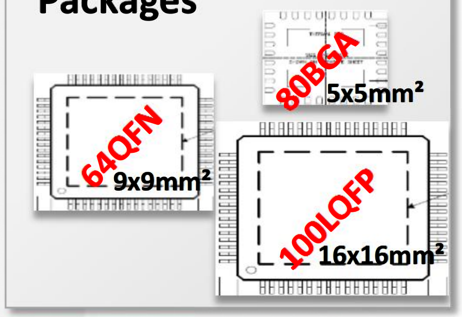
Target Board

- Designed for advanced development
- $89

Tools & Software

- **MSPWare –** leverage C-code portable MSP430 peripherals and analog
- **TI RTOS Support**
- **ARM 3rd Party Ecosystem**
- **Code Composer Studio™, IAR, KEIL IDEs, and gcc**

Packages



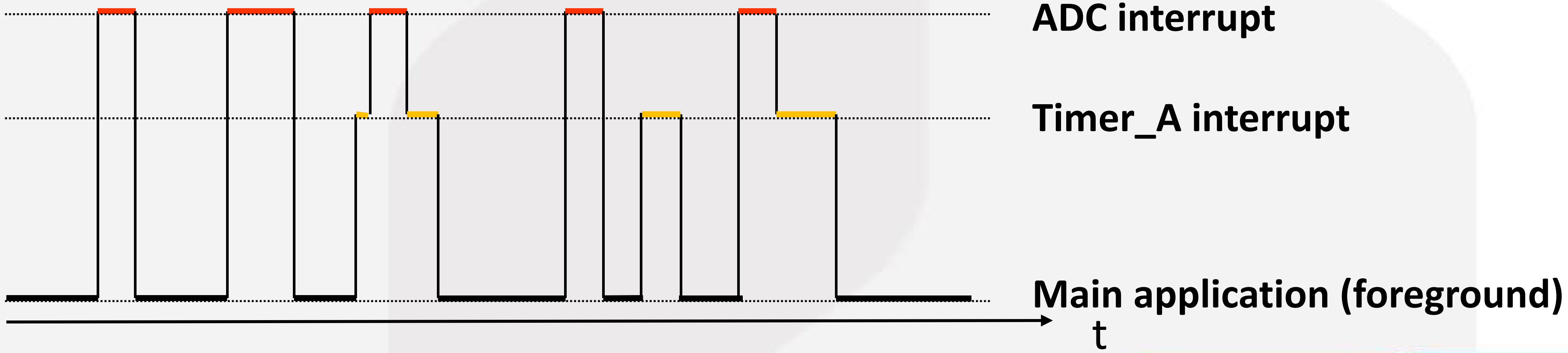
Execute and gain share

MCUs Training

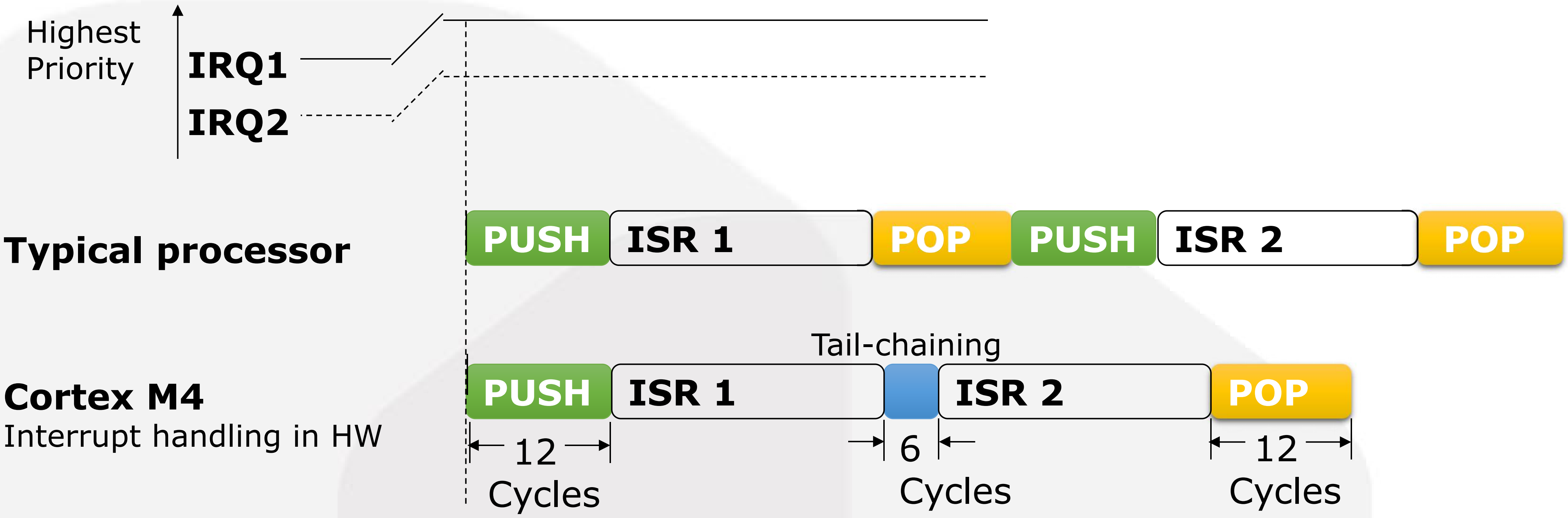
Nested Vector Interrupt Controller (NVIC)

Interrupts | Nested Vectored Interrupt Controller (NVIC)

- Handles exceptions and interrupts
- 8 programmable priority levels, priority grouping
- 7 exceptions and 71 Interrupts
- Automatic state saving and restoring: R0–R3, R12, LR, PSR, and PC
- Automatic reading of the vector table entry
- Pre-emptive/Nested Interrupts
- Tail-chaining
- Deterministic: always 12 cycles or 6 with tail-chaining



Interrupts | Latency: Tail Chaining



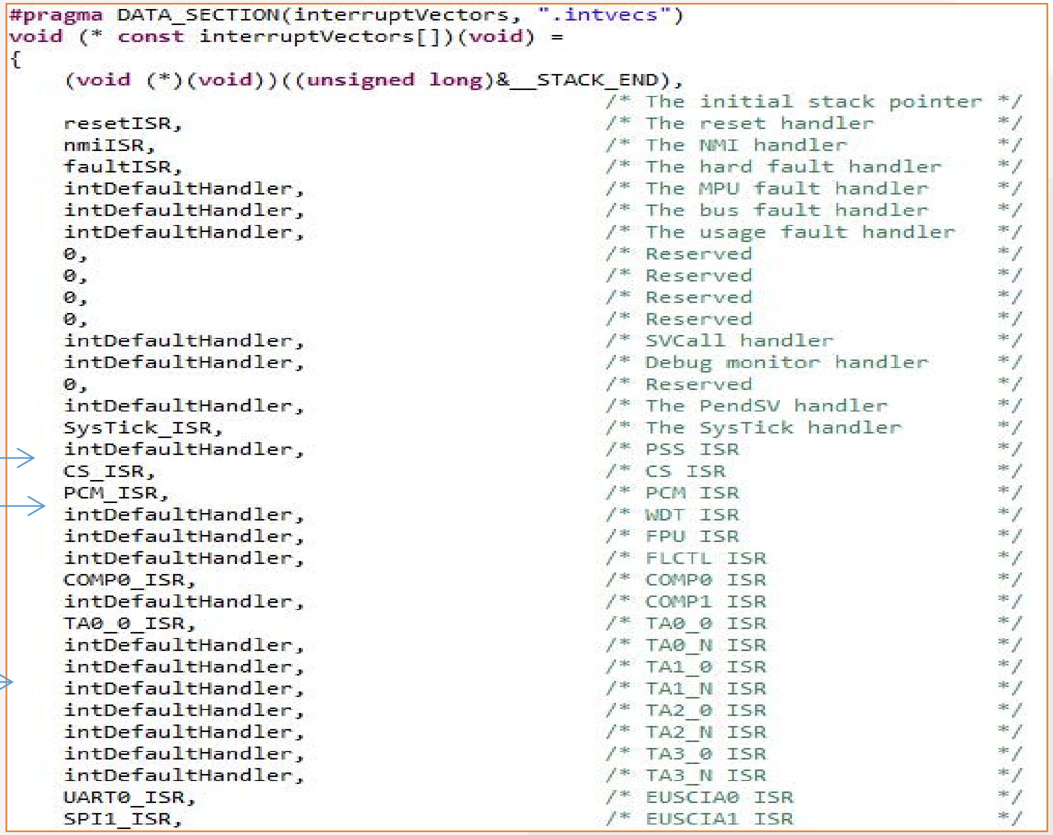
Saving 18 cycles:

24 cycles (POP + PUSH) → 6 cycles (Tail-chaining)

Interrupts | Declaration on MSP432

msp432_startup_ccs.c

Option 1: Declare the entire Interrupt Vector table



```
#pragma DATA_SECTION(interruptVectors, ".intvecs")
void (* const interruptVectors[])(void) =
{
    (void (*)(void))((unsigned long)&__STACK_END),
                                            /* The initial stack pointer */
    resetISR,                               /* The reset handler         */
    nmiISR,                                 /* The NMI handler           */
    faultISR,                               /* The hard fault handler    */
    intDefaultHandler,                      /* The MPU fault handler     */
    intDefaultHandler,                      /* The bus fault handler     */
    intDefaultHandler,                      /* The usage fault handler   */
    0,                                      /* Reserved                  */
    0,                                      /* Reserved                  */
    0,                                      /* Reserved                  */
    0,                                      /* Reserved                  */
    intDefaultHandler,                      /* SVCall handler            */
    intDefaultHandler,                      /* Debug monitor handler     */
    0,                                      /* Reserved                  */
    intDefaultHandler,                      /* The PendSV handler        */
    SysTick_ISR,                            /* The SysTick handler       */
    intDefaultHandler,                      /* PSS ISR                   */
    CS_ISR,                                 /* CS ISR                    */
    PCM_ISR,                                /* PCM ISR                   */
    intDefaultHandler,                      /* WDT ISR                   */
    intDefaultHandler,                      /* FPU ISR                   */
    intDefaultHandler,                      /* FLCTL ISR                 */
    COMP0_ISR,                              /* COMP0 ISR                 */
    intDefaultHandler,                      /* COMP1 ISR                 */
    TA0_0_ISR,                              /* TA0_0 ISR                 */
    intDefaultHandler,                      /* TA0_N ISR                 */
    intDefaultHandler,                      /* TA1_0 ISR                 */
    intDefaultHandler,                      /* TA1_N ISR                 */
    intDefaultHandler,                      /* TA2_0 ISR                 */
    intDefaultHandler,                      /* TA2_N ISR                 */
    intDefaultHandler,                      /* TA3_0 ISR                 */
    intDefaultHandler,                      /* TA3_N ISR                 */
    UART0_ISR,                              /* EUSCIA0 ISR               */
    SPI1_ISR,                               /* EUSCIA1 ISR               */
```

Option 2: MSP430 method

Use #pragma vector

```
#pragma vector = USCI_B0_VECTOR
__interrupt void USCI_B0_ISR(void)
{
  switch(__even_in_range(UCB0IV,12))
  {
  case  0: break;
  ……
}

//All unused interrupts trapped
#pragma vector = unused_interrupts
__interrupt void intDefaultHandler(void)
{
   //trap
}
```

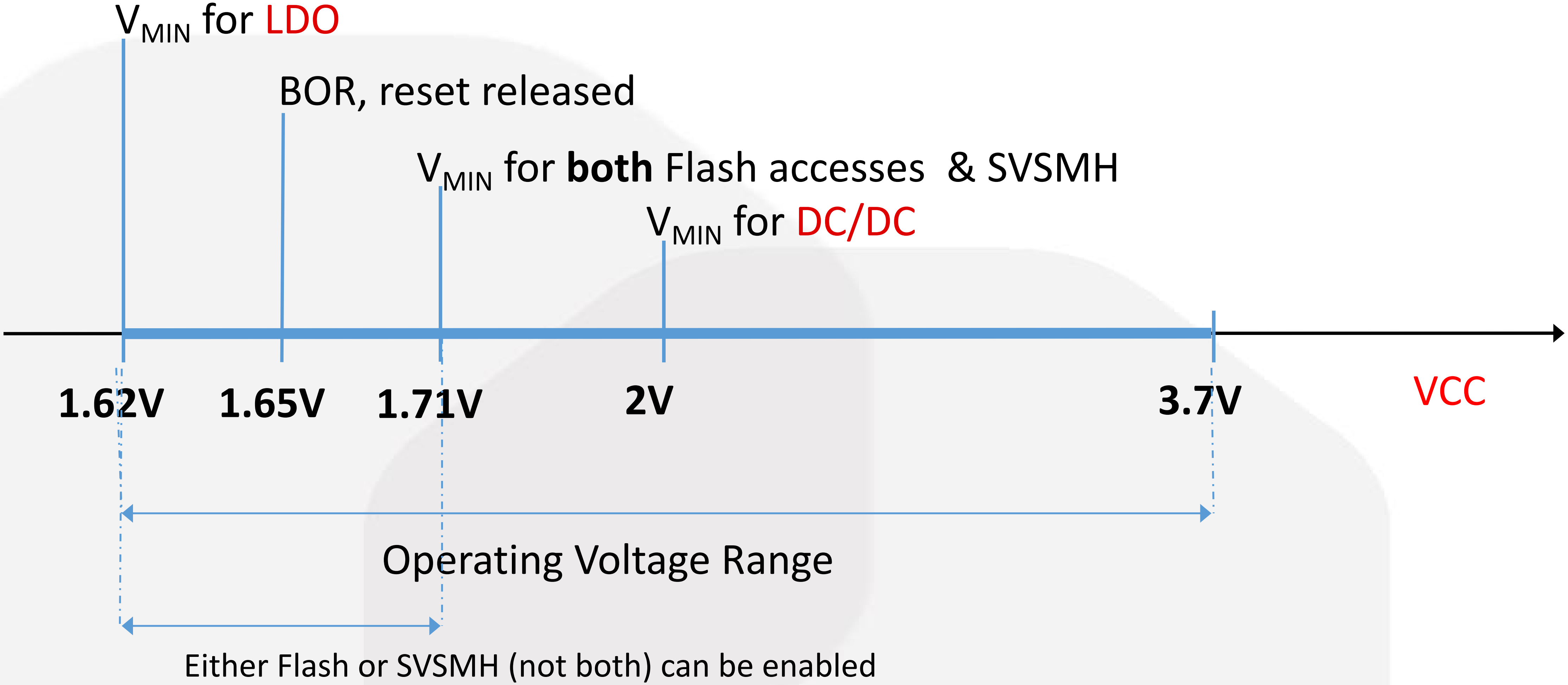
MCUs Training

MSP432™ : Power System

Power | Feature Overview

- Wide supply range with true 1.8V+/-10% operation: 1.62V-3.7V
- Two internal core voltages for system frequency power-scaling
  - 1.2V: 1-24MHz operation
  - 1.4V: 1-48MHz operation
- Two internal voltage regulators to adapt for power requirements/profiles
  - LDO: default regulator
  - DC/DC: additional regulator for better efficiency @ higher frequency
- Supply Voltage Monitor & Supervisor
- DriverLib-assisted power state transitions & configurations

Power | Operating Conditions



MCUs Training

MSP432™ : Clock System

CS | High-level Features

432

- Flexible clock sources & distribution:
  - 5 clocks from 7 sources (2 external, 5 internal)
  - Selections suitable for high-speed & low-power operations
- Wide range of operating frequency
  - 10kHz to 48 MHz
  - Fine intermediate steps with dividers & tuning
- Configurable & robust system:
  - Run-time lockable configuration
  - Failsafe mechanism with interrupts for external sources

CS | HF & LF Oscillators

Frequency		Oscillators	MCLK	SMCLK	HSMCLK	ACLK	BCLK	Comments
HF	1-48 MHz	DCO	✔	✔	✔			Internal integrated digitally controlled oscillator.
	1-48 MHz	HFXT	✔	✔	✔			High frequency crystal. Frequency range is SW configurable.
	24MHz	MODOSC	✔	✔	✔			Internal oscillator. option for peripherals such as ADC
	5MHz	SYSOSC						Internal, direct clock for ADC failsafe for HFXT
LF	32kHz	LFXT	✔	✔	✔	✔	✔	Low-frequency oscillator
	32kHz 128kHz	REFO	✔	✔	✔	✔	✔	Internal low-frequency oscillator. Failsafe* (32kHz) for LFXT
	10kHz	VLO	✔	✔	✔	✔		Internal ULP LF oscillator Clock selection for WDT

MCUs Training

MSP432™ : Software

Software | MSP Register-Level

- Traditional MSP register-level access code fully supported
- Header files provide complete register & bit definitions
- Complete portability for common peripherals across 16 & 32-bit platforms
- 100+ code examples for MSP430-shared & new MSP432 peripherals

c code example

```
WDTCTL = WDTPW | WDTHOLD;         // Stop WDT
P5SEL1 |= BIT4;                   // Configure P5.4 for ADC
P5SEL0 |= BIT4;
__enable_interrupt();             // MSP432: Enable master interrupt
SCS_NVIC_ISER0 = INT_ADC14_BIT;   // MSP432: Enable ADC14 interrupt
ADC14CTL0 = ADC14SHT0_2 | ADC14SHP | ADC14ON
ADC14CTL1 = ADC14RES_2
ADC14MCTL0 |= ADC14INCH_1;      // A1 ADC input select;
ADC14IER0 |= ADC14IE0;       // Enable conv. interrupt
SCS_SCR &= ~SCS_SCR_SLEEPONEXIT;  // MSP432: Wake up on exit from ISR
```

Software | Driver Library

```
GPIO_setAsPeripheralModuleFunctionOutputPin(PARAMETERS);
Timer_generatePWM(PARAMETERS)
```

API

```
P2DIR |= 0x04;
TA1CCTL1 = OUTMOD_7;
P2SEL |= 0x04;
TA1CCR1 = 384;
TA1CCR0 = 511;
TA1CTL = TASSEL_1 | MC_1 | TACLR;
```

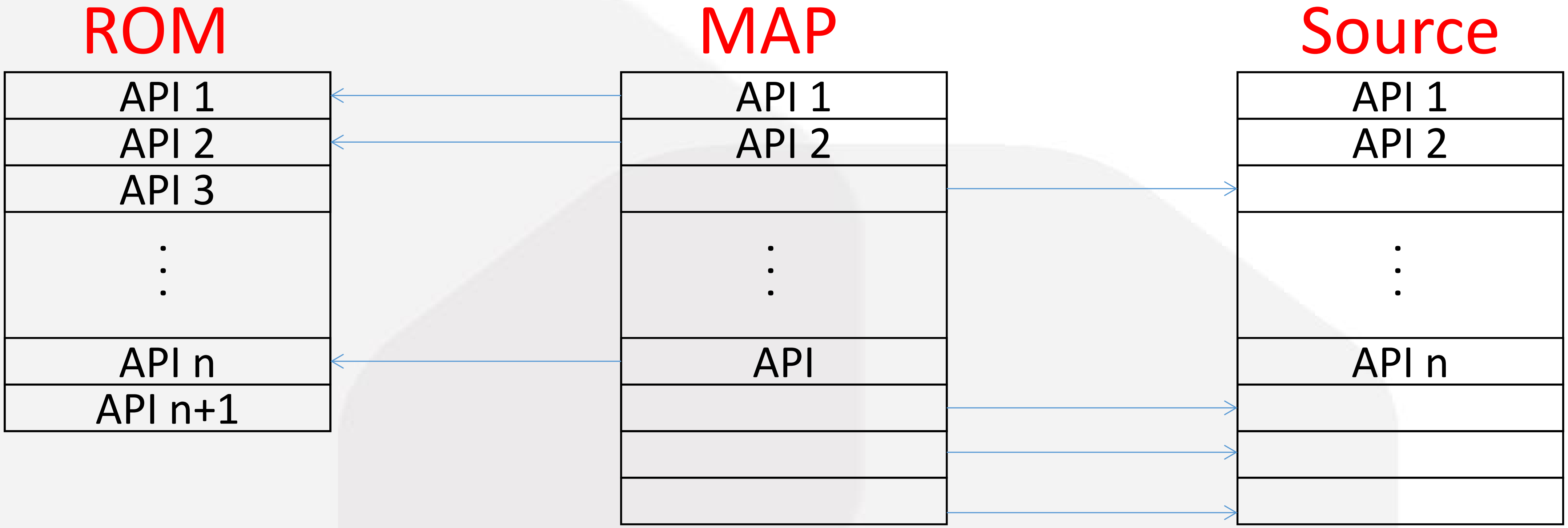
```
00101010
10010010
01010100
10010010
11001010
```

- Driver Library offers easy-to-understand functions
- No more cryptic registers to configure
- MSP430/432 shared peripherals also share DriverLib APIs → reduce porting effort

MSP432 DriverLib | ROM & Source

MAP file: method to ensure ROM API is always used

UNLESS there's an update (fix/enhancement) or APIs only available in Source



MSP432 DriverLib | Calling Convention

MSP432 DriverLib source (Flash)

- Include *driverlib* source folder in your project
- API Call: TimerA_generatePWM(param1, param2, etc.);

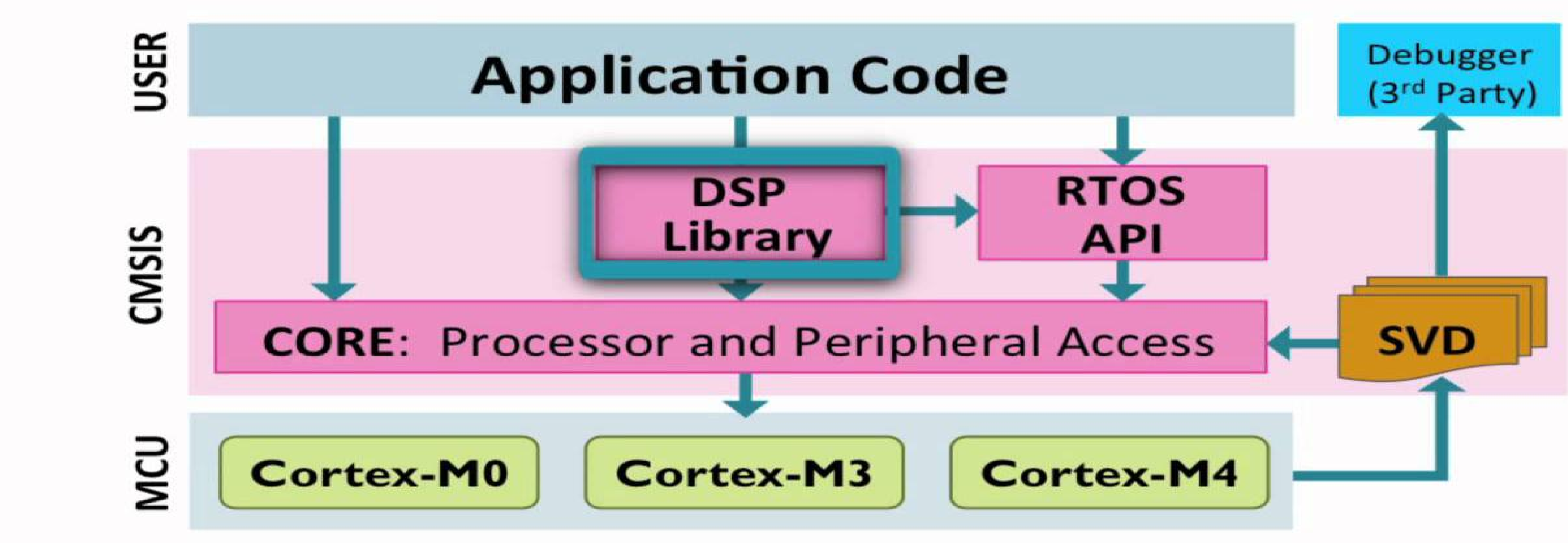
MSP432 DriverLib ROM

- Use *“rom.h”* header file
- API Call: ROM_TimerA_generatePWM(param1, param2, etc.);

- Use *“rom_map.h”* header file
- API Call: MAP_TimerA_generatePWM(param1, param2, etc.);

Software | CMSIS

- Cortex Microcontroller Software Interface Standard (CMSIS)



Printf

```
int putchar(int outChar)
{
      devIO = outChar;
      return outChar;
}
```

Assert

```
extern void __error__(char *pcFilename, const char *function, unsigned long line,
char *expr);
```

```
#ifdef DEBUG
#define ASSERT(expr) {
                    if(!(expr))
                    {      __error__(__FILE__,__func__,__LINE__,#expr);  }        }
#else
#define ASSERT(expr)
#endif
```

```
const char ASSERT_FAILED_STR[] = "\n\rASSERT failed at:\n\r  >File name: %s\n\r  >Function : %s\n\r  >Line No. : %d\n\r  >Condition: %s\n\r";
void __error__(char *pcFilename, const char *function, unsigned long line, char *expr)
{
        printf(ASSERT_FAILED_STR,pcFilename,function,line,expr);

        while (1)           //Assert fail
        {
                ;       //Waiting
        }
}
```

Thank You