Low power + high performance MSP432™ MCUs

Low-power at its best, performance at its core

MSP432 Deep-Dive

Texas Instruments

MSP432 Deep Dive | Agenda

- MSP432 Overview
- Cortex-M4F Core & Interrupts
- Power System
  - Lab 1: Power modes
- Clock System
  - Lab 2: Clock manipulation
- Memory System
  - Lab 3: [Take Home] Memory configuration & protection
- Digital Modules:
  - Digital I/O, Timers, DMA, USCI
- Analog Modules:
  - ADC14, REF, COMPE
  - Lab 4 [: [Option 1] ADC14
- Security
- Software
  - Lab 5: [Option 2] ADC14 w/ Software
- Migration
  - Lab 6: Platform Porting
- Open discussion & feedback

MSP432 Overview

MSP Offers a Complete Selection of Low-Power Microcontrollers

MSP Low-Power MCUs

Ultra-Low-Power MCUs

The World's lowest power MCUs

MSP430FRxx

MSP430™ 16-bit ULP MCUs

Low-Power + Performance MCUs

MCUs with expanded MHz and integration

MSP430F5x/6x, MSP432

MSP 16/32-bit LP + P MCUs

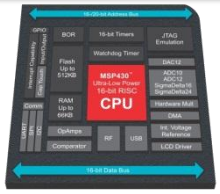
Security + Communications MCUs

MCUs with integrated RF and more!

RF430

RF430™ 16/32-bit MCUs

The Lowest Power → More Integration → Embedded RF



Texas Instruments

Top Customer Problems… Solved!

Increased Processing Capability

32-bit 48-MHz Cortex M4F provides 2x more performance than M3, at ½ the power

Low Power Operation

Industry's lowest power general purpose Cortex M device allows for 95uA/MHz of active power and 850nA of standby power

Tools & Ease of Use

Extract MSP power efficiency and ARM® performance through easy-to-use hardware and software tools

Scalability

Achieve seamless portability between MSP's 16-bit and 32-bit portfolio

Increased processing capability

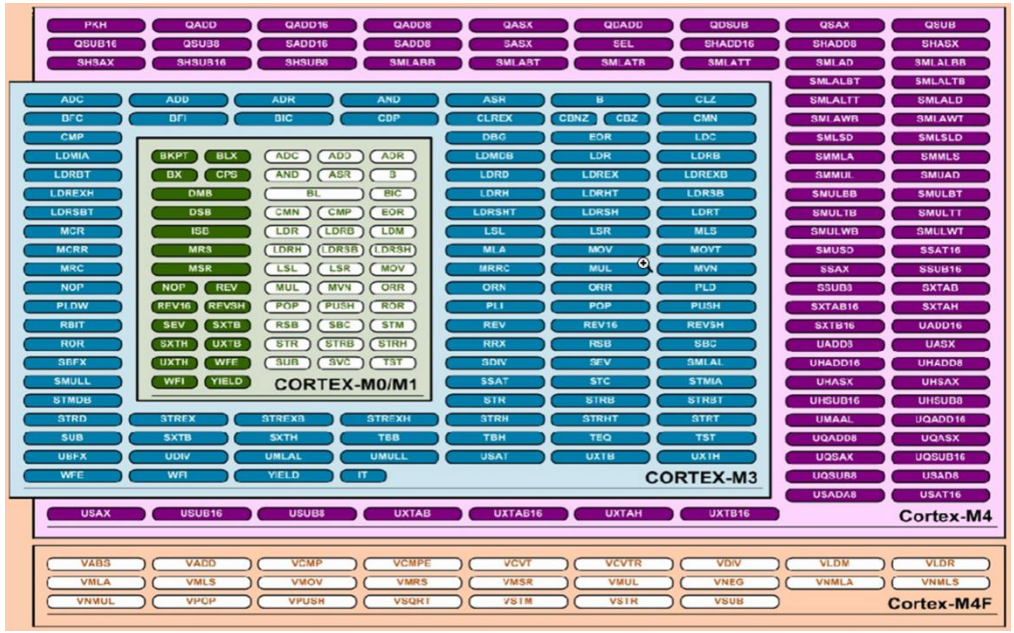
Selecting the highest performance Cortex M core

- 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

ARM Cortex M4F Advantages



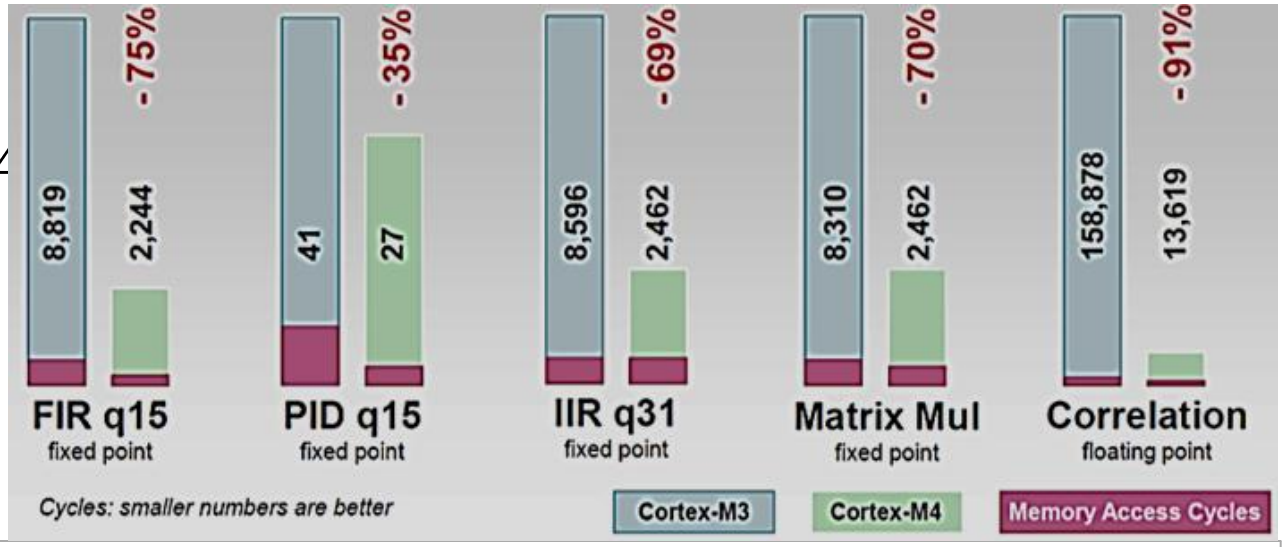
Cortex M3 vs. Cortex M4

(*SIMD* + *FPU*) DSP Library Benchmark:

- Fixed-point ~ 2x-4x faster
- Floating-point ~ 10x faster

Source: ARM CMSIS Partner Meeting Embedded World, Reinhard Keil

Utilize the full ARM instruction set with M4



Low power operation

Silicon optimized for low-power

- Power optimized capabilities
  - Wide voltage range
  - Integrated LDO & DC/DC
  - Selectable RAM retention
  - Simultaneous Flash read/write
- Reduce CPU active cycle
  - 128 bit Flash buffer pre-fetch
  - 1MSPS ADC14
  - 8 channel DMA
  - Peripheral,SRAM memory bit-band
  - NVIC with Tail-chaining

Software optimized for low-power

- Power optimized software
  - Driver Lib in ROM
  - MSPWare
- Tools to optimize power
  - ULP Advisor
  - EnergyTrace+ and Debuggers

Lower power consumption

- 95uA/MHz Active power , 850nA Sleep with RTC
- 5 flexible low power modes to enable ultra-low power consumption
- Ultra-low power peripherals: 200uA 1MSPS 14-bit ADC (@1.8V)
- Ultra-low power 90nm TI-owned process
  - Utilize low supply voltage, and run/erase code from flash as low as 1.62V
  - Individual SRAM bank enable and retention saves power by using only the memory you need

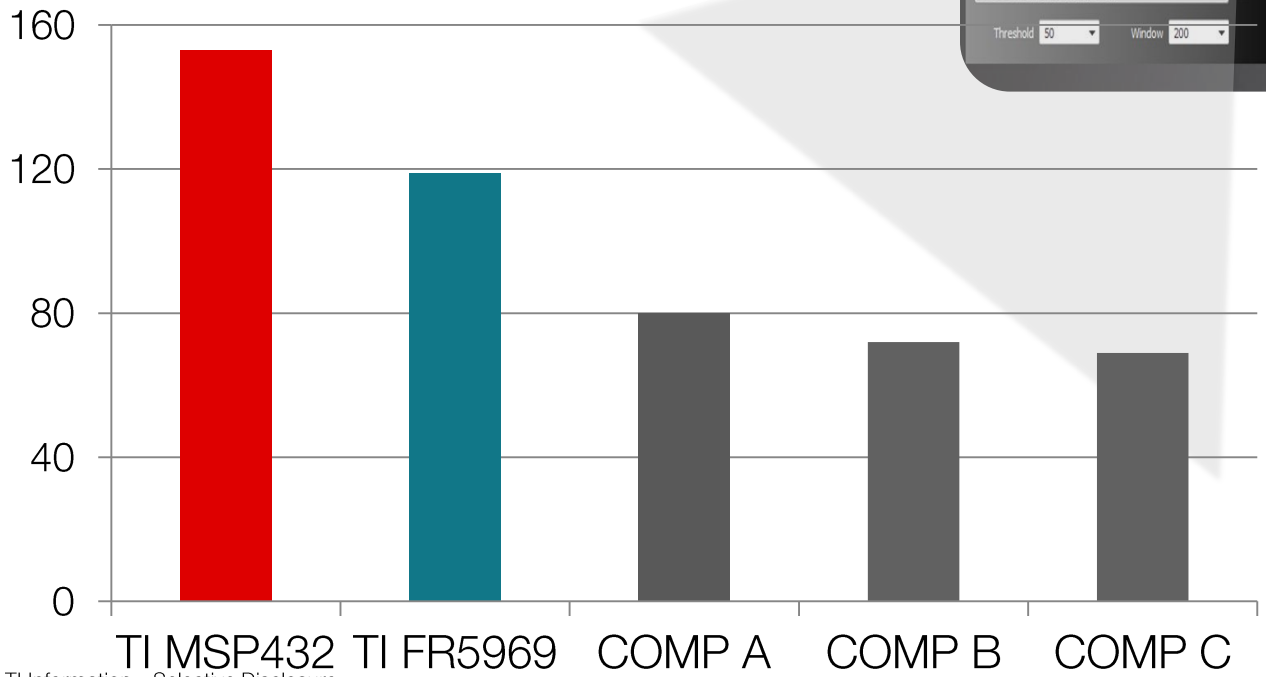
MSP432	ACTIVE	LPM0	LOW FREQUENCY ACTIVE	LPM3	LPM3.5	LPM4.5
Current	100uA/MHz (DCDC) 166uA/MHz (LDO)	65uA/MHz (DCDC) 100uA/MHz (LDO)	<40uA	<900nA	<670nA	<100nA
CPU				Retention		
FLASH						
SRAM				Retention	Retention	
LDO	In LDO mode	In LDO mode	Low Drive Mode	Low Drive Mode	Low Drive Mode	
DC-DC	In DCDC mode	In DCDC mode				
PSS Bandgap				Sampled Mode	Sampled Mode	
Clocks				Only BCLK	Only BCLK	
Core Domain Logic				Retention		
Backup Domain Logic (RTC/WDT)				Active	Active	
I/O State				Retention	Retention	Retention

TI Information – Selective Disclosure

MSP = ULP

EEMBC's ULPBench provides a true comparison of microcontroller current consumption and efficiency

ULPBench Scores



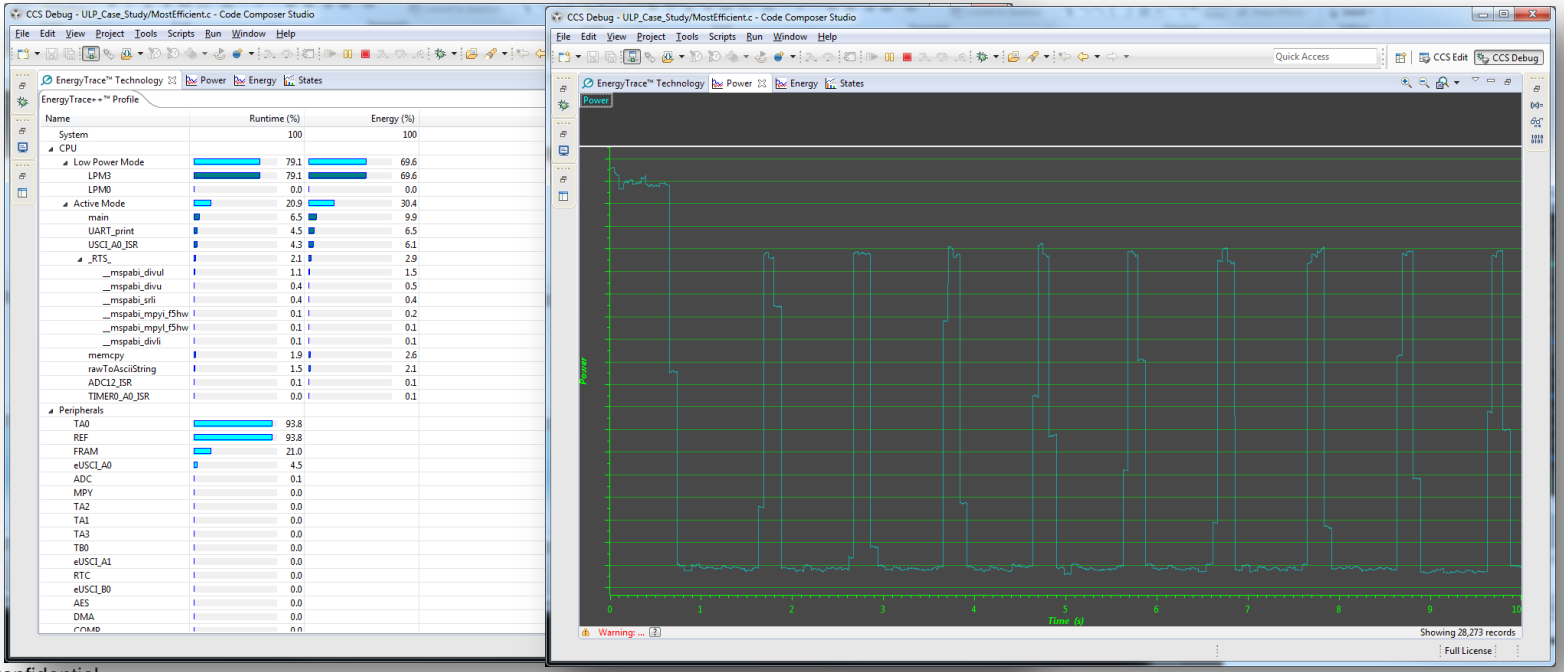
90% better score than our closest competitor!

See for yourself at www.eembc.org/ulpbench

Texas Instruments

EnergyTrace+™ technology

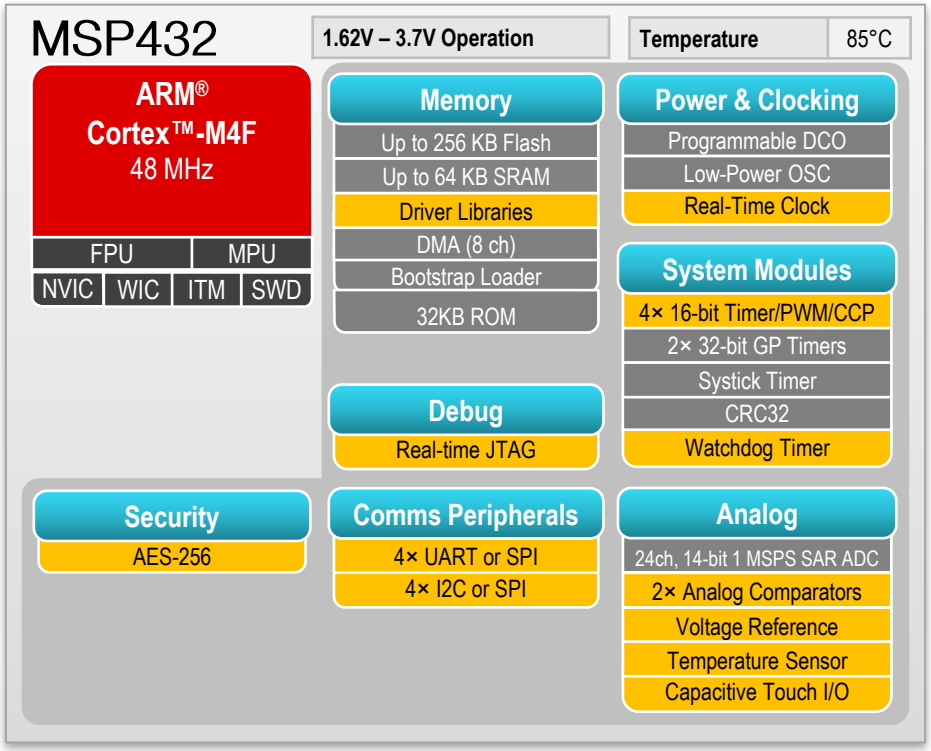
- Provides a complete ecosystem for real-time power debugging and quickens time to market. Spends less time debugging and more time developing.
- Graphical User Interface in TI's Code Composer Studio integrated development environment (IDE) and IAR Systems' Embedded provides energy profiles of your application
  - Current measurement and CPU states can be tracked over time to help identify power black holes



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
- **Simplified portability from MSP430** - Leverage software & know-how from existing MSP430 designs
  Using 430 Peripherals, Analog & Low Power Modes



Same as MSP430

Kits

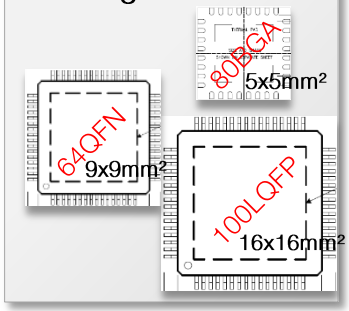
LaunchPad
- Designed for evaluation and initial development
- Includes on-board emulator
- $12.99

Target Board
- Designed for advanced development
- $89

Tools & Software

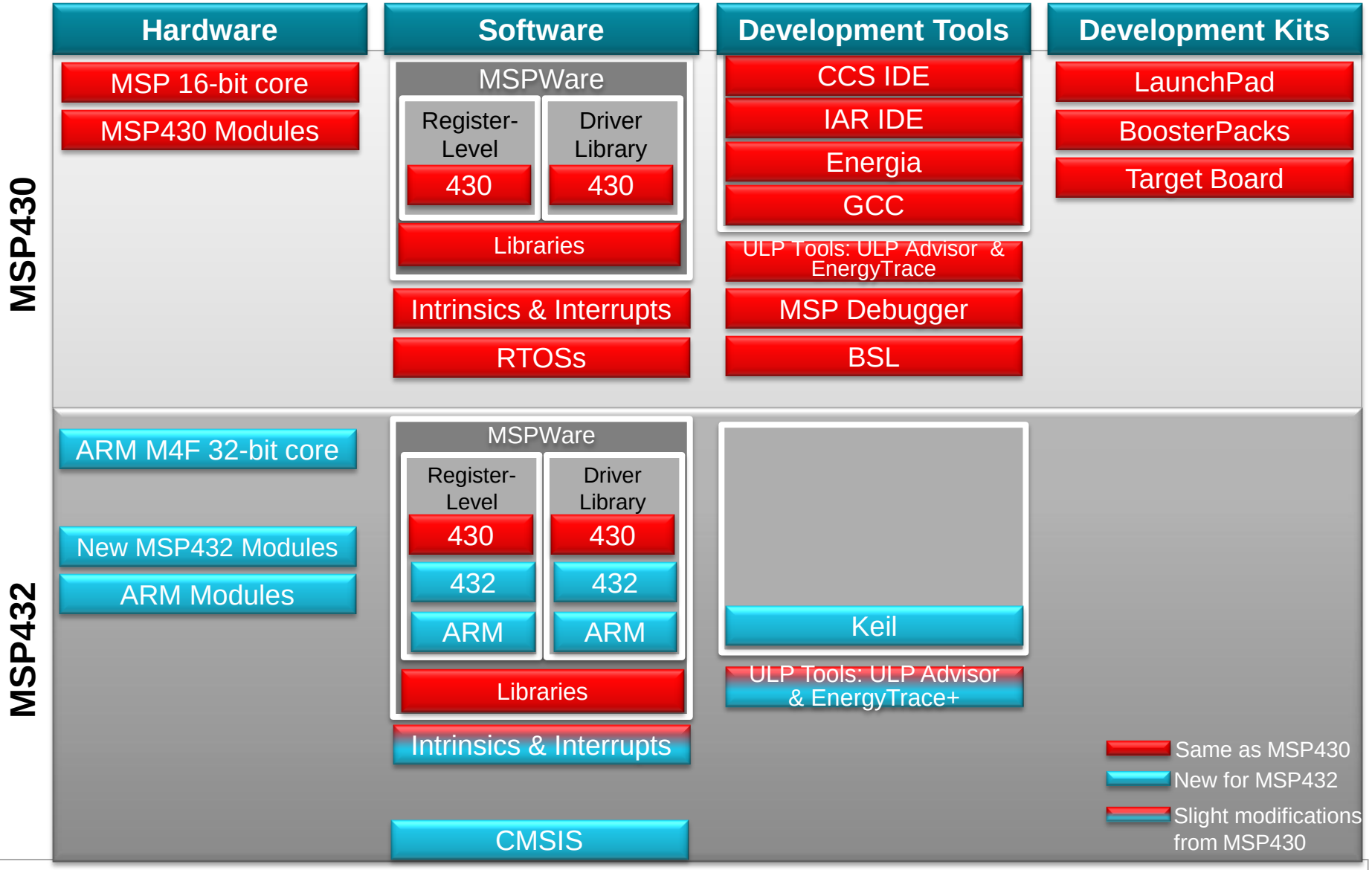
- **MSPWare** – leverage C-code portable MSP430 peripherals and analog
- TI RTOS Support
- ARM 3rd Party Ecosystem
- Code Composer Studio™, IAR, KEIL IDEs, and gcc

Packages



TI Information – Selective Disclosure

MSP Platform Portability



It's all in MSPWare

Home

1 - Select a Device or Development Tool

2 - Filter Results [optio

- Devices - (42577)
  - MSP430 - (42577)
    - MSP430F1XX - (6154)
    - MSP432P4XX - (244)
      - MSP432P401 - (244)
        - Documents - (3)
          - User's Guide
        - Datasheets - (1)
          - MSP432P401
        - Erratasheets - (1)
          - MSP432P401
        - Examples - (210)
        - Training - (28)
          - Deep Dive Training
        - App Notes and Guides -
          - MSP Platform Portin
          - CCS User's Guide fo
          - IAR User's Guide for

MSP432P401

MSP432P401 incorporates M4 processo

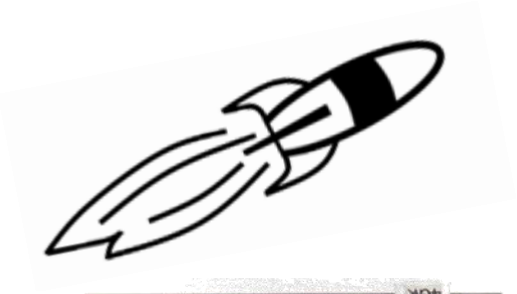
msp432p401_adc14_06	ADC14, Repeated Sequence of Conversions
msp432p401_adc14_10	ADC14, Sample A12 Temp and Convert to oC and oF
msp432p401_adc14_21	ADC14, Window Comparator, 2.5V ref
msp432p401_aes_01	AES256 Encryption & Decryption
msp432p401_comp_01	COMP output Toggle in Sleep Mode; input channel C01; Vcomp
msp432p401_comp_05	COMPE Hysteresis, COUT Toggle in SL; High speed mode
msp432p401_crc32_01	CRC32 in CRC16 mode, Compare CRC output with software-ba
msp432p401_cs_01	Output MCLK & ACLK @ DCO default frequency
msp432p401_cs_02	Configure MCLK for 12MHz operation
msp432p401_cs_03	Device configuration for operation @ MCLK = DCO = 48MHz
msp432p401_cs_06	LFXT sources ACLK. Toggles P4.2
msp432p401_euscia0_uart_01	eUSCI_A0 UART echo at 9600 baud using BRCLK = 12MHz
msp432p401_euscia0_uart_03	USCI_A0 External Loopback test @ 115200 baud
msp432p401_euscia3_spi_09	eUSCI_A3, SPI 3-Wire Master Incremented Data

Your one-stop shop for all technical collateral

- User's Guides
- Application Notes
- Deep-dive Training
- Code Examples

Introducing the MSP432 LaunchPad

Develop high performance applications that benefit from low power operation



Features

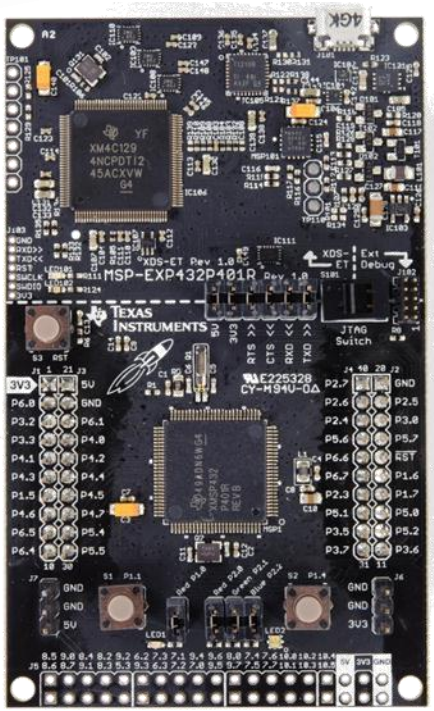
- Low-power, high performance MSP432P401R MCU
- 40 pin BoosterPack Connector, and support for 20 pin BoosterPacks
- Onboard XDS-110ET emulator featuring EnergyTrace+ Technology
- 2 buttons and 2 LEDs for User Interaction
- Back-channel UART via USB to PC

Kit Includes

- Development board with demo application
- USB cable
- Quick start guide

Software

- MSPWare featuring example code, User's Guides, Application notes, training, and more
- Out-of-box LaunchPad GUI



MSP-EXP432P401R
$12.99

MSP432 device options

Part Number	Flash (KB)	SRAM (KB)	ADC14 Chan	Comp-0 Chan	Comp-1 Chan	Timer A	eUSCI		20mA Drive I/O	Total I/O	Package Type
							Chan A: UART/ IrDA/SPI	Chan B: SPI/I2C			
MSP432P401RIPZ	256	64	24/ext 2/int	8	8	5,5,5,5	4	4	4	84	100 LQFP 16x16mm
MSP432P401MIPZ	128	32	24/ext 2/int	8	8	5,5,5,5	4	4	4	84	100 LQFP 16x16mm
MSP432P401RIZXH	256	64	16/ext 2/int	6	8	5,5,5	3	4	4	64	80 BGA 5x5mm
MSP432P401MIZXH	128	32	16/ext 2/int	6	8	5,5,5	3	4	4	64	80 BGA 5x5mm
MSP432P401RIRGC	256	64	12/ext 2/int	2	4	5,5,5	3	3	4	48	64 QFN 9x9mm
MSP432P401MIRGC	128	32	12/ext 2/int	2	4	5,5,5	3	3	4	48	64 QFN 9x9mm

Before we begin

Texas Instruments

Some features you will see frequently

MSP432 DriverLib API available for that feature

Feature new to MSP432 (compared to MSP430/ARM)

MSP-DNA feature, available across MSP platforms

Ultra-low-power (ULP) feature

Cortex

Feature inherited from ARM Cortex-M architecture/peripheral

Software & IDE

- Code Composer Studio & TI Resource Explorer
- MSP432 Header files & project configuration
- MSP432 DriverLib
  - Developer's/API Guide
  - *Ctrl+Space* for auto-completion

It’s all in MSPWare

Packages: All Devices: All Topics: All

enter search keyword Address:

- MSP430ware
  - Devices
    - MSP430F1xx
    - MSP430F2xx
    - MSP430G2xx
    - MSP430F4xx
    - MSP430F5xx/6xx
    - MSP432P4xx
      - User's Guide
      - User's Guide ARM Chapters
      - User's Guide Timer32 Chapter
      - Datasheets
      - Code Examples
      - Training
        - Deep Dive Training
          - MSP432 Deep Dive Training Lecture
          - Lab 1: Power
          - Lab 2: Clock
          - Lab 3: Memories
          - Lab 4: ADC14
          - Lab 5: Software
          - Lab 6: Platform Porting
          - Lab 6: Platform Porting (MSP430F5529)
        - App Notes and Guides
          - MSP Platform Porting Guide
          - CCS User's Guide for MSP432
          - IAR User's Guide for MSP432
  - Development Tools
  - Libraries
    - Driver Library
      - Release Notes
      - MSP430F5xx_6xx
      - MSP430FR57xx
      - MSP430FR5xx_6xx
      - MSP432P4xx
        - User's Guide
        - API Programmer's Guide

MSP432 Entry

!!!New!!! Training

MSP432 DriverLib

Nested Vector Interrupt Controller (NVIC)

12.4.2 Function Documentation

12.4.2.1 void Interrupt_disableInterrupt (uint32_t *interruptNumber*)

Disables an interrupt.

Parameters

interrupt	specifies the interrupt to be disabled.

The specified interrupt is disabled in the interrupt controller. Other enables for the interrupt (such as at the peripheral level) are unaffected by this function.

See Interrupt_enableInterrupt for details about the interrupt parameter

Returns
None.

Referenced by ADC14_unregisterInterrupt(), AES256_unregisterInterrupt(), Comp_unregisterInterrupt(), CS_unregisterInterrupt(), DMA_unregisterInterrupt(), FlashCtl_unregisterInterrupt(), GPIO_unregisterInterrupt(), I2C_unregisterInterrupt(), MPU_disableInterrupt(), PCM_unregisterInterrupt(), PSS_unregisterInterrupt(), RTC_unregisterInterrupt(), SPI_unregisterInterrupt(), Timer32_unregisterInterrupt(), TimerA_unregisterInterrupt(), UART_unregisterInterrupt(), and WDT_unregisterInterrupt().

12.4.2.2 bool Interrupt_disableMaster (void)

Disables the processor interrupt.

This function prevents the processor from receiving interrupts. This function does not affect the set of interrupts enabled in the interrupt controller; it just gates the single interrupt from the controller to the processor.

Returns
Returns **true** if interrupts were already disabled when the function was called or **false** if they were initially enabled.

Texas Instruments

MSP432 DriverLib | ROM & Source

MSP432 DriverLib available as source

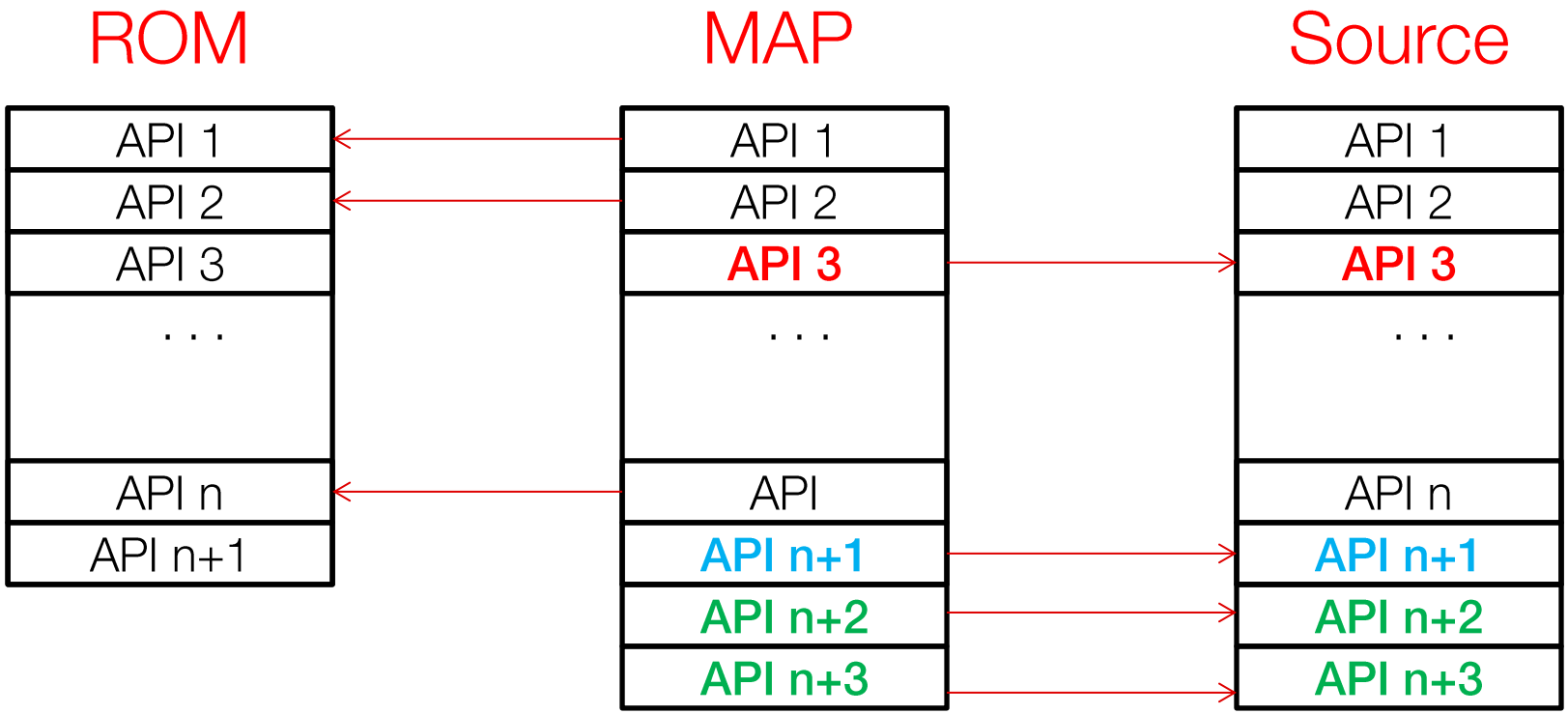
- Always the latest, most updated version
- Library available as BSD source = industry standard
- Flexibility for users to further customize/enhance

MSP432 DriverLib implemented & tested in ROM

- ROM has lower current consumption
- No wait-state [even @ max speed = 48MHz]
- Frees up flash space (25kB) for application code
- Fully validated & robust ROM code: no future major changes

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



Legend: Bug fix | Feature Enhancement | New API

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.);

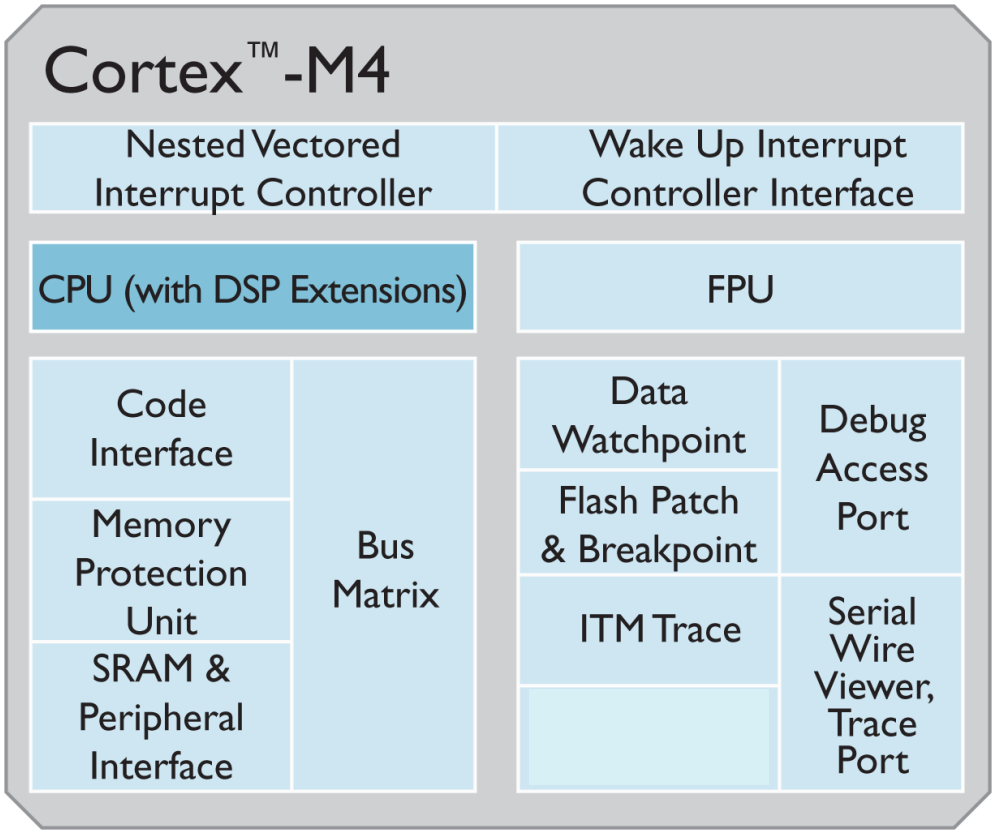
Patches new features or bug fixes:

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

Cortex-M4F Core

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-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

MSP432 | CoreMark = 3.41CM/MHz

Core	Device	CoreMark
Cortex-M4	MSP432	3.41
	Competitor A	3.34 [168MHz-Ohs, --no_size_constraints]
Cortex-M3	Competitor A	2.98
	Competitor B	2.15
Cortex-M0+	Competitor C	2.28
	Competitor B	2.07

Test Conditions

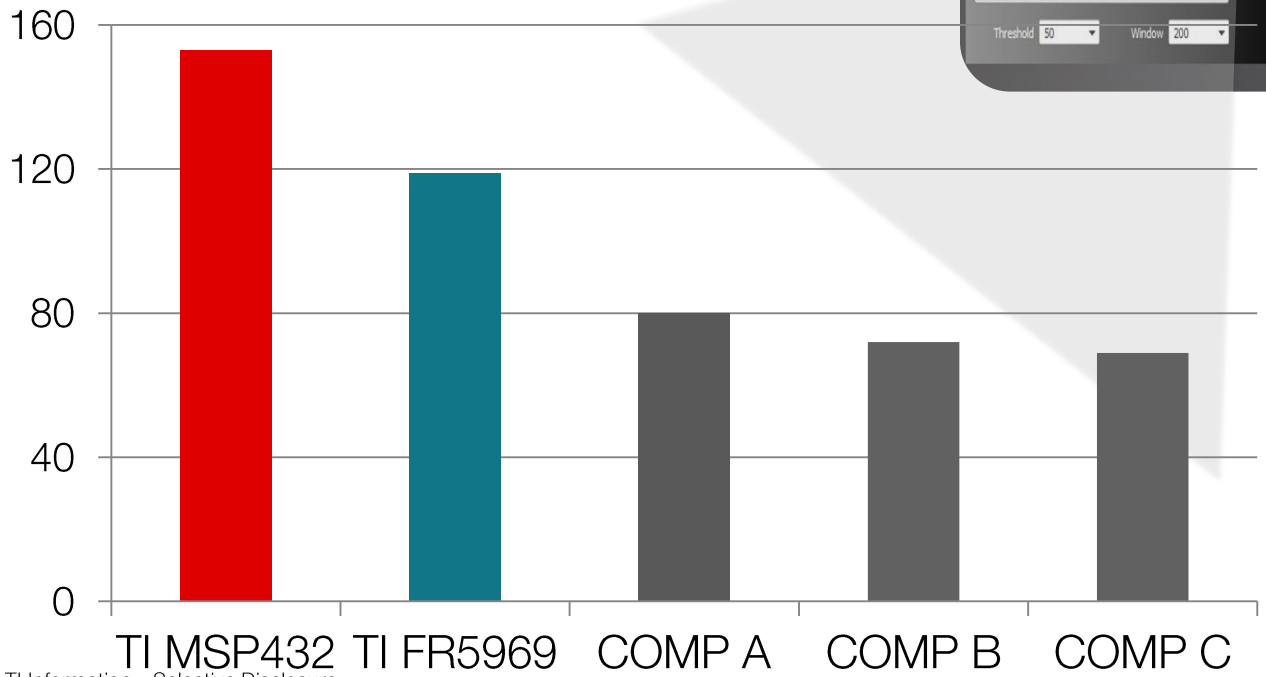
- Number measured with IAR
- 3MHz, Active Mode with VCORE=0

*Source: MSP-Benchmark twiki

ULPBench

EEMBC's ULPBench provides a true comparison of microcontroller current consumption and efficiency

ULPBench Scores



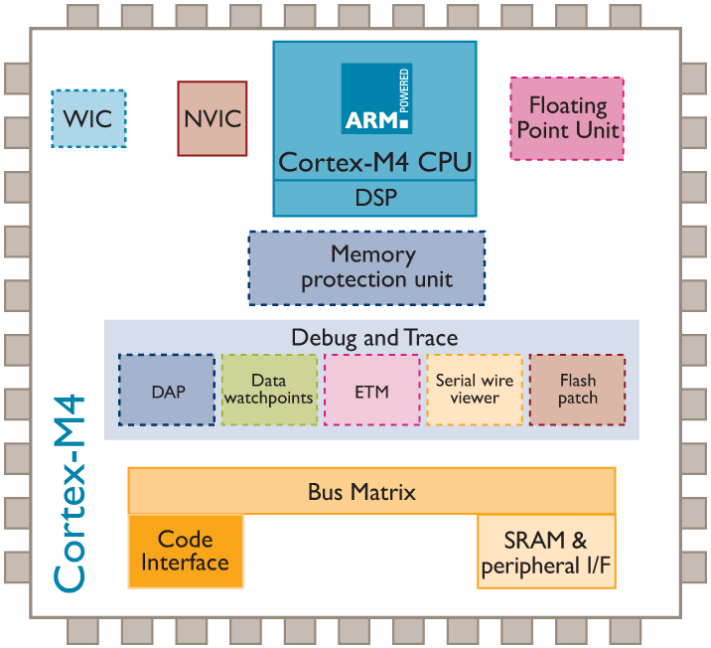
90% better score than our closest competitor!

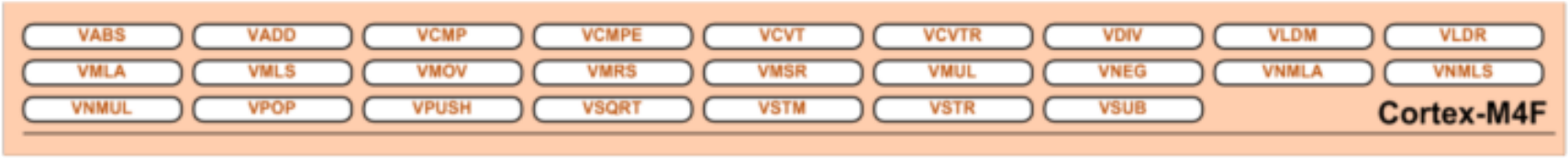
www.eembc.org/ulpbench

Texas Instruments

FPU | Floating-Point Unit

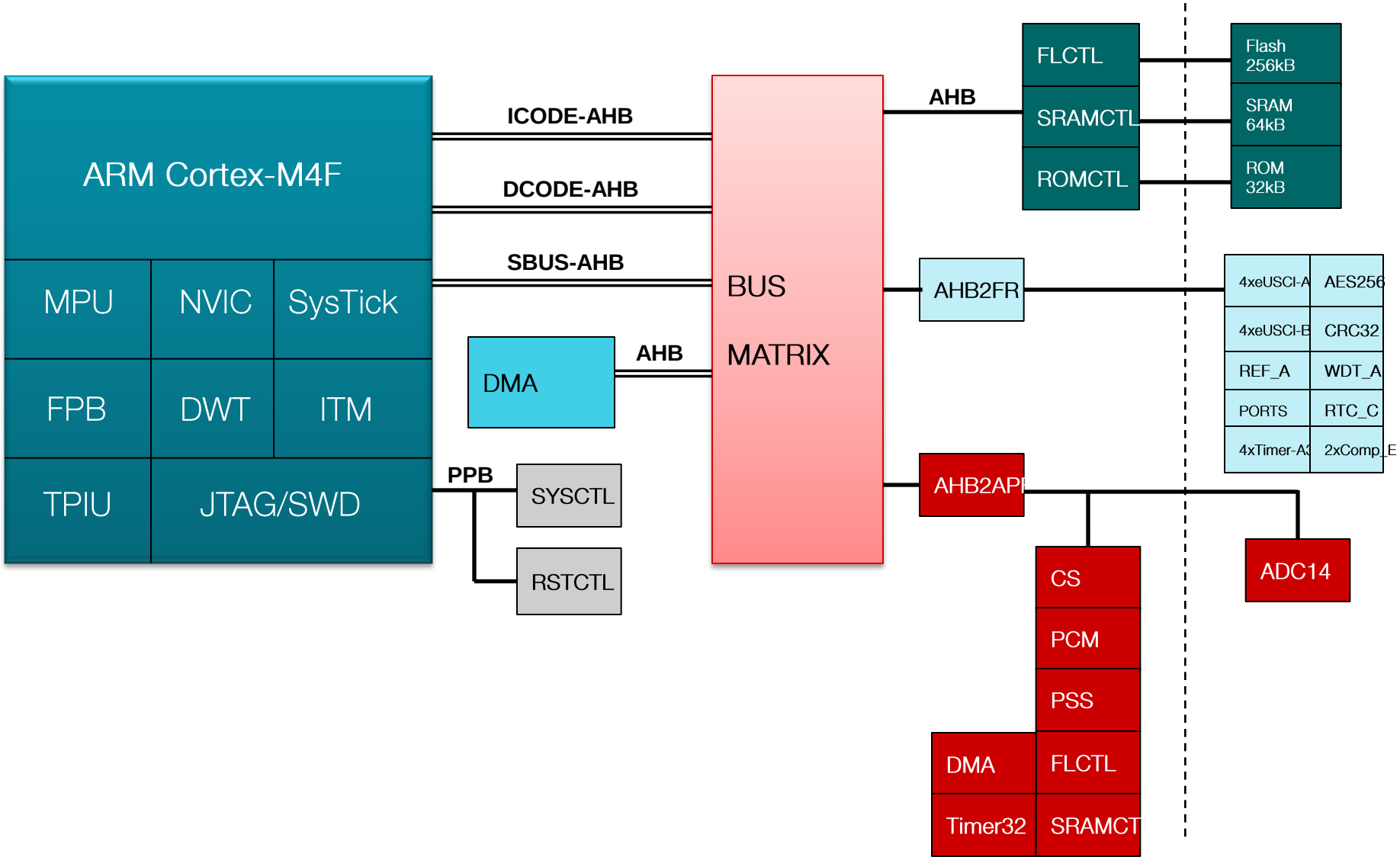
- The FPU provides floating-point computation functionality that is compliant with the IEEE 754 standard
- Enables conversions between fixed-point and floating-point data formats, and floating-point constant instructions
- The Cortex-M4F FPU fully supports single-precision:
  - Add
  - Subtract
  - Multiply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





TI Information – Selective Disclosure

MSP432P401xx Bus Architecture



Texas Instruments

MSP432 Power System

Power | Feature Overview

432

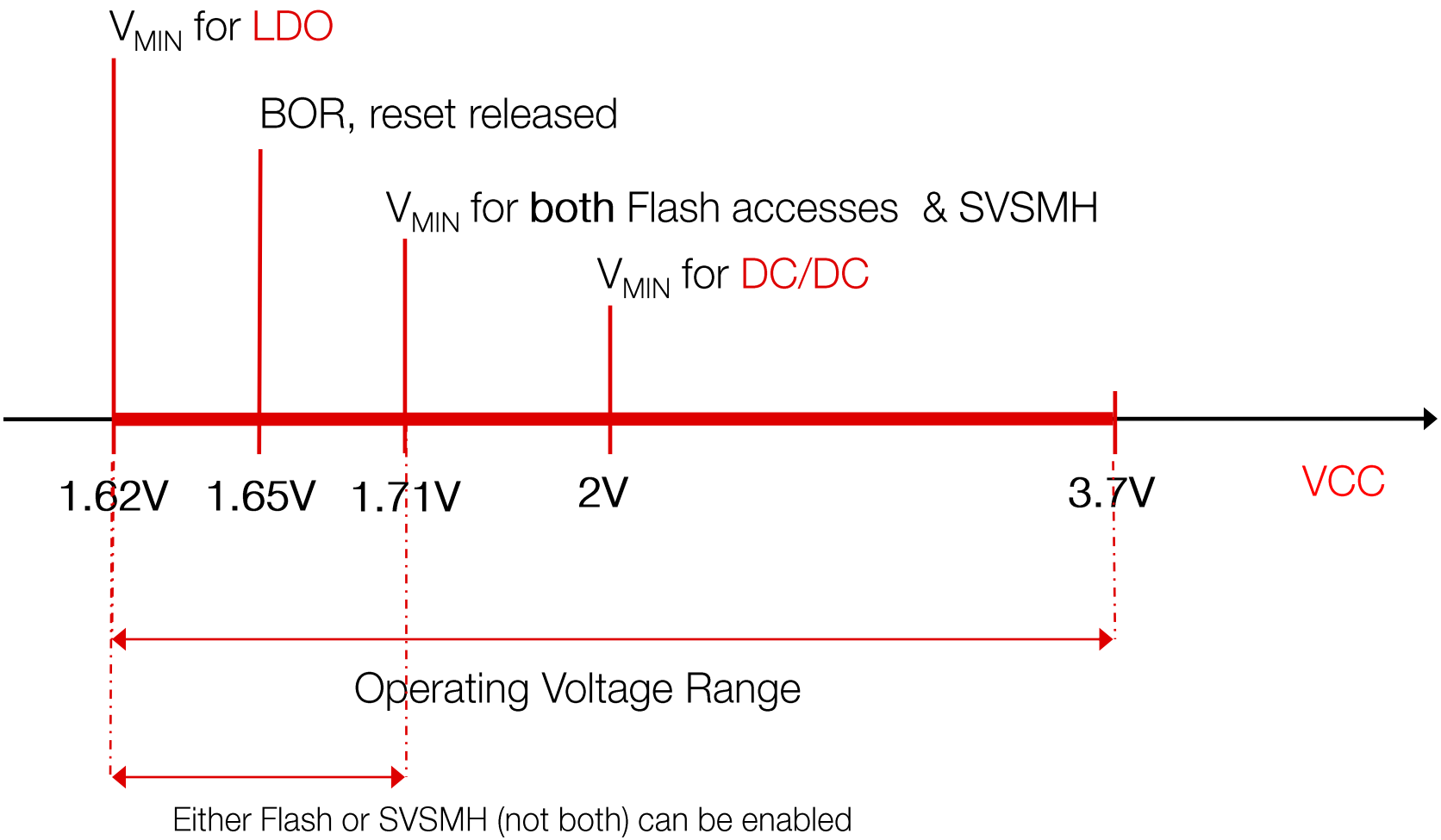
- Wide supply range with true 1.8V+/-10% operation: 1.62V-3.7V

- 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

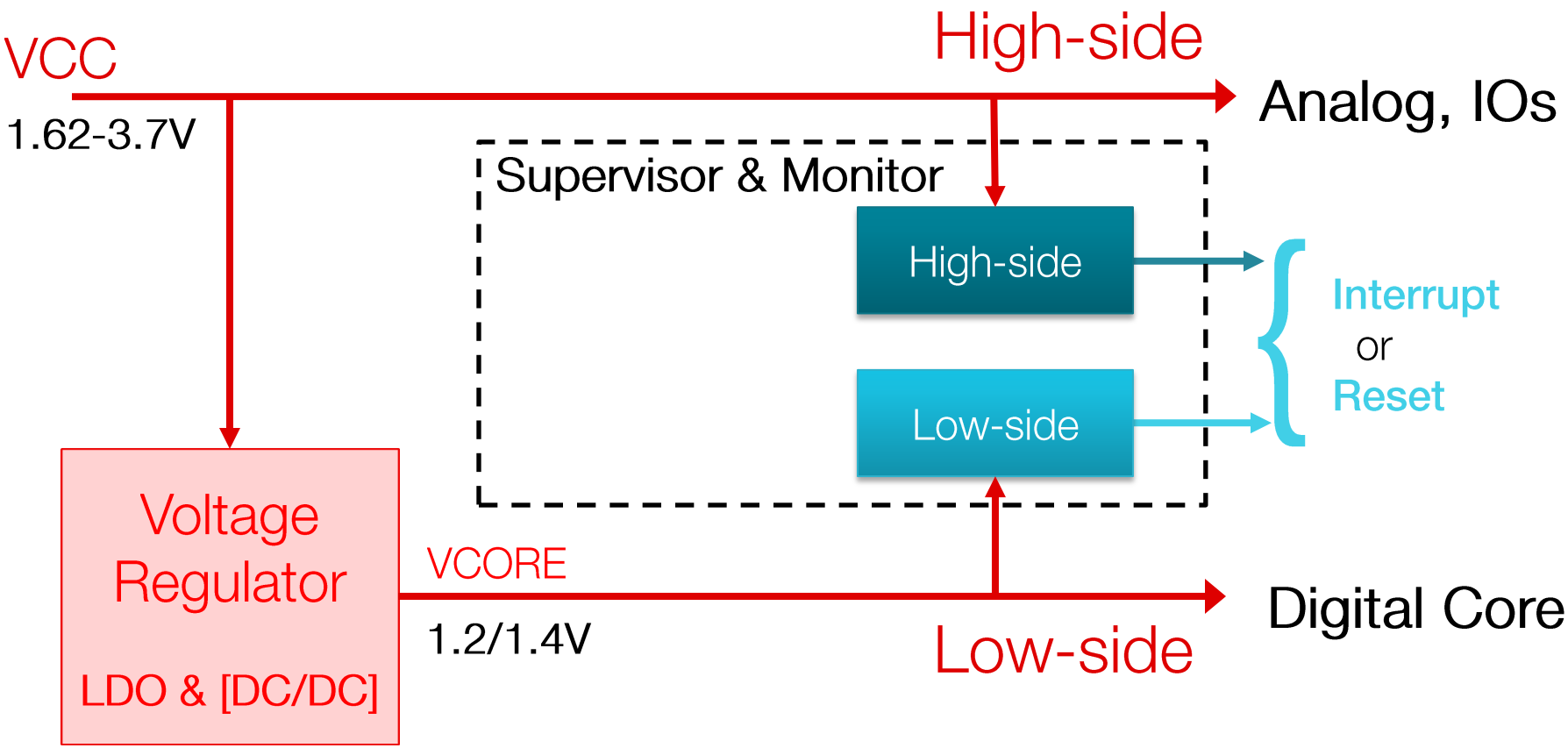
- Supply Voltage Monitor & Supervisor
  - Low performance modes for extremely low power in LPM3/4/x.5

- DriverLib-assisted power state transitions & configurations

Power | Operating Conditions

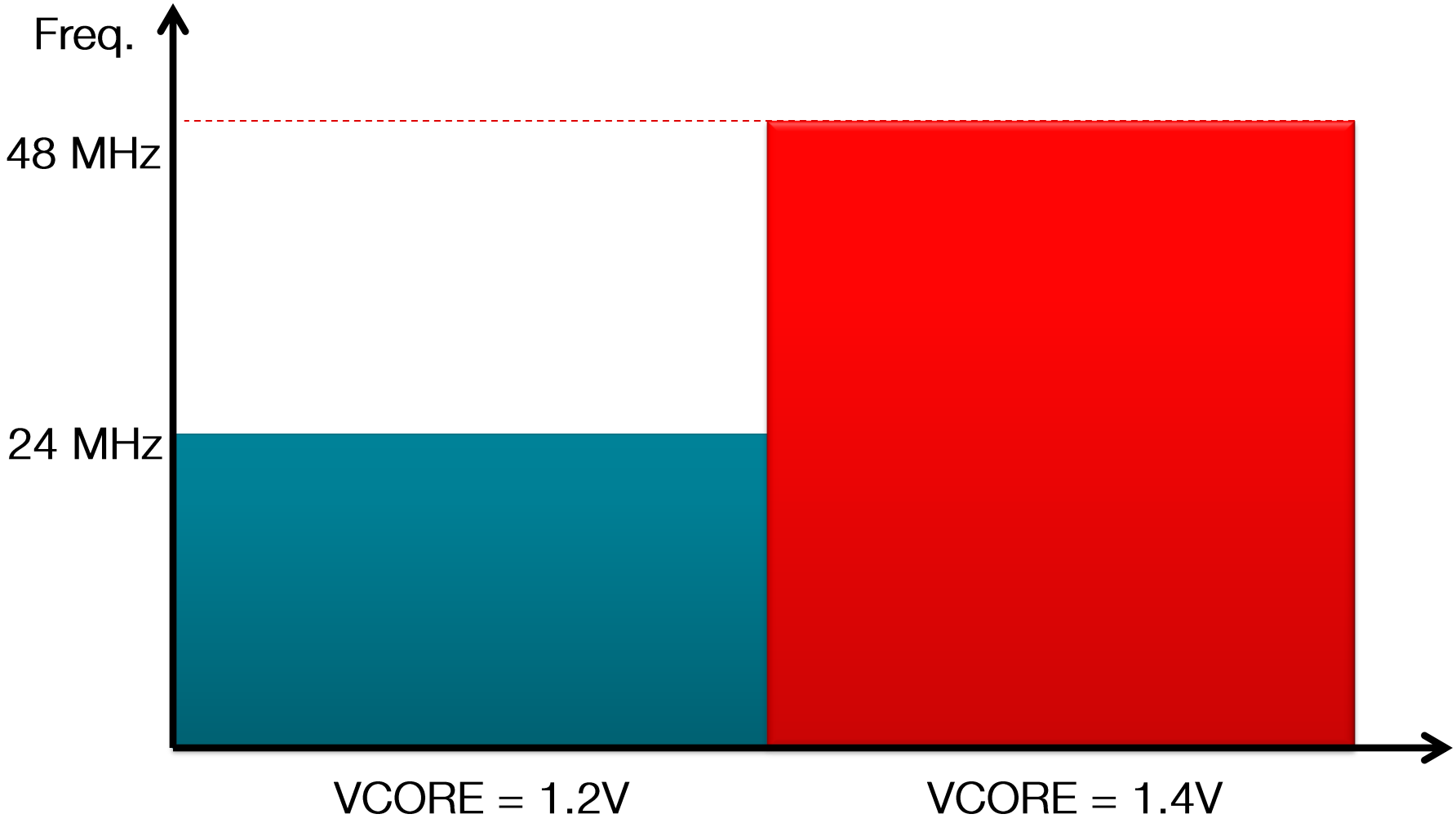


Power | Supply System Overview

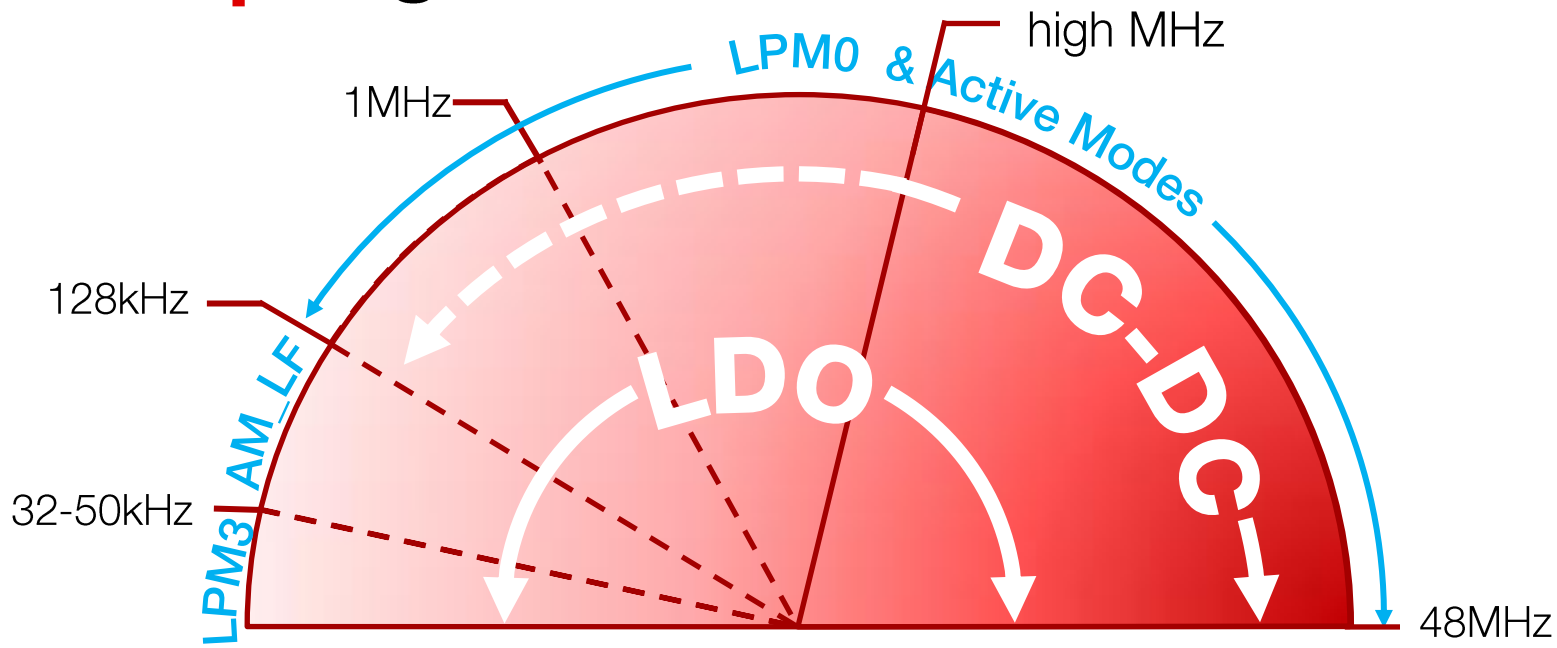


TI Information – Selective Disclosure

Power | Core Voltage & System Frequency



Power | Regulators: LDO & DC-DC



LDO	DC-DC
Default regulator at startup	Secondary, requires external inductor
VCC = [1.62V-3.7V]	VCC = [2.0V-3.7V]
Available in all power modes	Available in LPM0 & Active Modes
Flexible with scalable output loads for low power modes	Efficient, optimized for high-speed/high-load operations
Fast on/off switching operations	Slow on/off/failsafe switching from/to LDO

MSP430 & 432 Low-Power Modes

MSP430	MSP432	Industry/ARM description	Comments
Active	Active	Active Mode	CPU & peripherals
	Low-Frequency Active	Low-Power Run	CPU & peri. <128kHz
LPM0	LPM0	Sleep	Peri. on, CPU off
	Low-Frequency LPM0		Sleep + CLK < 128kHz
LPM1	N/A		
LPM2	N/A		
LPM3	LPM3	Deepsleep (ARM) Stand-by w/ RAM & RTC	A/BCLK, <32kHz, some peripherals available
LPM4	LPM4	Stand-by with RAM	No clocks, some peripherals available
LPM3.5	LPM3.5	Shutdown	RTC w/o RAM
LPM4.5	LPM4.5	Shutdown	Shutdown

Texas Instruments

MSP430 & 432 Low-Power Modes

MSP430	MSP432	Available Peripherals
Active	Active	CPU & peripherals available
	Low-Frequency Active	CPU & peripherals available, CLK <128kHz
LPM0	LPM0	All peri. available, CPU off
	Low-Frequency LPM0	All peri. available, CPU off, CLK < 128kHz
LPM1	N/A	
LPM2	N/A	
LPM3	LPM3	BCLK, <=32kHz, RTC, WDT, GPIO
LPM4	LPM4	No clocks, only GPIO available
LPM3.5	LPM3.5	RTC w/o RAM
LPM4.5	LPM4.5	Shutdown

Power | MSP432 Flexible Operating Modes

MSP432
Current
CPU
FLASH
SRAM
LDO
DC-DC
PSS Bandgap
Clocks
Core Domain Logic
Backup Domain Logic (RTC/WDT)
I/O State

LPMx.5 modes
Ultra low leakage modes, with no state retention (as low as 670nA typ with RTC active)

LPM3.5: allows wake up from RTC & WDT events

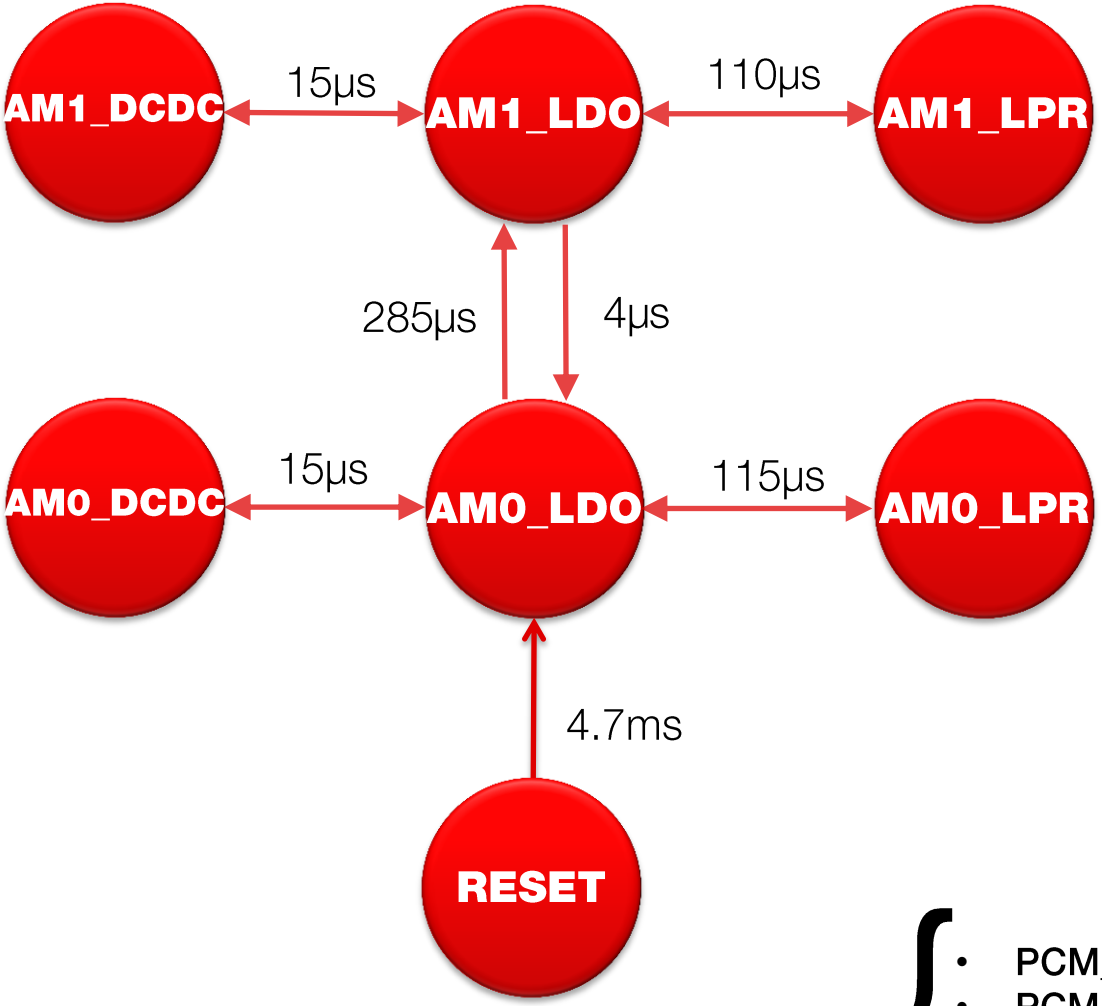
LPM4.5: allow reset wake up through reset

de : 0 – 24MHz
ugh LDO (for

e : 0 – 48MHz
ugh LDO or DCDC

TI Information – Selective Disclosure

Power | Active Mode Transitions



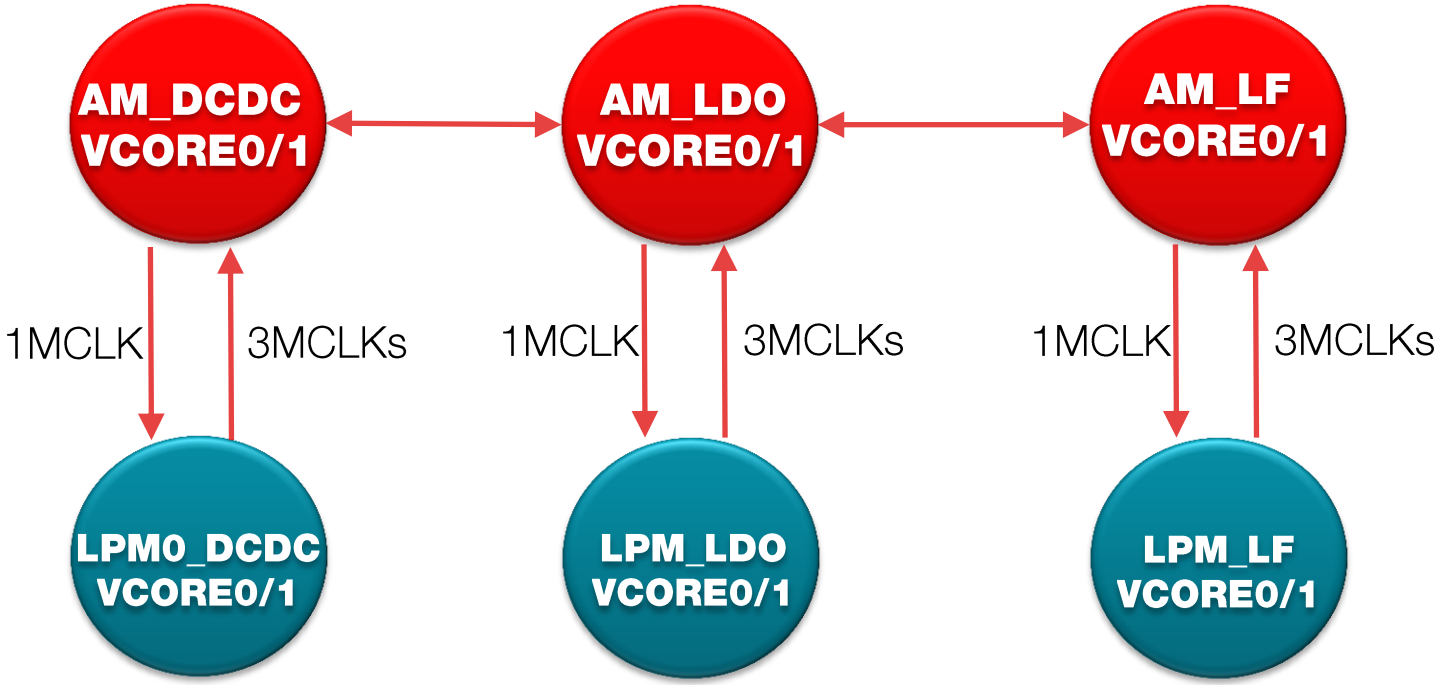
PCM_setPowerState(powerState), powerState=

- PCM_AM1_DCDC
- PCM_AM1_LDO
- PCM_AM0_LPR
- ...

Texas Instruments

Power | Sleep Mode Transitions

- VCore level (x) only change-able in AMx_LDO mode



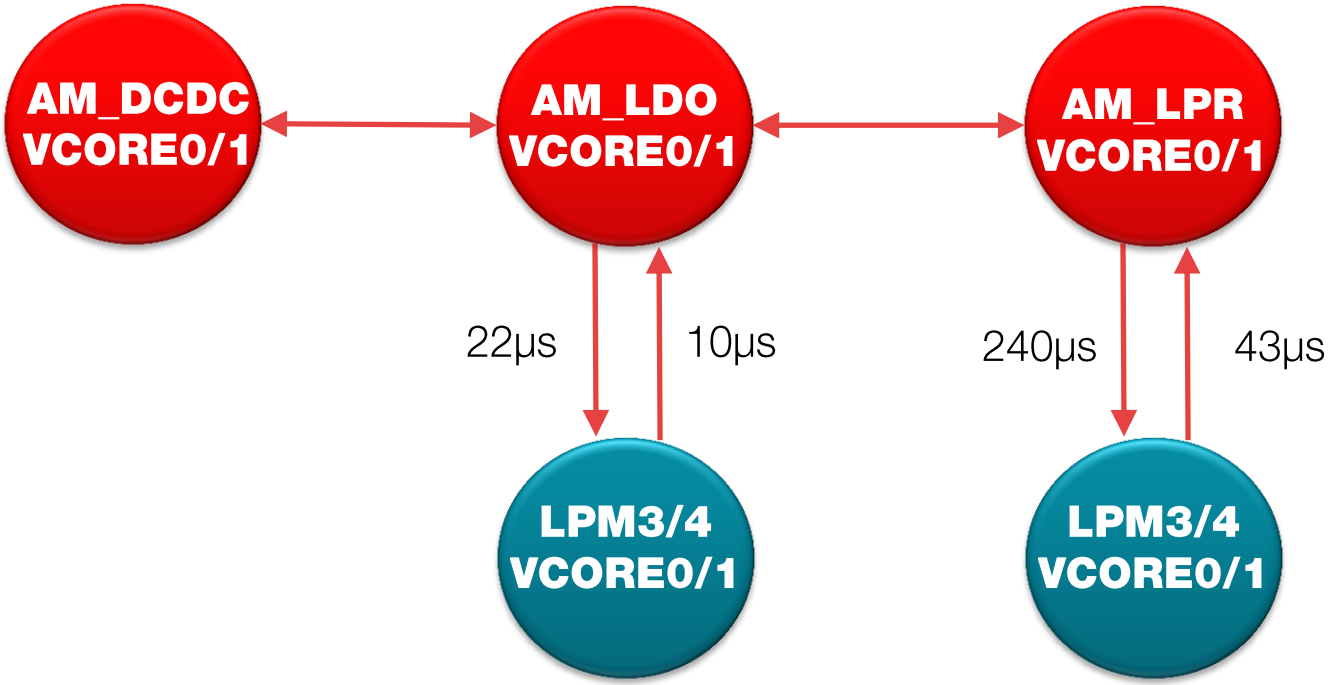
PCM_setPowerState(powerState)

- PCM_LPM0_DCDC_VCORE0
- PCM_LPM0_LDO_VCORE1
- PCM_LPM0_LF_VCORE0
- ...

Texas Instruments

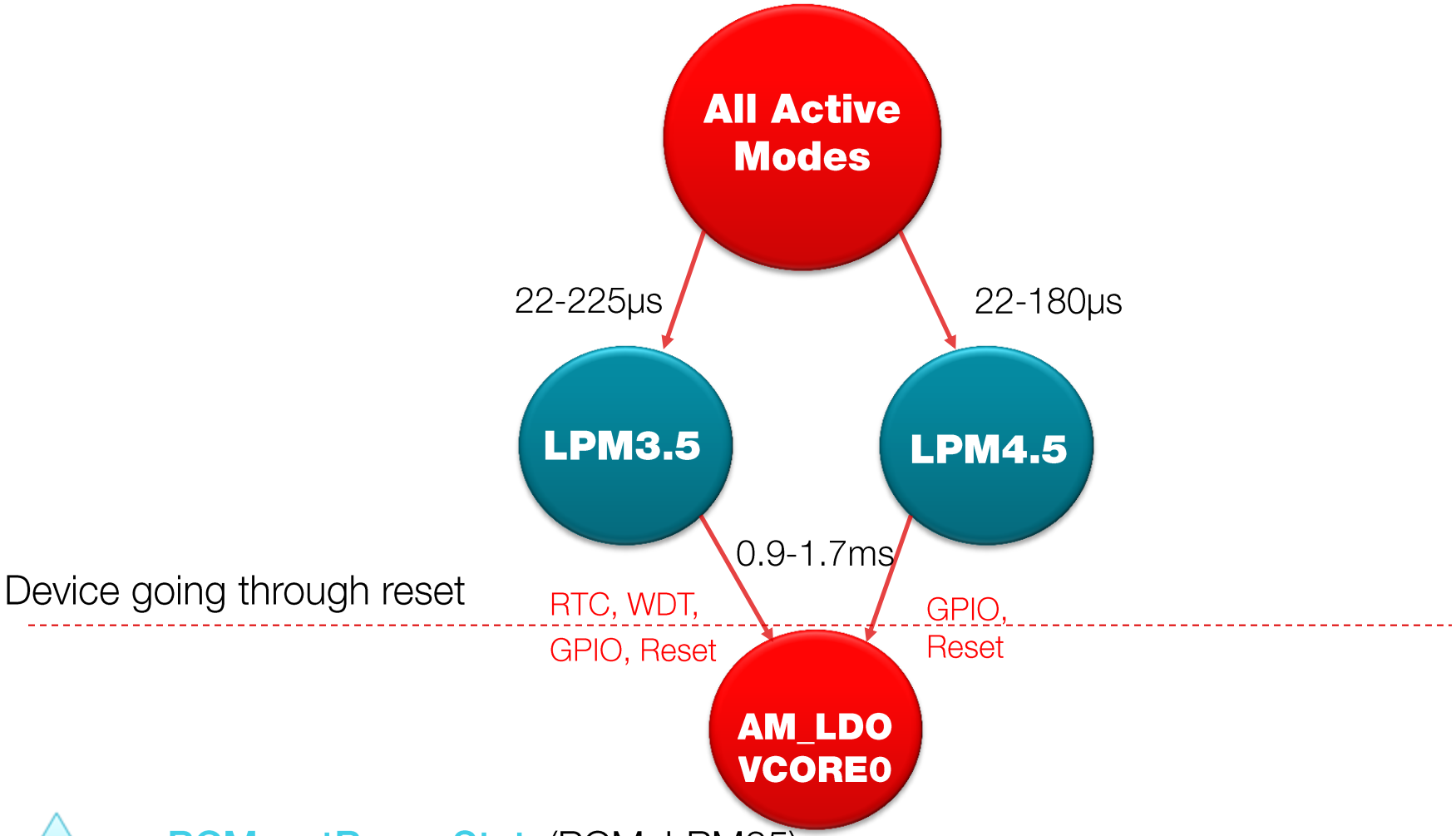
Power | LPM3/4 Transition

- VCore level (x) only change-able in AMx_LDO mode




```
PCM_setPowerState(PCM_LPM3);
PCM_gotoLPM3();
```

Power | LPMx.5 Modes




```
PCM_setPowerState(PCM_LPM35);
PCM_setPowerState(PCM_LPM45);
```

Power | Software & Intrinsic Support

- Blends with Cortex-M sleep & interrupt mechanisms

- Go to LPM0/Sleep using Cortex-M CMSIS instructions
  - __wfi()

- Go to LPM3/Deep-Sleep using CMSIS & MSP-DNA intrinsic
  - __deep_sleep()

- DriverLib calls:
  - PCM_setPowerState() to transitions between all states.
  - PCM_shutdownDevice(PCM_LPM45);
  - PCM_gotoLPM0();
  - PCM_gotoLPM3();
  - PSS_enableHighSide();
  - PSS_setHighSidePerformanceMode();

Texas Instruments

Power | ULP Recommendations

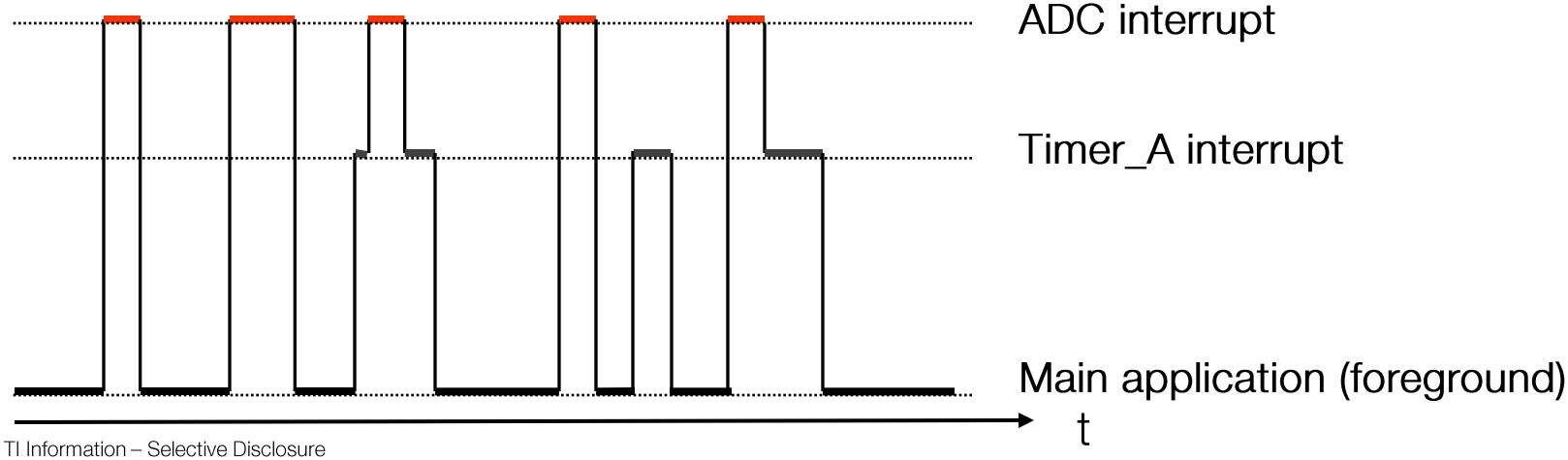
- Use DC-DC mode at higher frequencies
- Use VCORE=0 at lower frequencies
- Use Low-Frequency Modes (AM_LF_VCOREx or LPM0_LF_VCOREx) where applicable in application
- Use Supply Reference in sample mode for extremely ULP in LPM3/LPMx.5 modes

Texas Instruments

Cortex-M Interrupt
Nested Vector Interrupt Controller (NVIC)

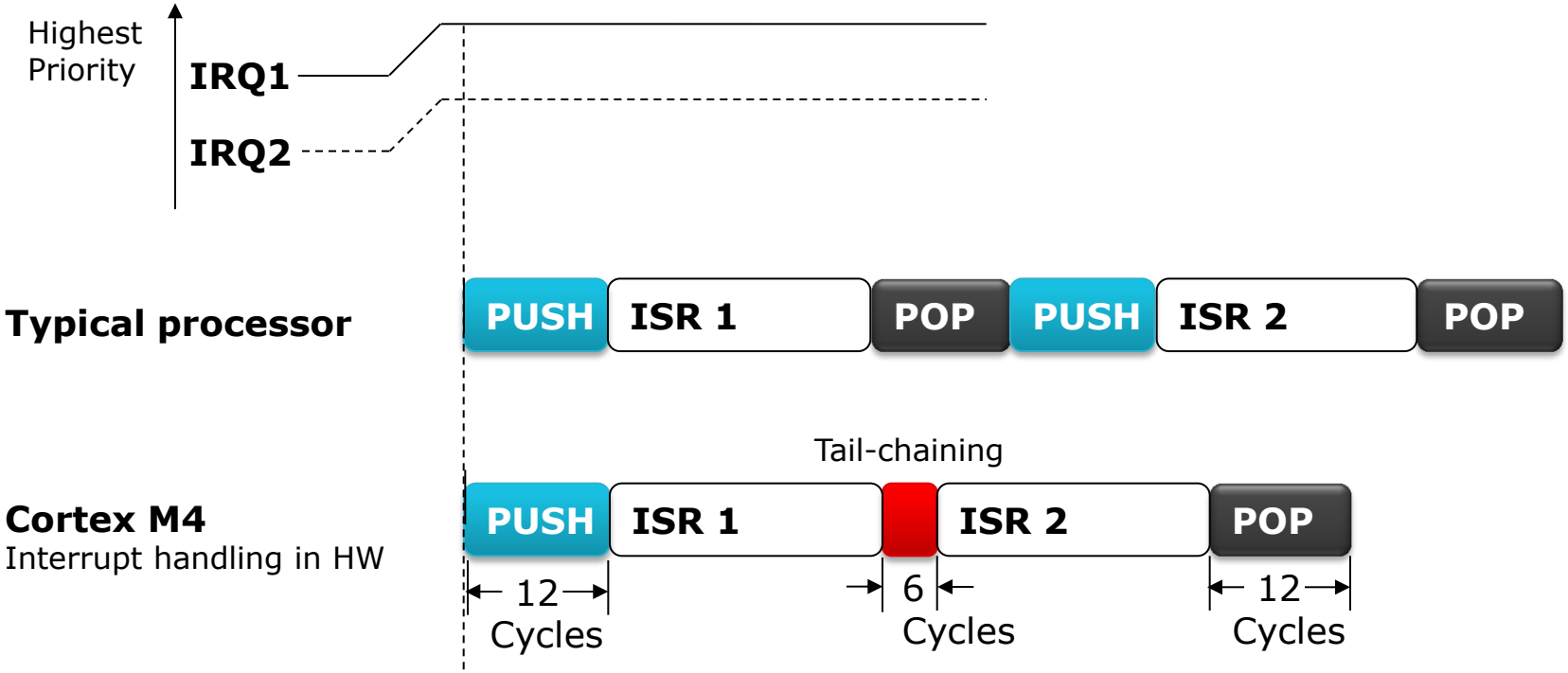
Interrupts | Nested Vectored Interrupt Controller (NVIC)

- 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



TI Information – Selective Disclosure

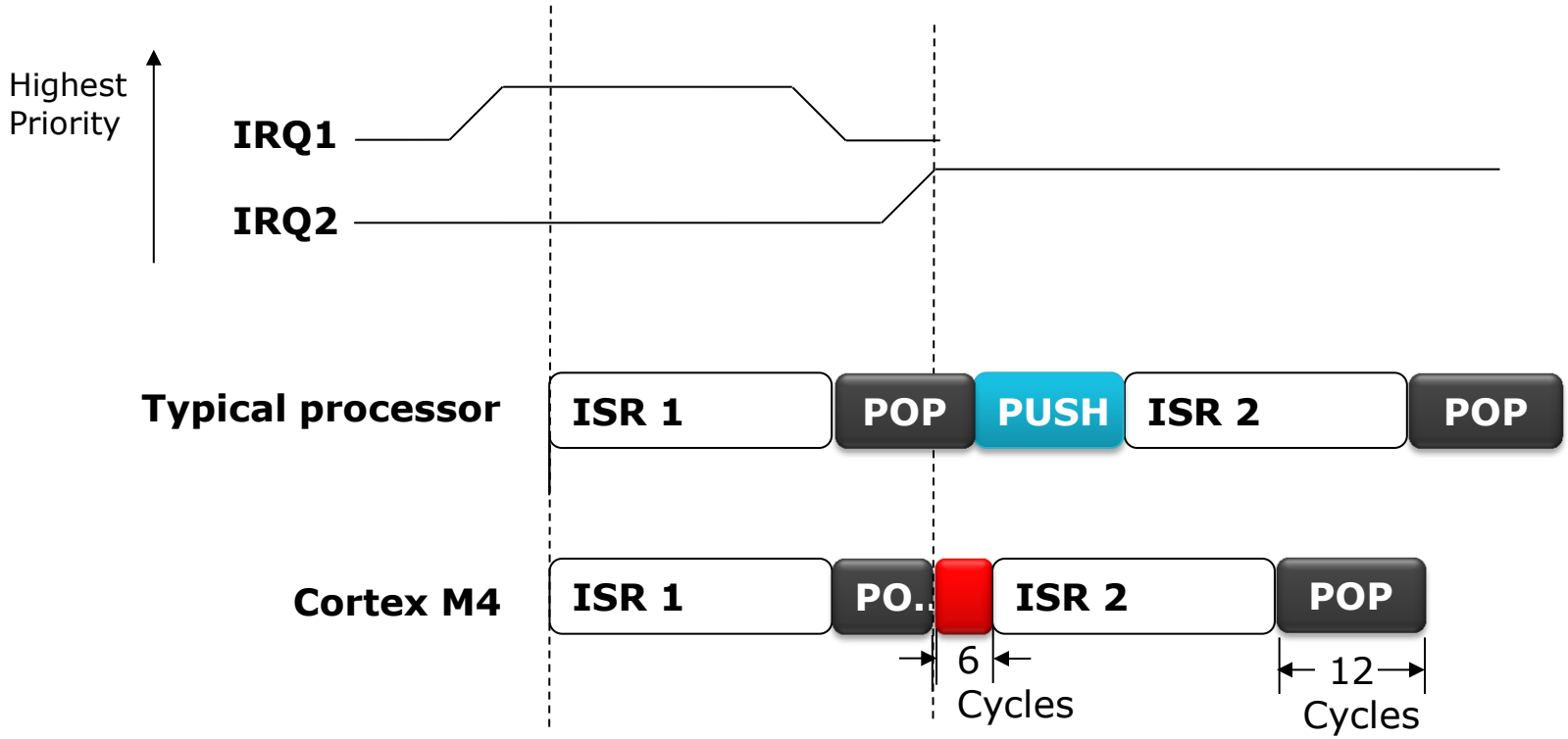
Interrupts | Latency: Tail Chaining



Saving **18** cycles:
24 cycles (POP + PUSH) → **6** cycles (Tail-chaining)

TI Information – Selective Disclosure

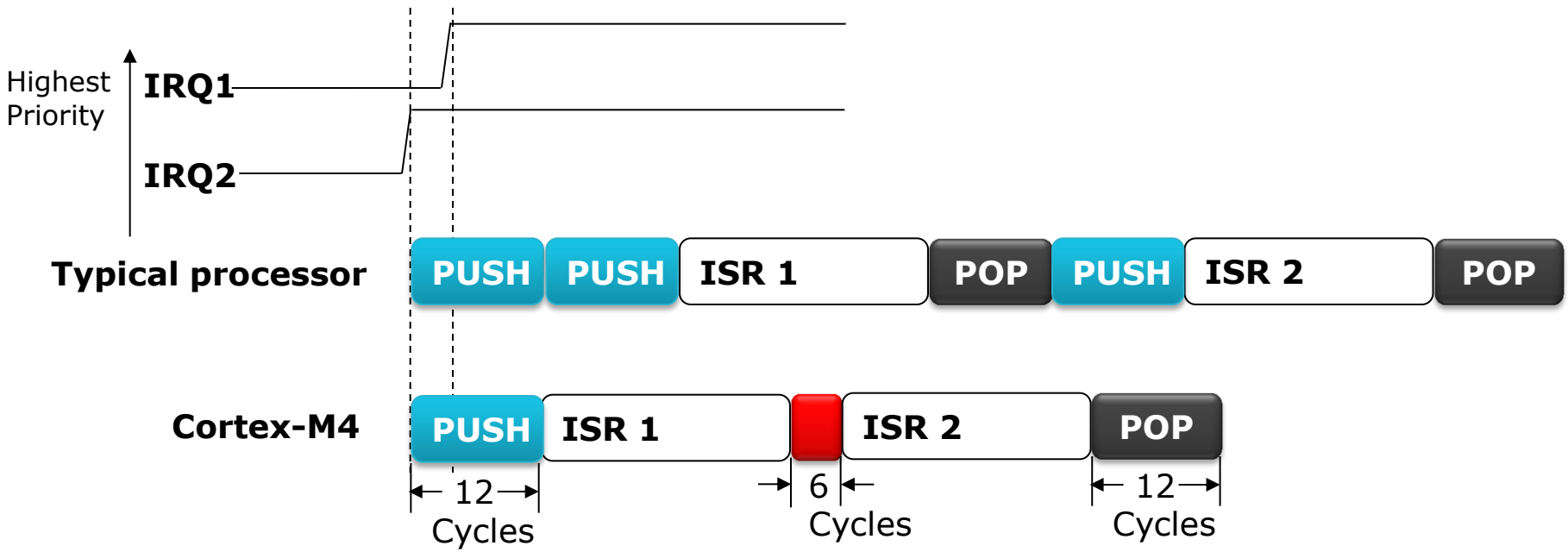
Interrupt | Latency: Late arrival



Saving from **7 to 17** cycles:

24 cycles (POP+PUSH) → **[1-11]** (pre-empted POP) + **6** cycles (Tail-chaining)

Interrupt | Latency: Pre-emption



Saving **18** cycles:
24 cycles (POP+PUSH) → **6** cycles (Tail-chaining)

TI Information – Selective Disclosure

Interrupts | Cortex-M4 Exception Types

Vector Number	Exception Type	Priority	Vector address	Descriptions
1	Reset	-3	0x04	Reset
2	NMI	-2	0x08	Non-Maskable Interrupt
3	Hard Fault	-1	0x0C	Error during exception processing
4	Memory Management Fault	Programmable	0x10	MPU violation
5	Bus Fault	Programmable	0x14	Bus error (Prefetch or data abort)
6	Usage Fault	Programmable	0x18	Exceptions due to program errors
7-10	Reserved	-	0x1C - 0x28	
11	SVCall	Programmable	0x2C	SVC instruction
12	Debug Monitor	Programmable	0x30	Exception for debug
13	Reserved	-	0x34	
14	PendSV	Programmable	0x38	
15	SysTick	Programmable	0x3C	System Tick Timer
16 and above	Interrupts	Programmable	0x40	External interrupts (Peripherals)

TI Information – Selective Disclosure

Interrupts | Declaration on MSP432

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                   */
    intDefaultHandler,                  /* 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               */
```

msp432_startup_ccs.c

ISR Handlers: stubs defined in table.
Treated as regular function, code in user's application

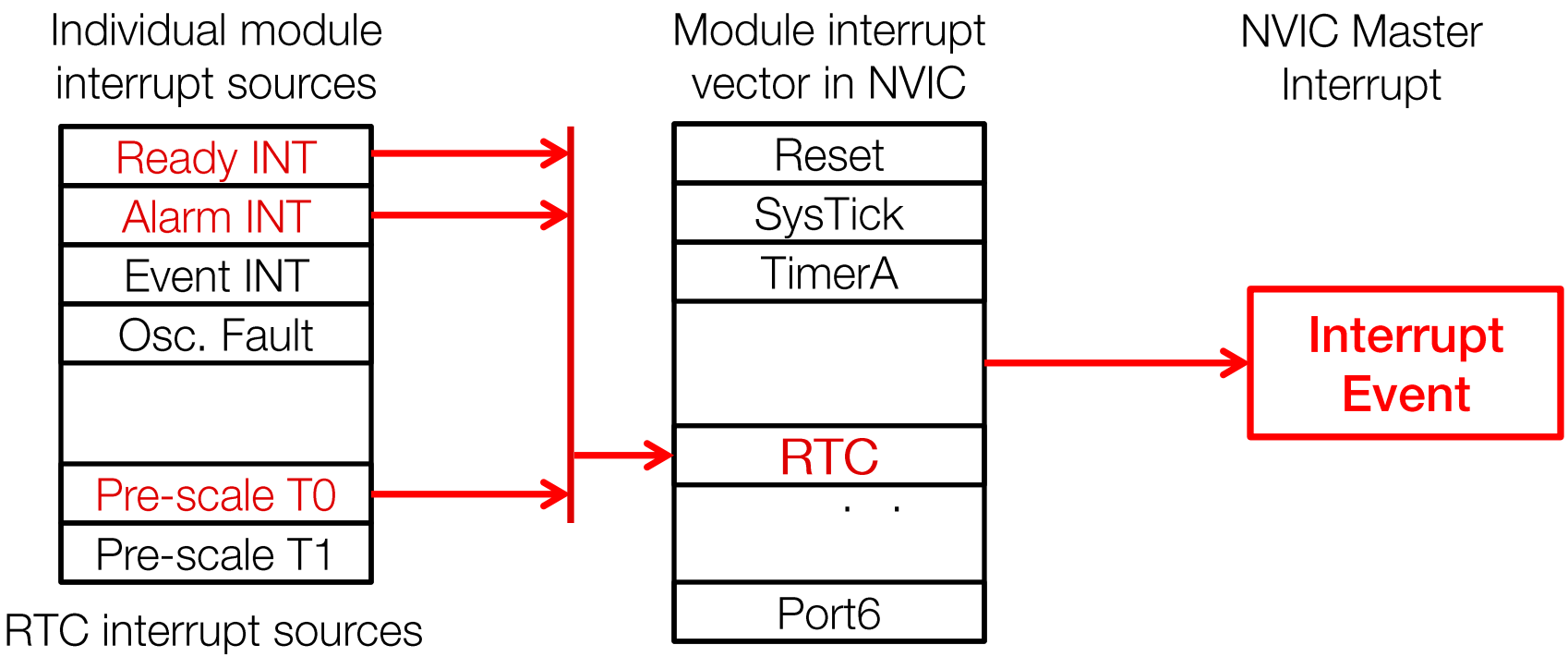
Option 2: MSP430 method
Use **#pragma vector**

```
#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
}
```

1. MSP430 code re-use
2. Interrupt vector & handler function defined together

Interrupts | Usage on MSP432



1. Enable module interrupt sources

```
RTC_enableInterrupt(
  RTC_CLOCK_READ_READY_INTERRUPT |
  RTC_CLOCK_ALARM_INTERRUPT |
  RTC_PRESCALE_TIMER1_INTERRUPT);
```

2. Enable module interrupt in NVIC

```
Interrupt_enableInterrupt(
INT_RTC);
```

3. Enable NVIC Master Interrupt

```
Interrupt_enableMaster();
```

Lab1: Power Modes [+NVIC]

Texas Instruments

Lab 1 | Power

Lab Goals

- 1st look at MSPWare featuring MSP432 DriverLib & Training
- Evaluate different MSP432 power states
- MSP432 DriverLib API to change power state
- First look at MSP432 Interrupt usage

Instructions

- Follow lab manual to complete & build project
- Use push button to cycle through power states
  - Indicated by different LED color & blink counts
- Measure current consumption with EnergyTrace

Hint

- powerStates[curPowerState] cycles through 13 power state definitions

```
PCM_AM_LDO_VCORE0
PCM_AM_LDO_VCORE1
PCM_AM_DCDC_VCORE0
PCM_AM_DCDC_VCORE1
PCM_AM_LF_VCORE0
PCM_AM_LF_VCORE1
PCM_LPM0_LDO_VCORE0
PCM_LPM0_LDO_VCORE1
PCM_LPM0_DCDC_VCORE0
PCM_LPM0_DCDC_VCORE1
PCM_LPM0_LF_VCORE0
PCM_LPM0_LF_VCORE1
PCM_LPM3
```


Texas Instruments

Clock System (CS)

TI Information – Selective Disclosure

CS | High-level Features

- 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

TI Information – Selective Disclosure

CS | HF & LF Oscillators

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

TI Information – Selective Disclosure

CS | Oscillator usage

Frequency		Oscillators	Accuracy	Power	Applications
High	1-48MHz	DCO	2.7% internal R 0.4% external R	High	High-speed applications that require medium-low accuracy HF
	1-48MHz	HFXT	High	High	Applications with high-accuracy high-speed communication or timing requirements
	25MHz	MODOSC	+/- 2MHz	High	Internal osc. option for peripherals such as ADC
	5MHz	SYSOSC	• +/-0.75MHz	Medium	Internal, direct clock for ADC failsafe for HFXT
Low	32kHz	LFXT	High	Medium	RTC, time-keeping requirements
	32kHz 128kHz	REFO	+/-3%	Low	ULP with lax requirements for accurate time-keeping, drift over time/temp acceptable
	10kHz	VLO	+/- 4kHz	Low	ULP with no requirements for time-keeping, interval/periodic clock source needed.

TI Information – Selective Disclosure

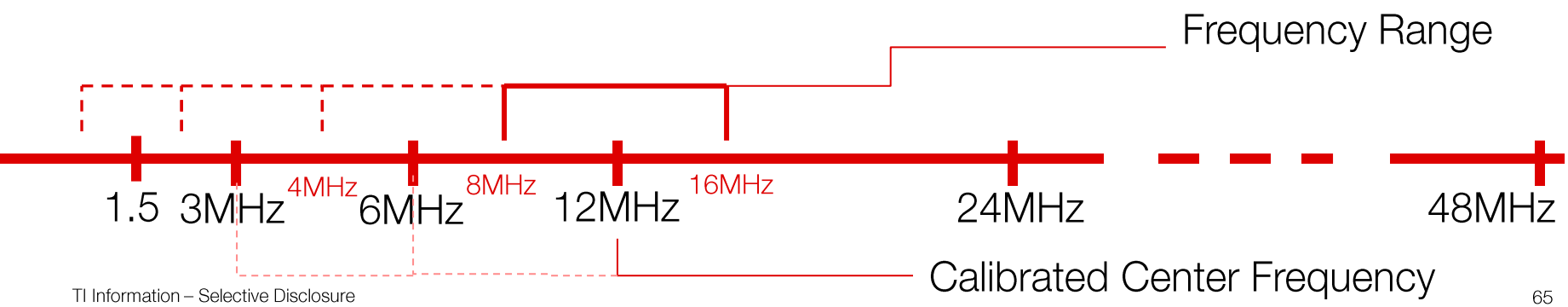
CS | Clock availability in power modes

Power Modes	MCLK	SMCLK	HSMCLK	ACLK	BCLK	Comments
Active	✔	✔	✔	✔	✔	
Low-Frequency Modes	✔ <128kHz	✔ <128kHz	✔ <128kHz	✔	✔	ACLK, BCLK always <128kHz
LPM0	✘	✔	✔	✔	✔	
LPM3	✘	✘	✘	✘	✔	
LPM3.5	✘	✘	✘	✘	✔	
LPM4.5	✘	✘	✘	✘	✘	

TI Information – Selective Disclosure

CS | High-accuracy tune-able DCO

- 6 tune-able frequency ranges
  - Each range has calibrated center frequency
  - Example: [8-16MHz] range has a calibrated 12MHz center frequency
- Tune-able within each frequency range
  - Center Frequency +/- 2^{12} steps → DCOTUNE register
- DCO accuracy:
  - Internal resistor: ± 2.65 % [Calibrated]
  - External resistor : ± 0.4 % [91kΩ ± 0.1%]
    - Failsafe for internal resistor mode



CS | High-frequency System Design

- DC-DC or LDO
  - DC-DC yields higher efficiency than LDO at higher speed
  - DC-DC requires longer start-up time and transitions from Sleep Modes
- Flash wait states
  - @ MCLK > 12MHz for VCORE = 0
  - @ MCLK > 16MHz for VCORE = 1

SystemFrequency	VCORE	LDO/DC-DC*	Flash Wait States
0-12MHz	0	LDO	0
12-16MHz	0	LDO	1
16-24MHz	0	LDO	1
24-32MHz	1	DC-DC	1
32-48MHz	1	DC-DC	2

CS | DriverLib & Software Usage

- DriverLib Usage highly recommended

- DriverLib call examples:
  - CS_setDCOFrequencyRange() – set DCO to a calibrated frequency
  - CS_setDCOFrequency() – tune DCO closest to any frequency
  - CS_startHFXT() – start high-frequency crystal (normal/bypass mode)
  - CS_startLFXT() – start low-frequency crystal (normal/bypass mode)
  - CS_initClockSignal() – initialize clock signal sources
  - CS_getACLK() – returns ACLK frequency (in Hz)
  - CS_getMCLK() – returns MCLK frequency (in Hz)

Texas Instruments

Lab 2: Clock System

Texas Instruments

Lab 2 | Clock System

Lab Goals

- Configure device's clock & operations at different frequency
- Exercise MSP432 DriverLib API to configure clocks
- Notice the inter-peripheral dependencies of core modules at high-speed

Instructions

- Follow lab manual to complete & build project
- Use push button to cycle device through different speed
- LED blink speeds indicate relative speed change
- [Optional] Measure MCLK output on port pin with oscilloscope

Hint

- frequencyCycle[curFrequency] cycles through 6 DCO frequencies.

Texas Instruments

Memory

Memory | Overview

432

Memory	Size	Speed	Features	
Flash	256kB + 4kB Sector: 4kB	16MHz	Speed boost with 128-bit buffer & pre-fetch	Powerful security features
SRAM	64kB Bank: 8kB	48MHz	Dynamic bank power-down & retention options for low power	
ROM	32kB	48MHz	Robust DriverLib APIs integrated to save application space	Lower power execution
BSL	8kB	16MHz	UART/I2C/SPI Boot-Strap Loader provided	

*Possible change: programmable BSL in next devices/revisions

Texas Instruments

Memory | Flash Features

432

- **Independent banks** simultaneous read/execute and program/erase operations
- **128-bit buffer** Power savings & higher effective speed with ARM's pre-fetch
- **Hardware assisted operations**
  - Burst data comparison for fixed patterns (data fill check)
  - Flash program modes with auto-computed parity & auto-verify:
    - Write immediate, 128-bit full word write, or 4*128-bit burst mode
- **Flexible Code security & protection options:**
  - Individual Flash IP sector protection, further secured/protected with MPU
  - Up to 4 IP-protection memory zones

Texas Instruments

Memory | RAM

432

- Up to 64KB of banked SRAM architecture
- 8 dynamically configurable banks:
  - Enable/disable banks to optimize active mode power consumption
  - Retain/not retain content in DSL to optimize sleep modes' leakage power consumption

SRAM banks	Memory size
Bank 0 enable/retention (always enabled)	8KB
Bank 1 enable/retention	16KB
Bank 2 enable/retention	24KB
. . .	…
Bank 6 enable/retention	56KB
Bank 7 enable/retention	64KB

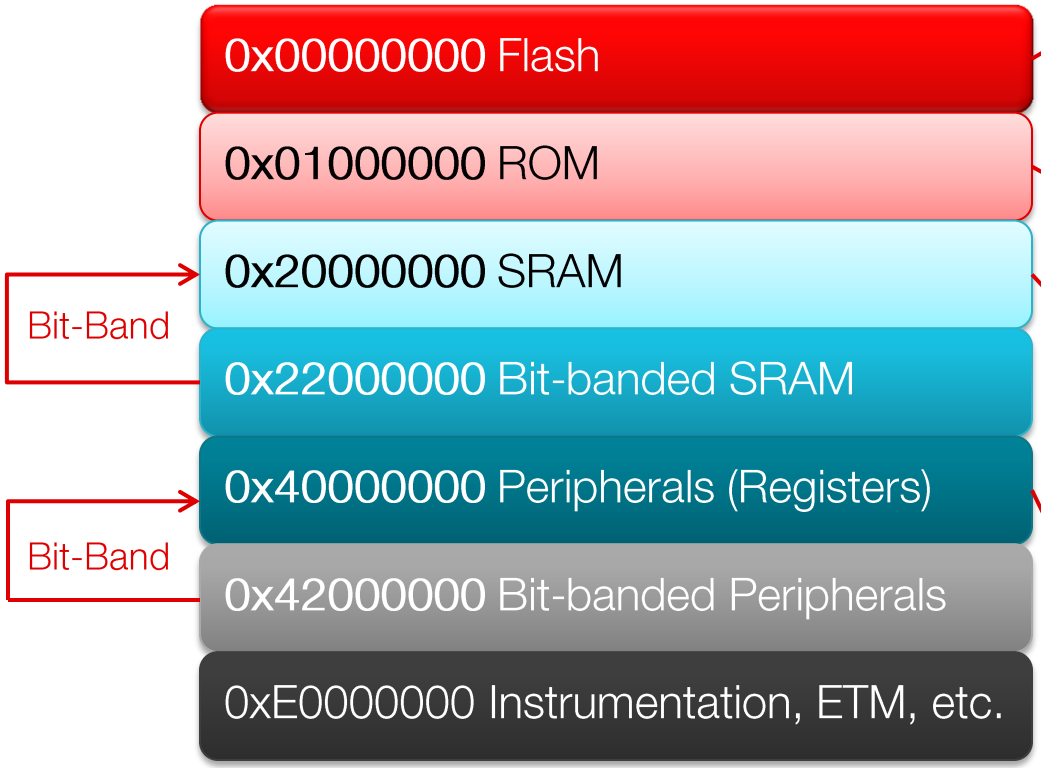
TI Information – Selective Disclosure

Memory | Flash & SRAM DriverLib API

- Flash is a complex module, DriverLib API usage highly recommended
- DriverLib API examples:
  - FlashCtl_protectSector()
  - FlashCtl_isSectorProtected()
  - FlashCtl_eraseSector()
  - FlashCtl_performMassErase()
  - FlashCtl_programMemory() – program flash memory with data buffer
  - FlashCtl_setWaitState() – required for flash operation @ CPU > 16MHz
- SRAM API, part of SysCtl group:
  - SysCtl_enableSRAMBank(); – enable SRAM bank(s)
  - SysCtl_disableSRAMBank(); – disable SRAM bank(s)
  - SysCtl_enableSRAMBankRetention(); – retain SRAM content in DSL
  - SysCtl_disableSRAMBankRetention(); – does not retain content in DSL (lower power consumption)

Texas Instruments

Memory | Memory Map



256kB + 4kB
- Interrupt Vector Table
- Application Code

Peripheral Driver Library

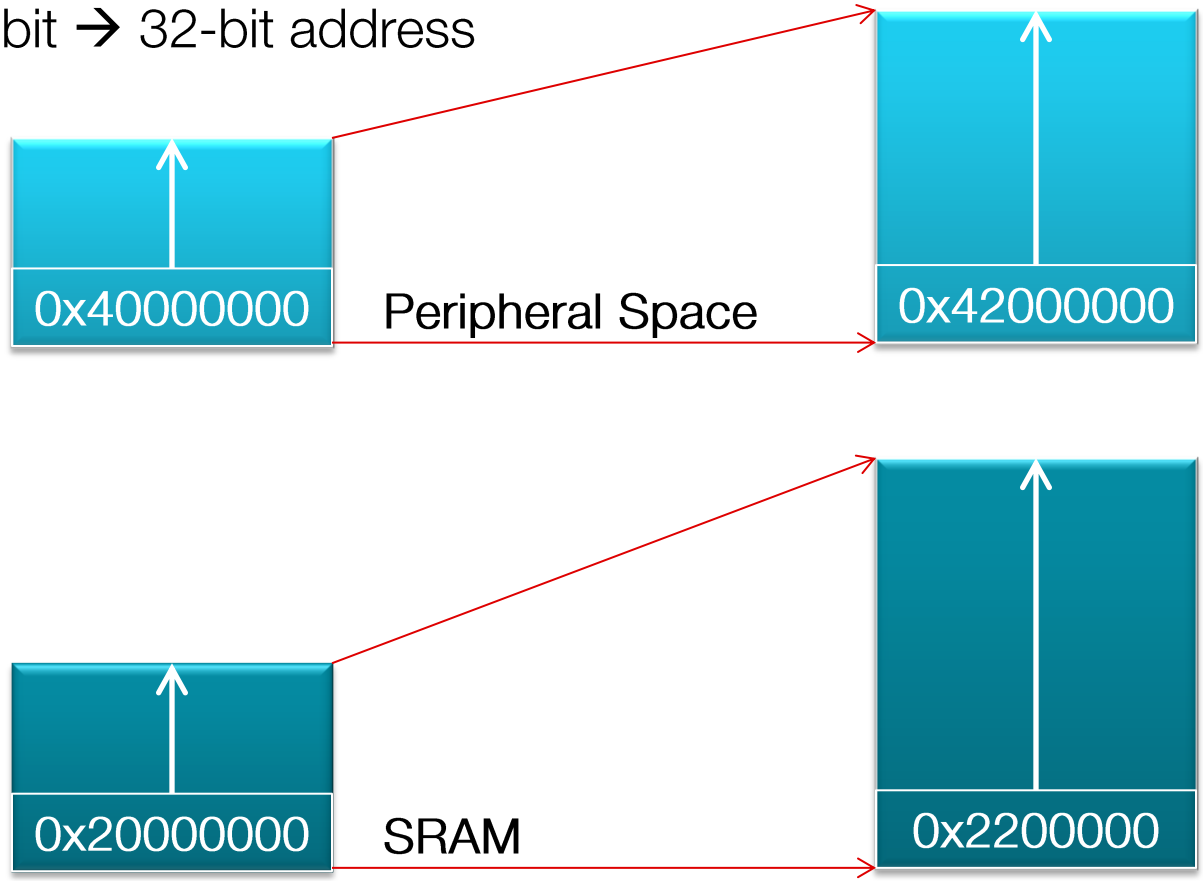
Ultra-low-leakage SRAM
- 64kB = 8 x 8kB banks
- Bit-banded

Peripheral Space
- Register directly accessible
- Bit-banded

TI Information – Selective Disclosure

Memory | Bit-Banding

- Reduces the number of read-modify-write operations
- Address aliases to access individual bits in a single, atomic operation
- 1 bit → 32-bit address



Peripheral Bit-Band

SRAM Bit-Band

Memory | Bit-Banding Advantage

P1DIR |= BIT0;

LDR	R1, #40004C04	6-byte
LDRB	R0, [R1]	2-byte
ORR.W	R0, R0, #1	4-byte
STRB	R0, [R1]	2-byte
		14 bytes in total

BITBAND_PERI(**P1DIR**, 0) = 1;

LDR	R1, #42098080	6-byte
MOVS	R0, #1	2-byte
STRB	R0, [R1]	2-byte
		10 bytes in total

BIT0 of P1DIR register @0x40004C04, is mapped to 32-bit word at 0x42098080

STRB Store instruction instead Read-Modify-Write sequence.

Texas Instruments

Memory | Bit-Banding Usage

Calculation

```
bit-band alias =
bit-band base + (byte offset * 0x20) + (bit number * 4)
```

For example, bit-7 at address 0x20002000 is:

```
0x22000000 + (0x2000 * 0x20) + (7 * 4) = 0x2204001C
```

Use header definitions:

- `#define HWREGBIT8(x, b)`
- `#define HWREGBIT16(x, b)`
- `#define HWREGBIT32(x, b)`
- `#define BITBAND_SRAM(x, b)`
- `#define BITBAND_PERI(x, b)`

DriverLib extensively uses bit-banding for all applicable peripheral register accesses

TI Information – Selective Disclosure

Memory | Memory Protection Unit (MPU)

- 8 memory regions
  - plus a background region accessible from privileged mode
- Region size >= 256 bytes
- Each region divided into 8 equal-sized sub-regions
- MPU definitions for all regions include:
  - Location
  - Size
  - Access permissions: Read, Write, eXecute
  - Memory attributes
- Accessing a prohibited region causes a memory management fault
  - Fault can cause Reset or be handled in an ISR

TI Information – Selective Disclosure

Lab 3: Memory

Lab 3 | Memory System

Lab Goals

- Exercise various Flash & SRAM DriverLib APIs
- Notice the Flash protection, erase, and program sequence.
- Examine the customized linker command file, notice SRAM & STACK relocation and consideration for SRAM enhancement feature.

Instructions

- Follow lab manual to complete & build project
- Test 1: Flash
  - exercise Flash protection, erase, and program APIs.
  - Set breakpoints on lines as indicated by lab manual to observe Flash operations
- Test 2: SRAM
  - Complete SRAM bank disable/enable code, use breakpoints to observe the results

Digital I/O

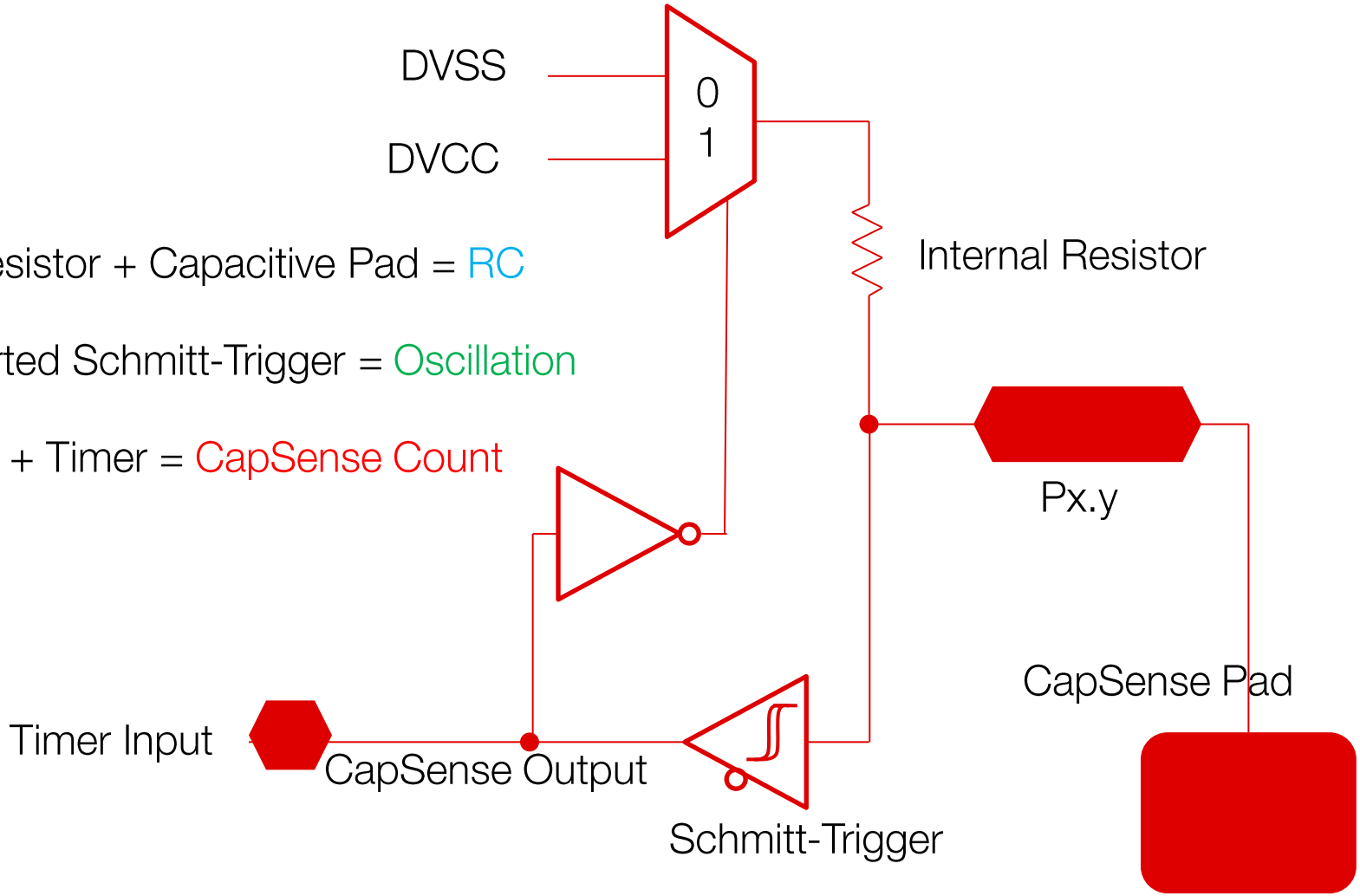
Digital I/O | Overview

- Ultra low-leakage IOs (± 20 nA Max)
- Capacitive touch function available on all Ios
- Pull-up/Pull-down
- Interrupt/Wake-up capability available on 6 ports (48 pins)
- Port Map: 30 digital functions available to each of 24 port-mappable IOs
- Up to 4 pins with 20mA max output, suitable for optional high-drive functionality (P2.0-P2.3)
- 8 IOs with glitch filtering capability

432

Digital I/O | Capacitive Touch

1. Internal Resistor + Capacitive Pad = RC
2. RC + inverted Schmitt-Trigger = Oscillation
3. Oscillation + Timer = CapSense Count



Texas Instruments

Digital I/O | Capacitive Touch

Benefits

- Low cost
- No external components
- Simple solution

Performance

- Sense caps can be in the range of 2pF to 50pF
- Low power (examples w/ <10uA, 1uA)

Features

- Used with Timer_A to measure the oscillation frequency
- The control registers allow scanning to adjacent pins easily
- Dedicated cap sense I/O registers

Digital I/O | Glitch Filtering

432

- Glitch suppress filter available on some interrupt/wake-up I/Os
- The analog filter suppresses < 250ns-wide glitches.
- Advantageous for ESD-prone applications (e-metering, hand-held, etc.)
- The glitch filter is enabled by default.
- Flexibility to bypass filter by:
  - Clear GLTFLT_EN bit in SYS_DIOGLTFLTCTL_REG
  - When I/O is configured for peripheral or analog functionality

Texas Instruments

Digital I/O | DriverLib APIs

- Inherits majority APIs from MSP430 →portable to MSP432

- Example:
  - GPIO_setAsOutputPin(); Configure output pin
  - GPIO_setAsInputPin(); Configure input pin
  - GPIO_setAsPeripheralModuleFunctionOutputPin();
  - GPIO_setOutputHighOnPin();
  - GPIO_getInputPinValue();
  - GPIO_enableInterrupt();
- New APIs:
  - GPIO_setDriveStrength(); set drive strength [only] for high-drive IO

Texas Instruments

Timers

Timers | Overview

Timer	Instances	# of Bits	Lowest Operating mode	Interval	PWM	Main/Other Function
Timer_A	4	16	LPM0, LPM0_LF	✔✔	✔	Capture, Compare
Timer32	2	32	LPM0, LPM0_LF	✔✔		
SysTick	1	24	LPM0, LPM0_LF	✔✔		
RTC	1	32*	LPM3, LPM3.5	✔		RTC
WDT	1		LPM3, LPM3.5	✔		WDT

✔ RTC & WDT: only have a few selection of pre-set intervals

✔✔ Timer_A, Timer32, SysTick: configurable intervals with 16/32/24-bit granularity

Timers | 16-bit Timer_A

General-purpose timer/interval mode

- Interval, periodic timer
- 16-bit resolution/granularity

Up to seven configurable capture/compare registers

- Six output modes
- Configurable outputs with pulse width modulation (PWM) capability
- Capture mode to capture external events, measure durations

Autonomous trigger sources for other modules

- Periodic ADC conversion, no CPU intervention required
- Triggers DMA transfer

Asynchronous input and output latching

Fully interruptible: overflow & CCRx interrupts

Timers | 32-bit General Purpose Time

Standard 32-bit ARM dual-timer

- Count-down timer, interrupt/flag when counter gets to 0
- Can be used in 16-bit or 32-bit mode

Flexible configuration

- Configurable 32-bit period
- Next period count can be loaded in background
- Single interrupt event

Timers | SysTick

ARM-standard system timer

- Uses MCLK source
- Can be used as system timer/OS timer

Flexible configuration

- Configurable 32-bit period
- Single interrupt event

Timers | Real-Time Clock Calendar & Alarm

- Calendar mode only
- LFXT1 32768Hz required
- Advanced interrupt capability – alarms, OF fault, RTCREADY and RTCEV
- Selectable BCD format
- Calibration
- Multiple Alarms
- Operation in SD0

Timers | DriverLib APIs

- RTC
  - RTC_initCalendar()
  - RTC_setCalendarAlarm ()
  - RTC_definePrescaleEvent ()
  - RTC_startClock ()
- Timer_A
  - TimerA_startCounter ()
  - TimerA_configureContinuousMode ()
  - TimerA_configureUpDownMode ()
  - TimerA_initCapture ()
  - TimerA_generatePWM ()
  - TimerA_setCompareValue()
  - TimerA_getCaptureCompareCount()
  - TimerA_clearTimer()
- SysTick_
  - SysTick_enableModule()
  - SysTick_setPeriod()
  - SysTick_getValue()
- Timer32
  - Timer32_setCount()
  - Timer32_getValue()
  - Timer32_startTimer()
- WDT
  - WDT_holdTimer()
  - WDT_initWatchdogTimer()
  - WDT_initIntervalTimer()

Texas Instruments

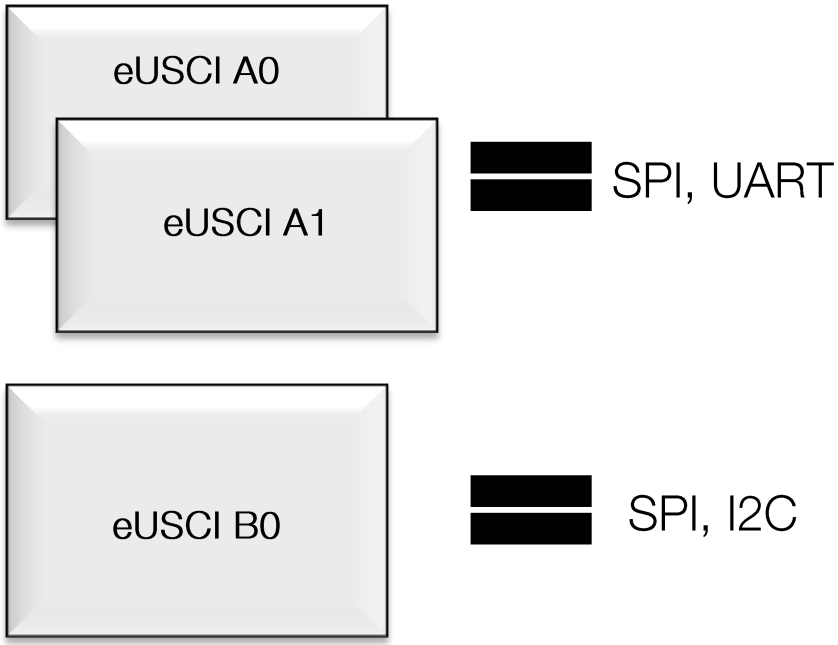
Timers | ULP Features

- Use RTC & WDT for timer functionality in DSL
- Replace software delays & __delay_cycles() with Timer interval interrupt & ULP mode usage
- Use Low-power clocks to source Timers in LPM modes.

Serial Communication (eUSCI)

eUSCI | Serial Communication

- I²C
  - 400kHz & up to 1MHz
  - 4 hardware slave addresses supported with dedicated TX/RX interrupts
  - Byte counter for autonomous operation
- UART
  - Enhanced baud rate generation
  - Auto baud rate detection
- SPI
  - 3 & 4-wire modes
  - Up to 14MBps



TI Information – Selective Disclosure

eUSCI_A | UART

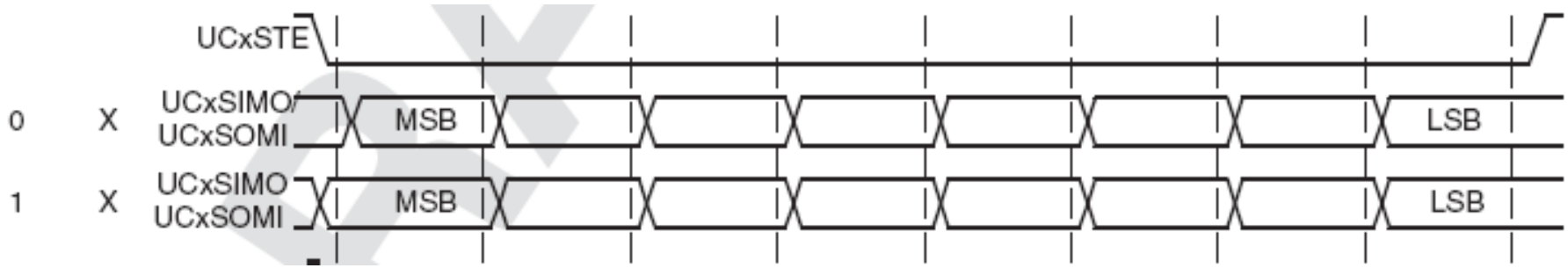
- Architecture is maintained mostly compatible with USCI_A
- Register mapping from USCI to eUSCI available in migration document
- New features include
  - UCTXCPTIE interrupt similar to TXEPT flag in USART
  - Enhanced baud rate calculator: Increased flexibility with modulation pattern settings
  - UCSTTIE interrupt for start bit detection
  - Increased flexibility with deglitch filter

fractional portion of N	UCBRSx setting[1]
0.0000	0x00
0.0529	0x01
0.0715	0x02
0.0835	0x04
0.1001	0x08
0.1252	0x10
0.1430	0x20

Texas Instruments

eUSCI_A/B | SPI

- Architecture is maintained mostly compatible with USCI_A
- Register mapping from USCI to eUSCI available in migration document
- Supports higher baud rates
  - Up to 14MHz @ 3.0V
  - Up to 6MHz @ 2.0V
- Modified 4-pin SPI mode
  - Can now be used as a 'true' chip select in master mode



eUSCI_B | I2C

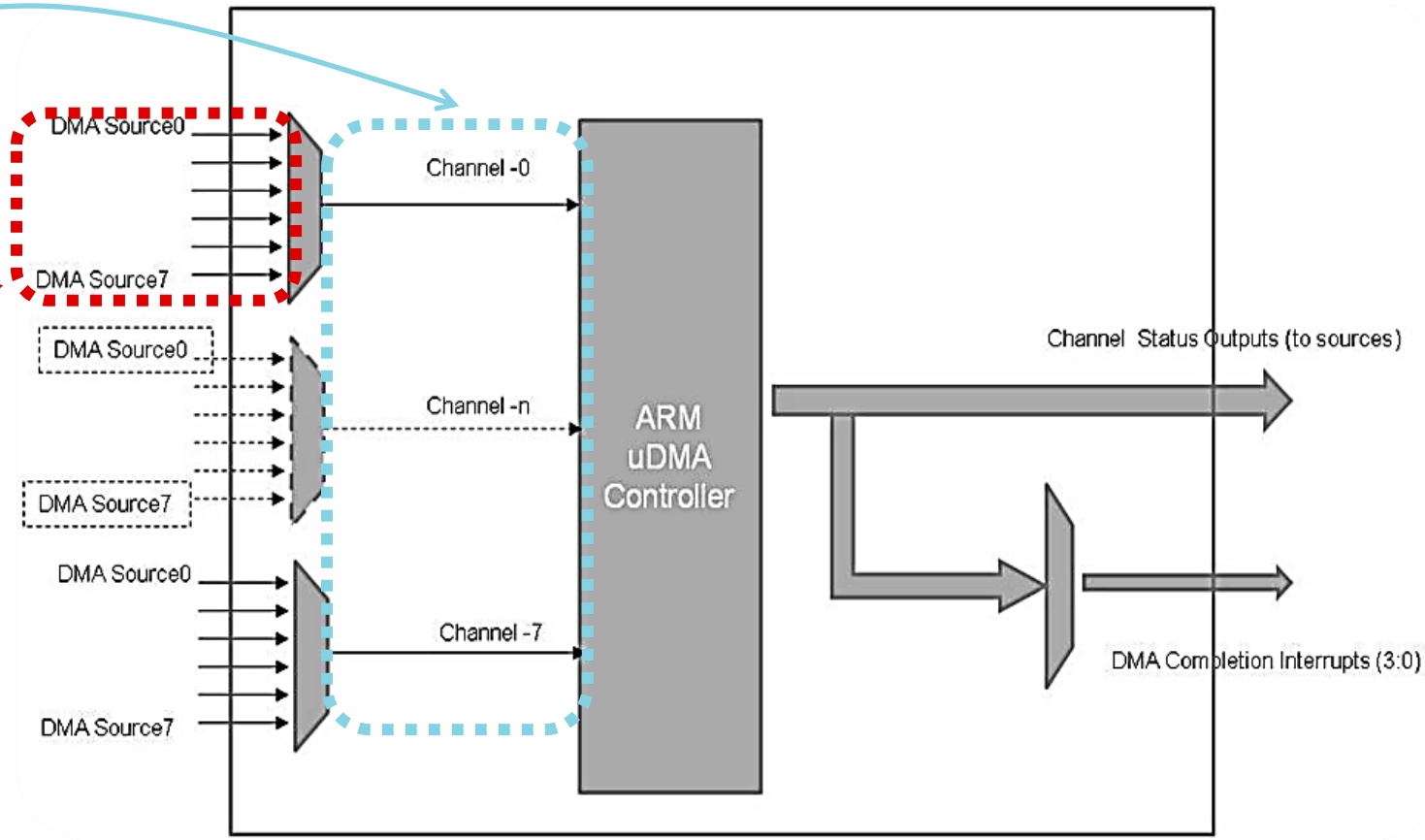
Many new features have been added:

- Multiple slave addresses
- Clock low timeout for SMBus compatibility
- Byte counter
- Automatic stop assertion
- Preload for master/slave transmitter
- Address bit masking
- Selectable deglitch timing
- Address ACK/NACK selectable in software

DMA

DMA | Controller

8 channels
with up to
8 trigger
sources
for each
channel



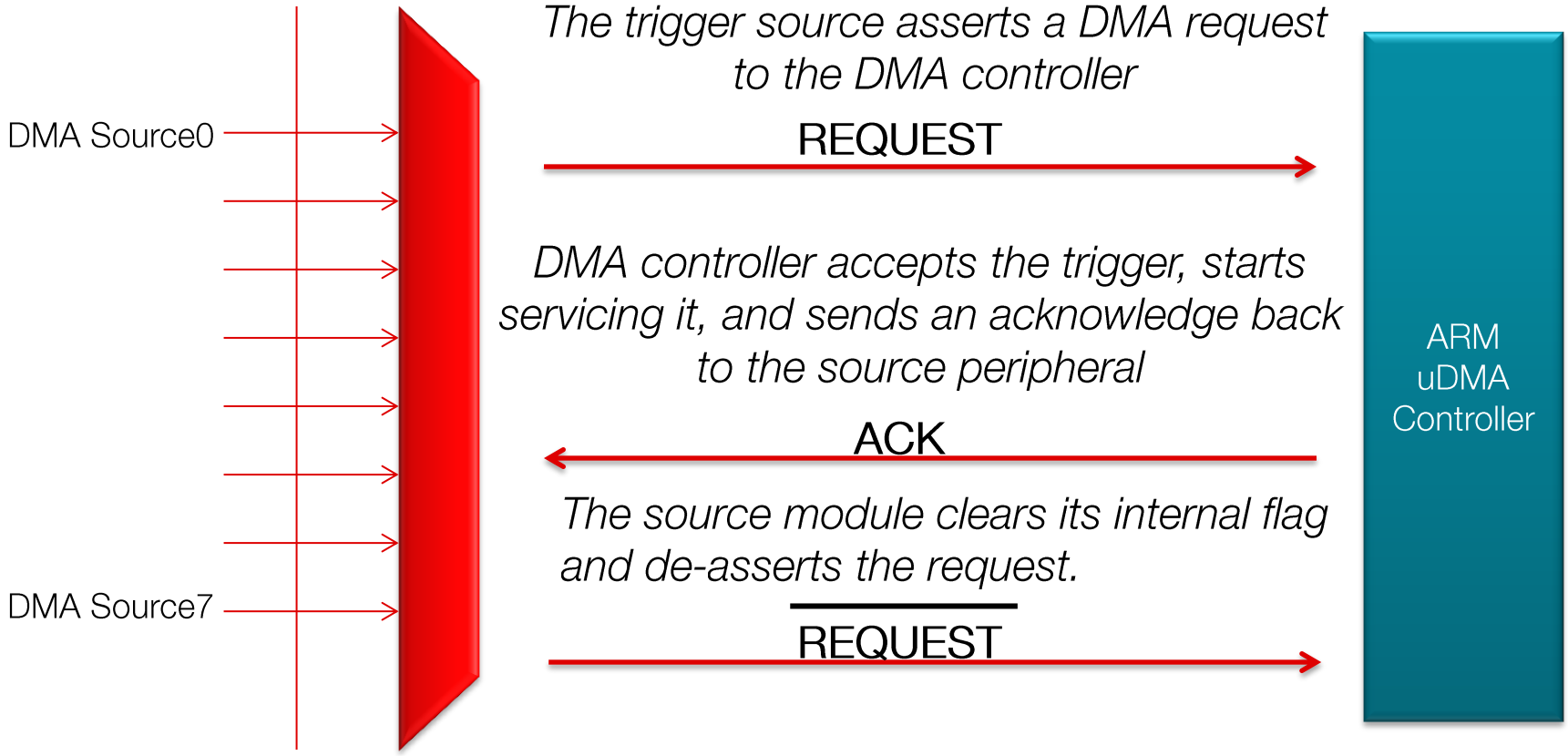
Automatically switches to lower power consumption mode if no active transfer is present

DMA | PL230 MicroDMA

- Single 32-bit AHB-Lite master interface for transferring data
- Supports multiple DMA transfer data widths and address increment values
- 1-1024 transfers in a single DMA access
- Each DMA channel has 2 channel control data structures (primary/alternate) stored in RAM
- Channel control data is stored in RAM
- Multiple DMA transfer types:
  - Basic
  - Auto-Request
  - Ping-Pong
  - Memory Scatter-Gather
  - Peripheral Scatter-Gather
- Each DMA channel has a programmable priority level.
- Error indication in case any transfer comes back with a bus error response.

DMA | Triggers

Each trigger works in a request-acknowledge mechanism with the requesting source



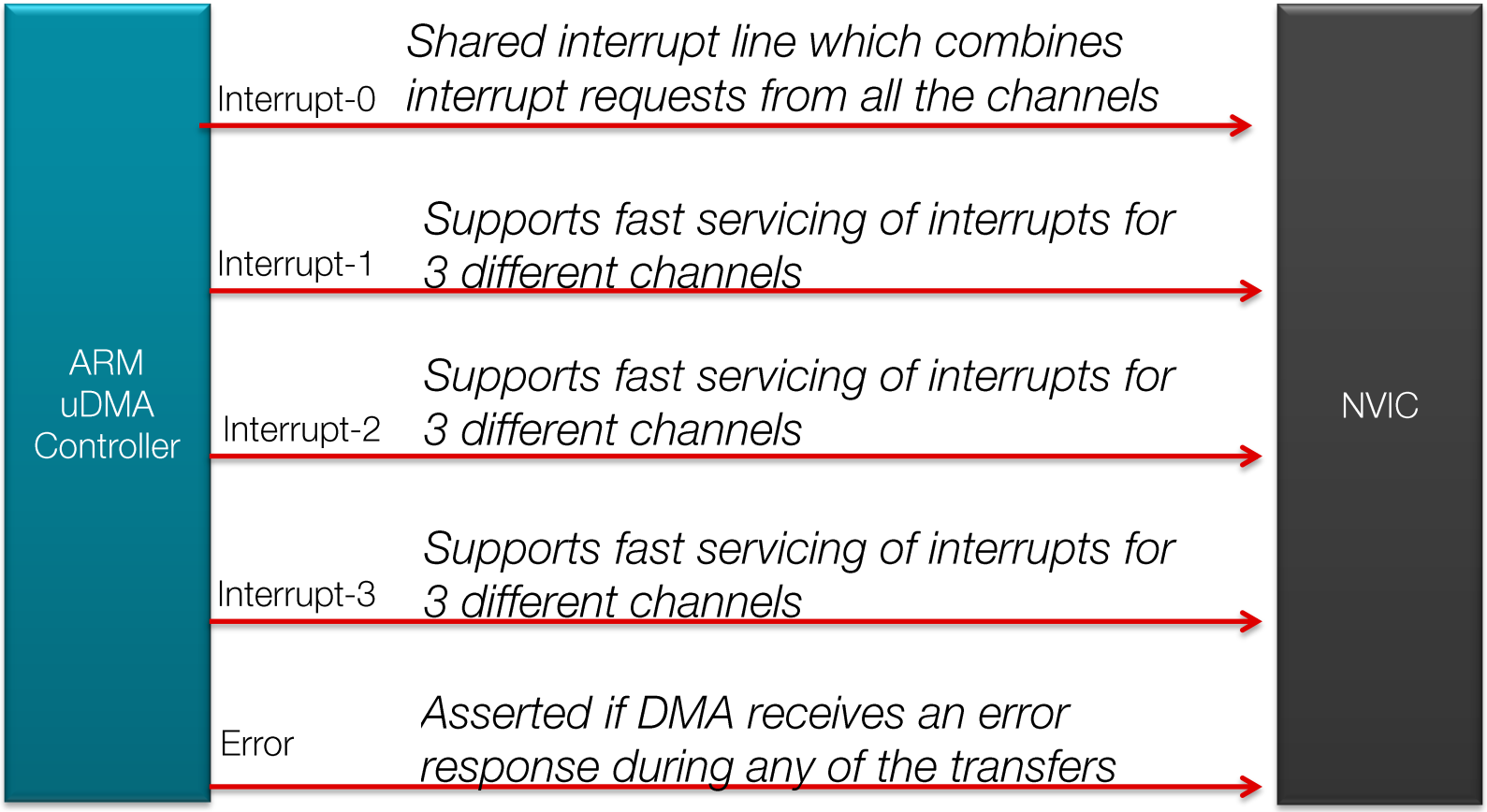
Each of the channels can also be triggered by a software trigger register

DMA | Triggers mapping

Channel	SRCCFG=0	SRCCFG=1	SRCCFG=2	SRCCFG=3	SRCCFG=4	SRCCFG=5	SRCCFG=6	SRCCFG=7
0	Reserved	eUSCI_A0 TX	eUSCI_B0 TX0	eUSCI_B3 TX1	eUSCI_B2 TX2	eUSCI_B1 TX3	TA0CCR0	AES256_Trig gen0
1	Reserved	eUSCI_A0 RX	eUSCI_B0 RX0	eUSCI_B3 RX1	eUSCI_B2 RX2	eUSCI_B1 RX3	TA0CCR2	AES256_Trig gen1
2	Reserved	eUSCI_A1 TX	eUSCI_B1 TX0	eUSCI_B0 TX1	eUSCI_B3 TX2	eUSCI_B2 TX3	TA1CCR0	AES256_Trig gen2
3	Reserved	eUSCI_A1 RX	eUSCI_B1 RX0	eUSCI_B0 RX1	eUSCI_B3 RX2	eUSCI_B2 RX3	TA1CCR2	Reserved
4	Reserved	eUSCI_A2 TX	eUSCI_B2 TX0	eUSCI_B1 TX1	eUSCI_B0 TX2	eUSCI_B3 TX3	TA2CCR0	Reserved
5	Reserved	eUSCI_A2 RX	eUSCI_B2 RX0	eUSCI_B1 RX1	eUSCI_B0 RX2	eUSCI_B3 RX3	TA2CCR2	Reserved
6	Reserved	eUSCI_A3 TX	eUSCI_B3 TX0	eUSCI_B2 TX1	eUSCI_B1 TX2	eUSCI_B0 TX3	TA3CCR0	**External pin**
7	Reserved	eUSCI_A3 RX	eUSCI_B3 RX0	eUSCI_B2 RX1	eUSCI_B1 RX2	eUSCI_B0 RX3	TA3CCR2	ADC14

DMA | Interrupts

The DMA controller supports 5 interrupt lines to the NVIC



Texas Instruments

DMA | DriverLib APIs

- DMA_enableChannel ()
- DMA_setControlBase ()
- DMA_enableChannelAttribute ()
- DMA_setChannelControl ()
- DMA_setChannelScatterGather ()
- DMA_getChannelSize ()
- DMA_assignChannel ()

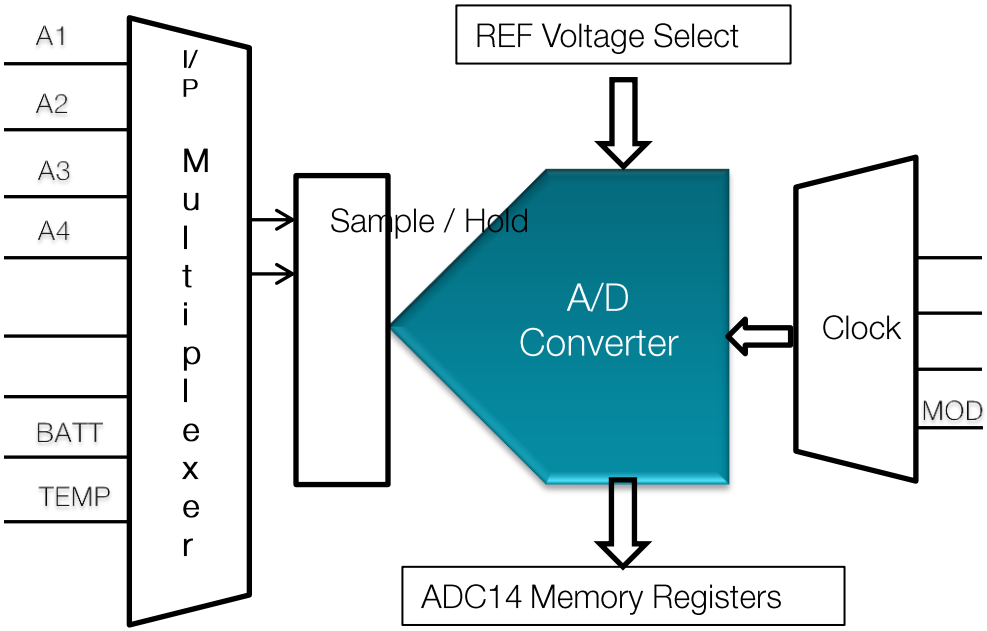
Analog Modules

ADC14, COMP_E, REF

Texas Instruments

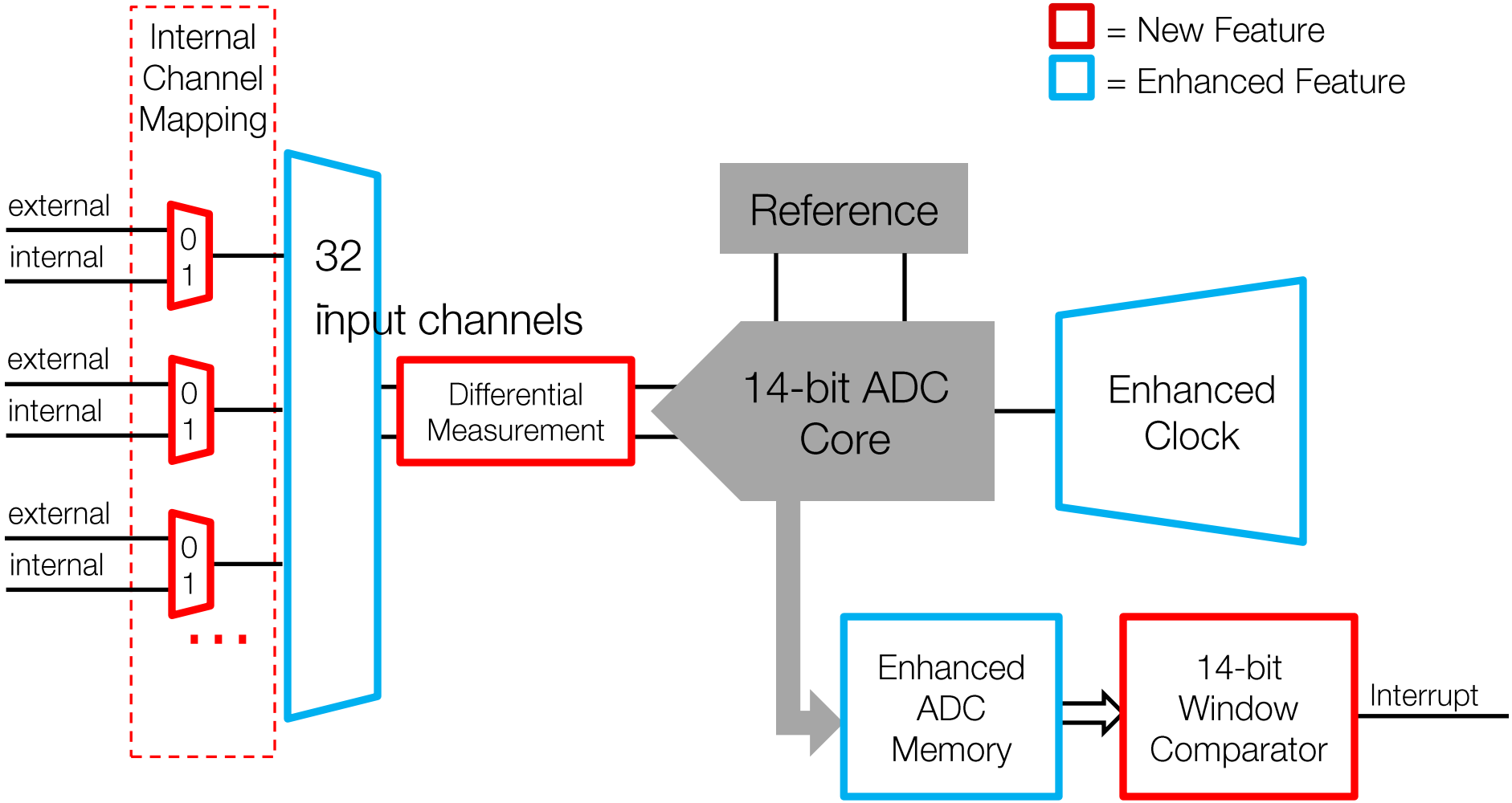
ADC14 | Overview

- 14-bit Accuracy
  - INL <= +/- 2 LSB;
  - DNL <= +/- 1 LSB
  - ENOB 13-bit
- 32-input channels
- Single-ended & differential Inputs
- 2 Window comparators
  - High interrupt
  - Low interrupt
  - In [Between] interrupt
- Separate internal channels for AVcc and TempSensor
- Ultra Low current consumption
  - Single ended 210uA @ 1.8V, 1Msps
  - Differential 260uA @ 1.8V, 1Msps



TI Information – Selective Disclosure

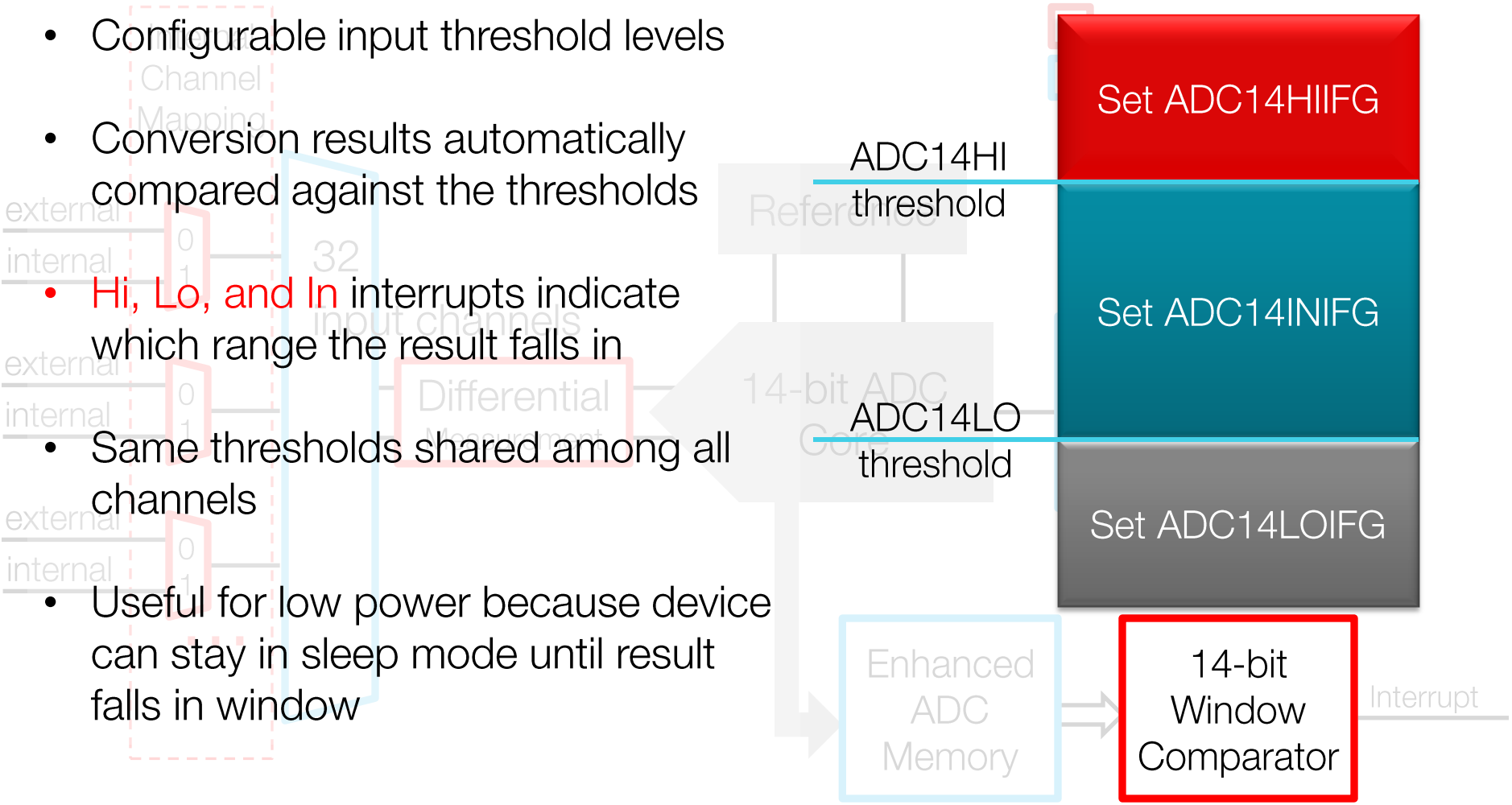
ADC14 | New Features



Texas Instruments

ADC14 | Window Comparator

- Configurable input threshold levels
- Conversion results automatically compared against the thresholds
- Hi, Lo, and In interrupts indicate which range the result falls in
- Same thresholds shared among all channels
- Useful for low power because device can stay in sleep mode until result falls in window

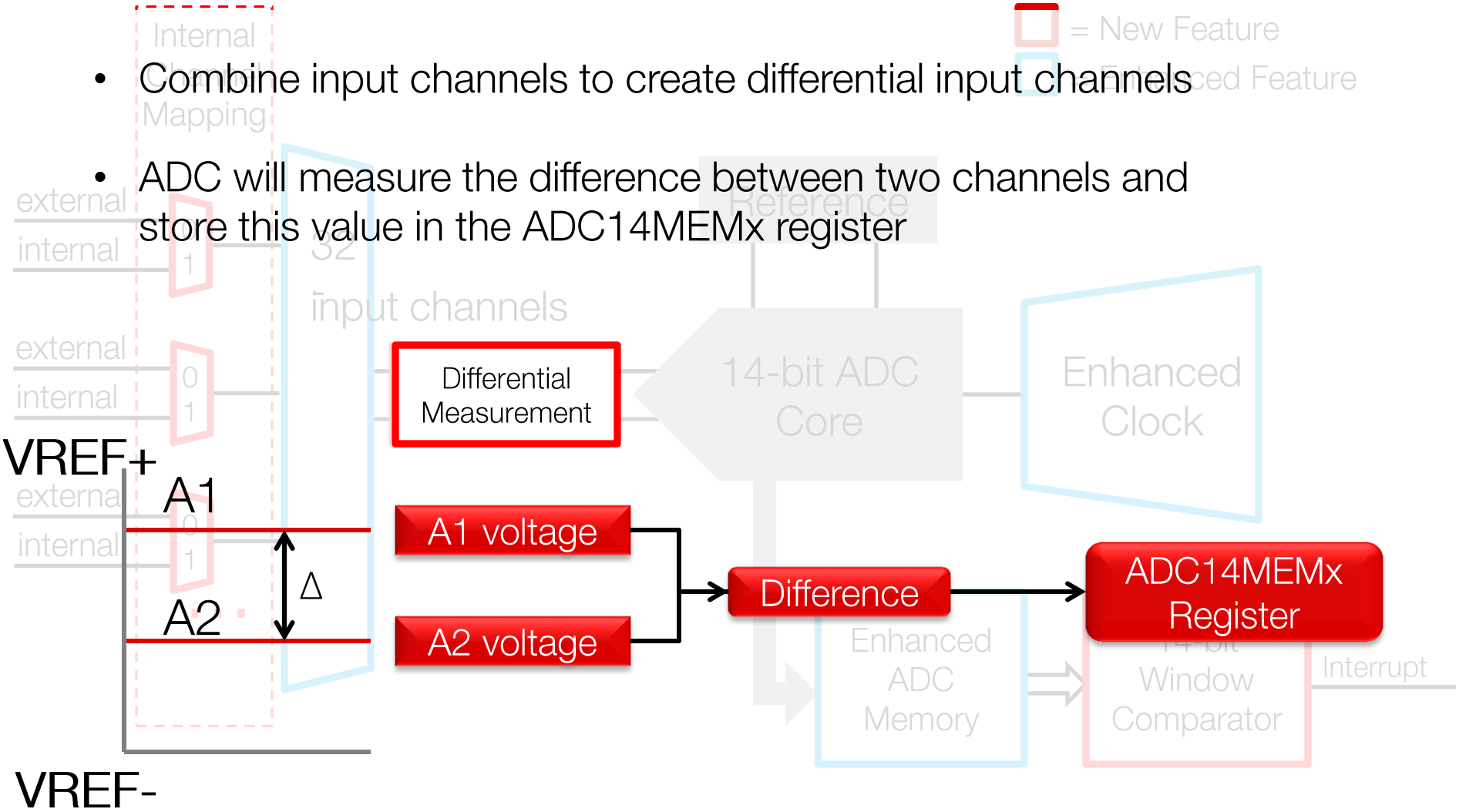


Texas Instruments

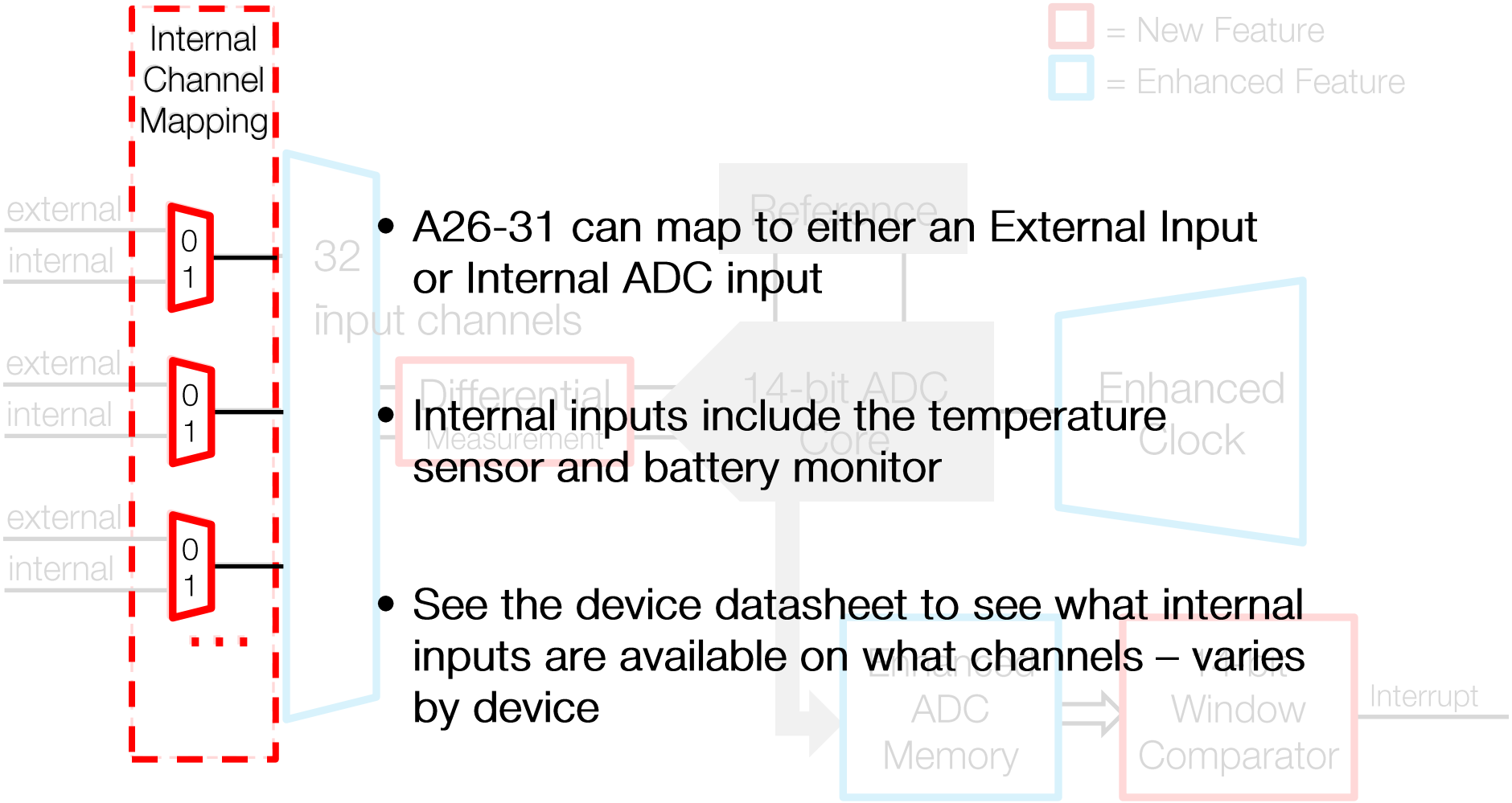
ADC14 | Window Comparator Example

ADC14 | Differential Measurement

- Combine input channels to create differential input channels
- ADC will measure the difference between two channels and store this value in the ADC14MEMx register

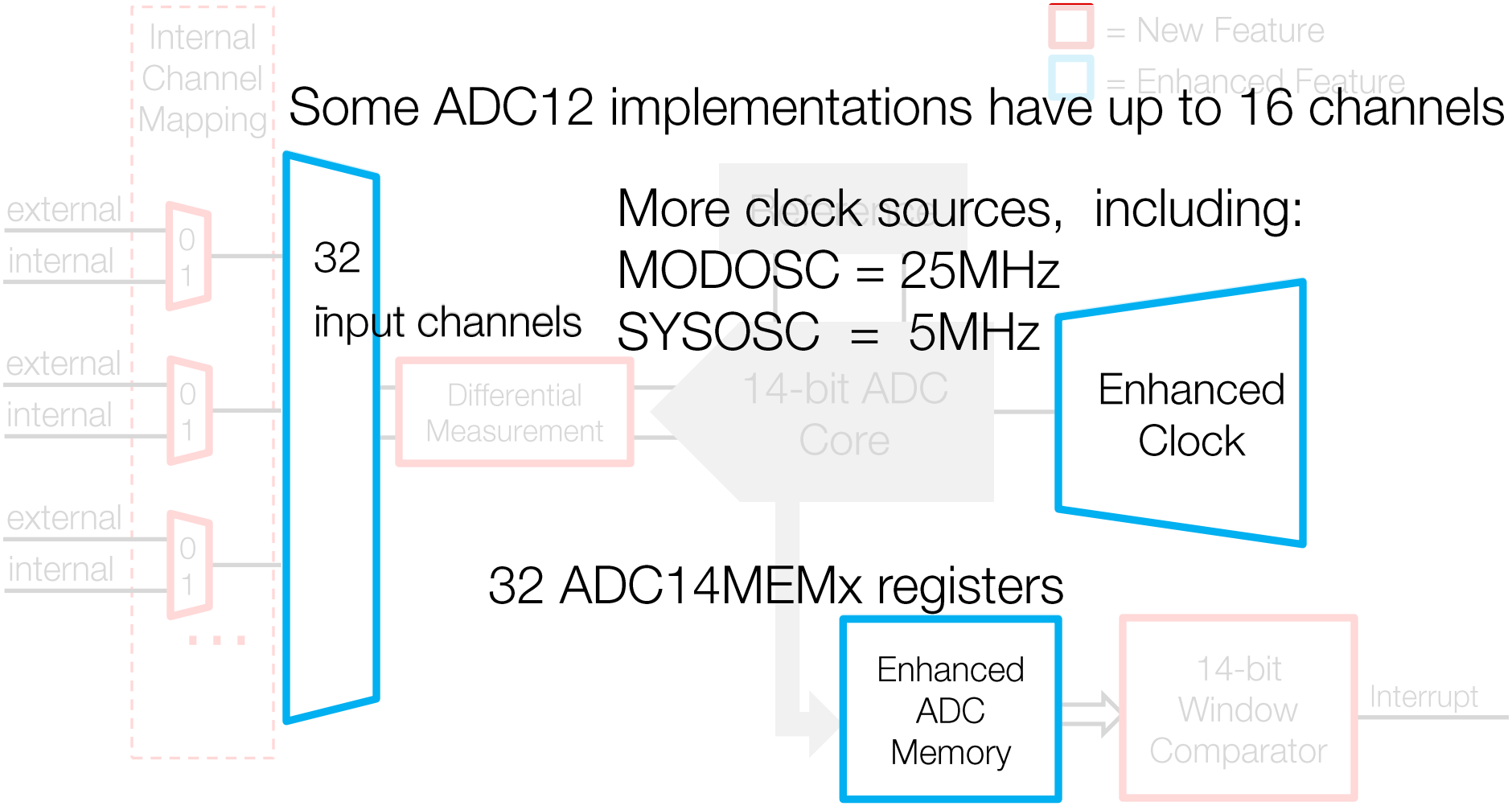


ADC14 | Internal Channel Mapping



- A26-31 can map to either an External Input or Internal ADC input
- Internal inputs include the temperature sensor and battery monitor
- See the device datasheet to see what internal inputs are available on what channels – varies by device

ADC14 | Other enhancements



Texas Instruments

Enhancements compared to ADC12_A/B

ADC12_A/B	ADC14

ADC14 | DriverLib APIs

- DriverLib API examples:
  - ADC14_enableModule();
  - ADC14_setResolution ();
  - ADC14_configureSingleSampleMode ();
  - ADC14_configureMultiSequenceMode();
  - ADC14_enableConversion();
  - ADC14_enableComparatorWindow();

Comp_E | Overview

- Interrupt driven for low power
- Uses the REF module like ADC14
- Up to 15 external input channels
- Ultra-low-power comparator mode
- Software selectable RC filter
- Selectable reference voltage generator
- Voltage Hysteresis generator
- Output internally connected to Timer A capture input (for event capture & capacitive touch implementations)

Comp_E | DriverLib APIs

- Comp_setReferenceVoltage ();
- Comp_setReferenceAccuracy(); // Static & sampled modes
- Comp_setPowerMode(); // high speed, normal, and ulp modes
- Comp_disableInputBuffer();

REF | Reference Module

- Generates voltage references for analog modules
  - 1.2V, 1.45V, 2.5V
- Available on output pin
- Scale-able power options
  - Static mode for high precision outputs
  - Sampled mode for low-power operations
- Two operation modes:
  - Continuous Mode: reference output continuously on
  - Burst Mode: reference output on only during ADC conversions
- Two buffers to throttle power consumption:
  - Large buffer for continuous mode or output VREF to pin
  - Small buffer for burst mode
- Provides temp sensor channel for ADC
  - Option to disable to minimize power

REF | DriverLib APIs

- Ref_setReferenceVoltage()
- Ref_disableTempSensor ()
- Ref_enableReferenceVoltageOutput ()
- Ref_isBandgapActive ()
- Ref_isRefGenBusy ()
- Ref_setReferenceVoltageOneTimeTrigger ()

Lab 4 | ADC14 [& Timer_A]

Texas Instruments

Lab 4 | ADC14 [&Timer_A]

Lab Goals

- Learn how to configure the new & enhanced ADC14 module & perform ADC.
- Exercises Timer_A & GPIO operations using direct register access.
- 1st look at the new Educational BoosterPack MK II

Instructions

- Follow lab manual to complete & build project
- Using LaunchPad & BoosterPack
  - The X-axis of the JoyStick is the analog input
  - The LED on the LaunchPad is the PWM output
- Use the joystick to adjust the LED's brightness
- Bonus exercise: use the lab4_adc14_bonus_buzzer.c file to output the joystick analog values to the buzzer instead of the LED.

Security

Security | AES256

Benefits

- Hardware acceleration for AES (Advanced Encryption Standard – FIPS PUB 197)
- Accelerates AES en- and decryption by one to two orders of magnitude (compared to software)
- Lower power (compared to software)
- Off-loads CPU

Performance

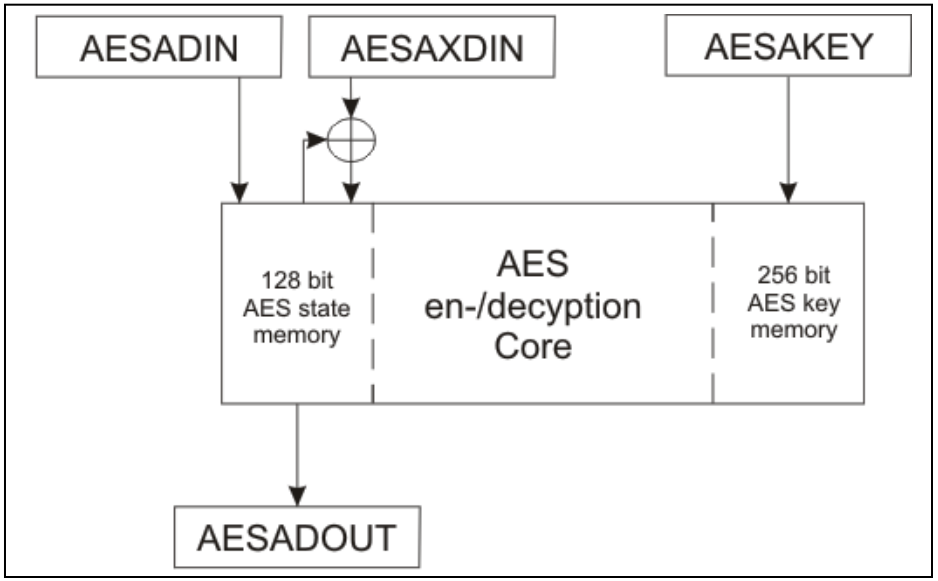
- 128-bit of data are en- or decrypted with a 128-bit key within 167 MCLK cycles, 256-bit encryption in 234 cycles

Features

- supporting 128-, 192-, and 256-bit key lengths
- On-the-fly key expansion
- Off-line key generation for decryption
- Shadow registers for initial key
- 128 bit truly random seed to generate the random key
- Block mode with DMA: OFB, ECB, etc.

Applications include

Securing of communication channels like RF links, UART communications, etc.



Security | Device Security Overview

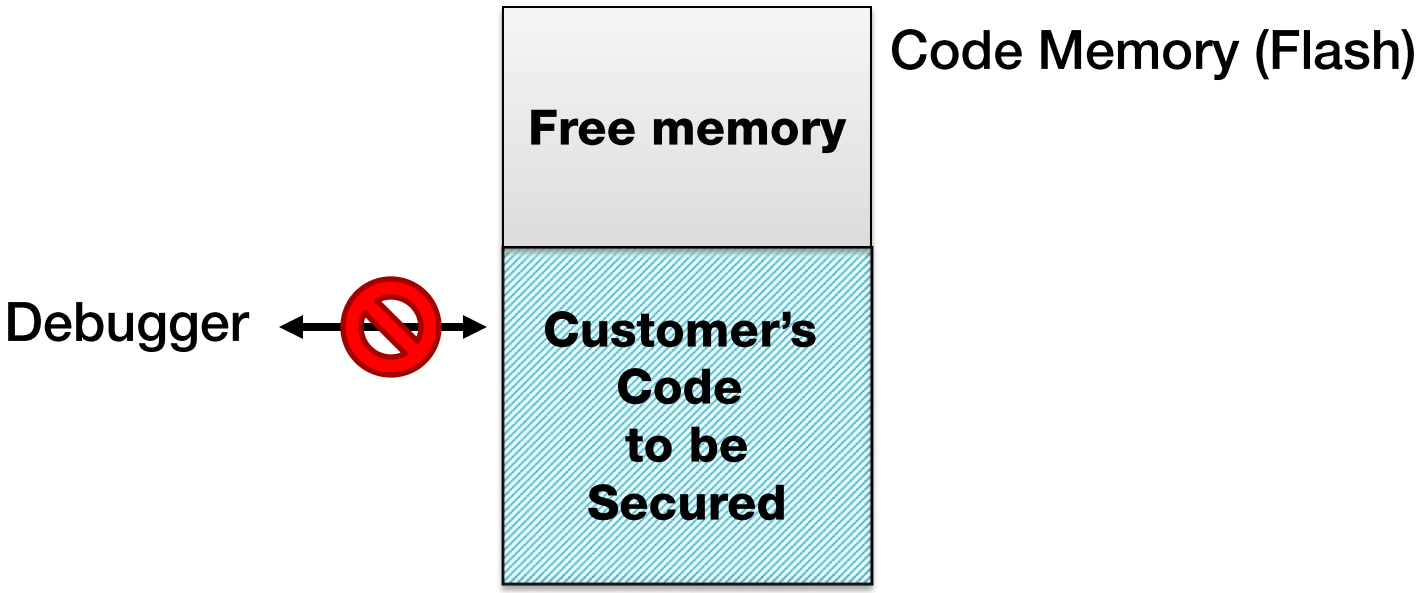
432

- Full chip security: JTAG Protection
  - Device is fully secure from debug accesses.

- Regional security: IP Protection
  - Up to 4 regional secure zones of memories can be configured in flash
  - These zones are non-intrusive to debug (JTAG/debugger) and code outside this region.
  - Regular debug allowed outside these regional secure zones.

- Combination of regional and full-chip security
  - This is a combination of the above two where:
    - Debug accesses are blocked for the entire device
    - Code/Data access restrictions and protections apply for the secure regions.

Security | Full chip/JTAG security

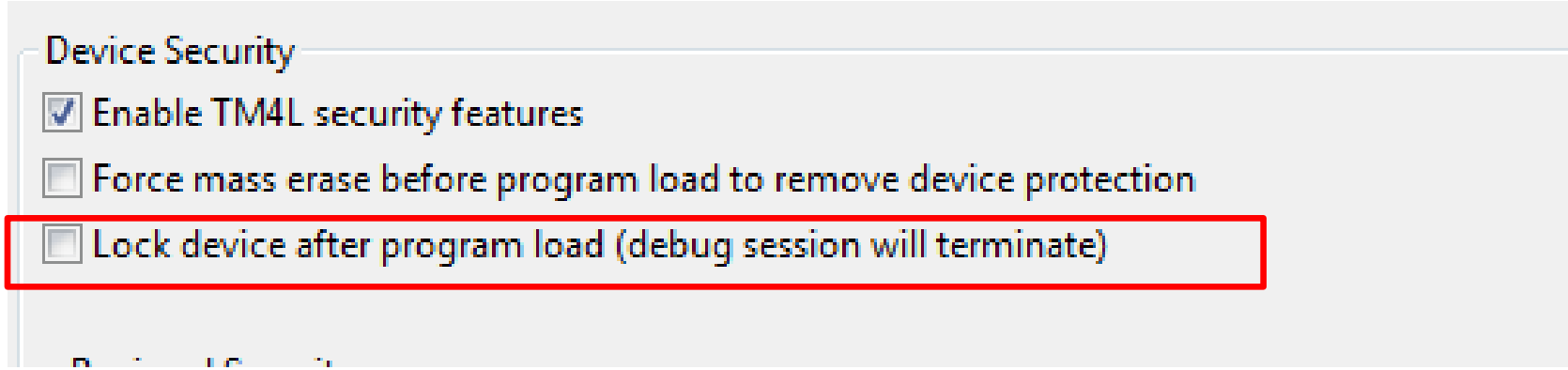


- Similar to MSP430 JTAG security control
- Code is developed and loaded into Flash by customer
- Device is then completely locked from external accesses (full chip secure)
- All memory areas within the device can be accessed by the customer's code.

Texas Instruments

Security | Full chip/JTAG security

- Only recoverable as blank device after mass erase
- Option available in CCS Project Settings



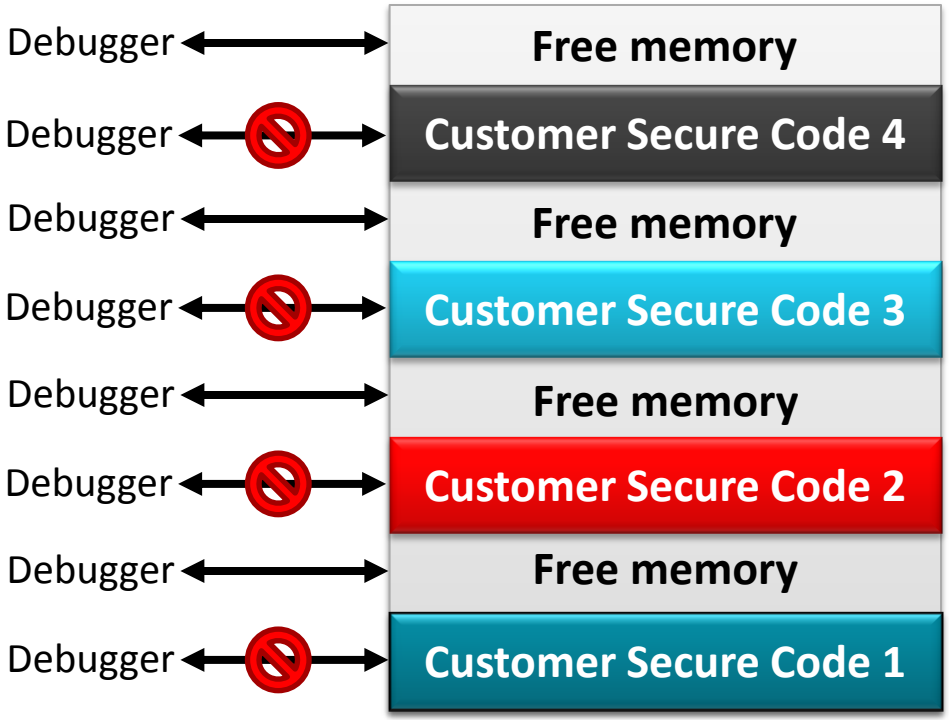
Security | IP Protection - Regional Security

- Configure up to 4 regionally secured zones to protect IP from read/write access
- Facilitate securing protected IP code from vendors
- Prevent access during IP code execution by stalling JTAG until end of code execution
- Regionally secured zones can be updated via BSL
- Ease of use: Use CCS & other IDEs to configure and secure zones

Texas Instruments

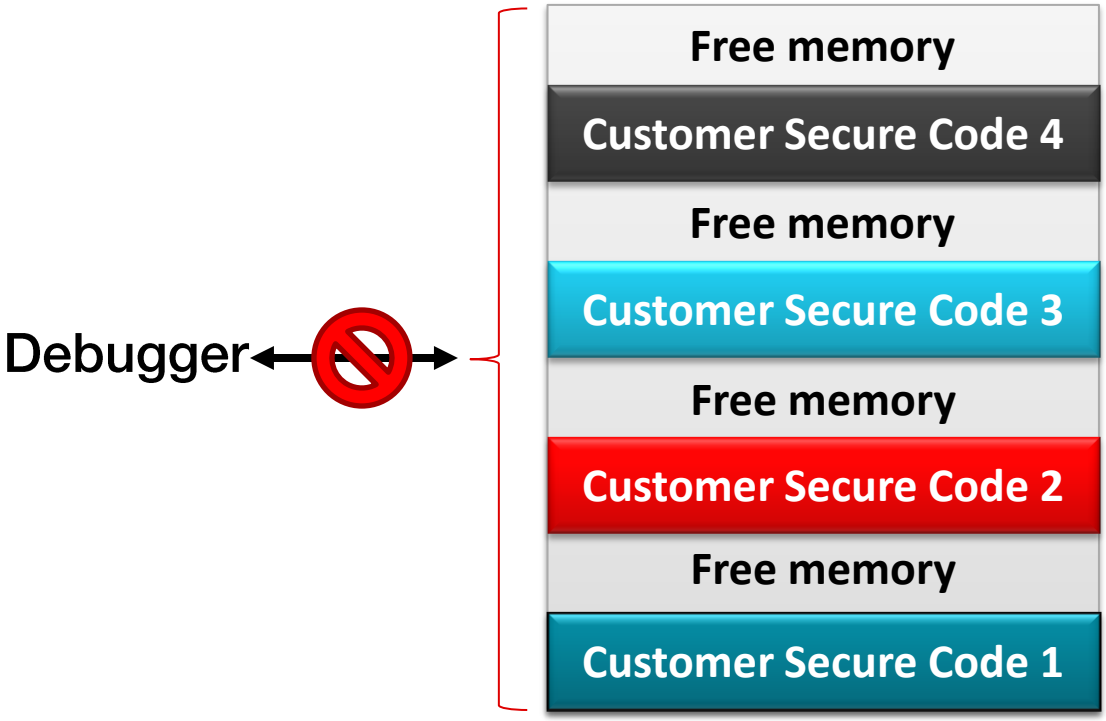
Security | Regional secure zones

- Up to **4 regions of code** can be setup to be secured against:
  - Debugger accesses
  - Code accesses from other regions
- Remaining regions of memory are still open for debug/code accesses
- Enables multi vendor development scenario with specific vendors' protected software IP can be protected in secure regions
- Code in secure zones can only be called into as APIs
  - Direct read accesses result in an exception



Texas Instruments

Security | Combined Full-chip / Regional secure zones



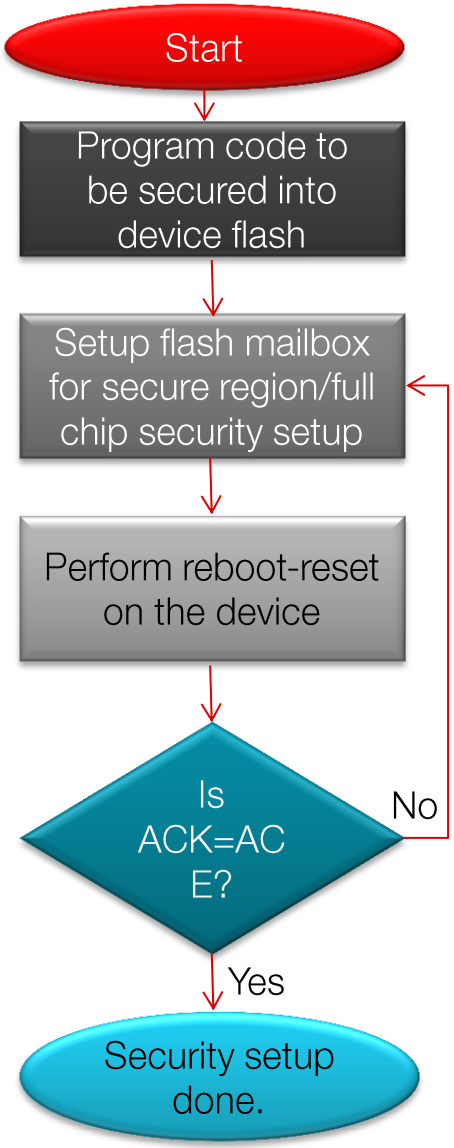
- Combined advantage of regional secure zones and full-chip device security
- Entire device is not accessible through debugger

TI Information – Selective Disclosure

Security | Device security setup

Security setup Mailbox structure:

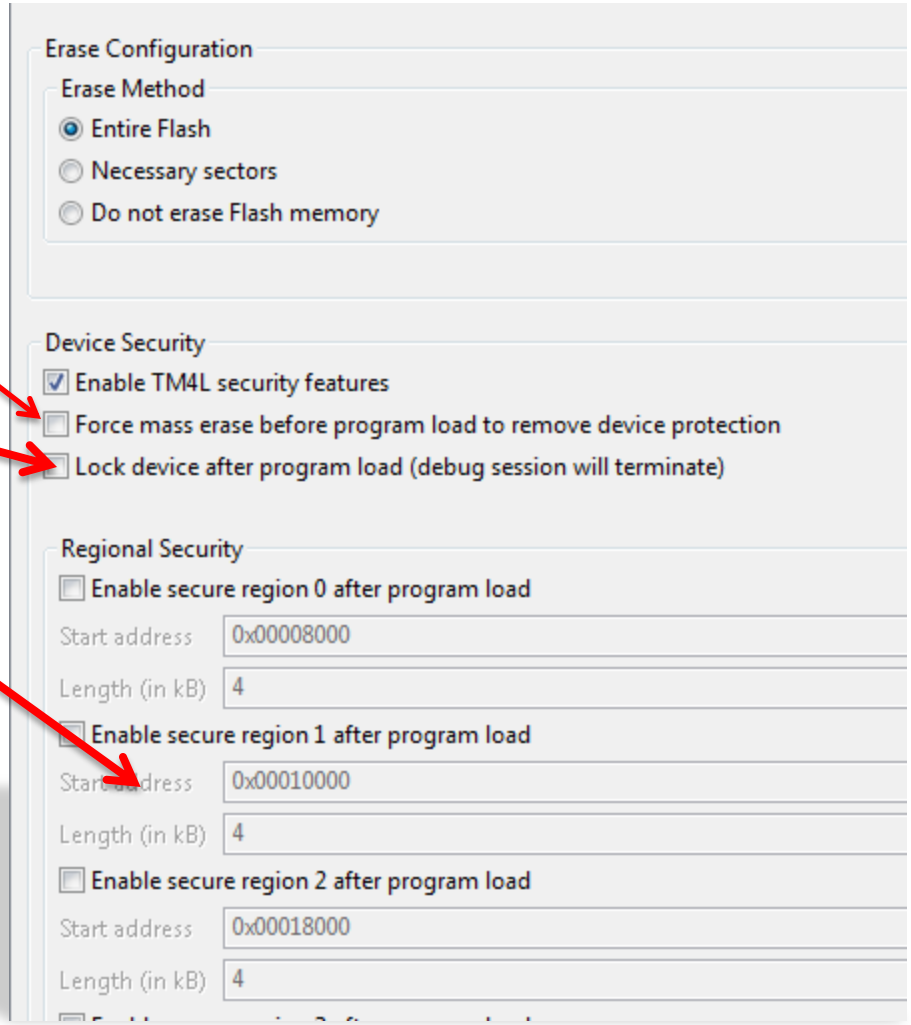
- CMD – Command (regional secure zone X setup/full chip secure)
- ACK – SUCCESS/ERROR for CMD
- PARAMS – security parameters
  1. ENABLE
  2. START_ADDR
  3. LENGTH
  4. AES_INIT_VECTOR
  5. AES_DECRYPTION_KEYS
  6. UPGRADE PASSWORD
  7. ENCRYPTED_UPDATE_EN
  8. DATA_ACCESS_EN



Texas Instruments

Security | Device security setup through CCS

- Device security setup has been implemented in code composer studio for ease of use
  - Erase all the secure code first and then delete security settings
  - Enable full-chip security
  - Enable regional security for each of the 4-regions and configure start address and size



Texas Instruments

Security | [Field] Firmware Updates

- Multiple options to update code in a secure device
  1. Full-chip secure unencrypted update
  2. Full-chip secure encrypted update
  3. Regional secure zone secure unencrypted update
  4. Regional secure zone secure encrypted update

- **Unencrypted updates**: option to use password-based authentication

- **Secure encrypted updates:**
  - Code transfer packets are encrypted with AES
  - Useful in protecting a snooping on the communication line.
  - In addition to password-based authentication.

MSP432 Software

Software | Get Started Quickly

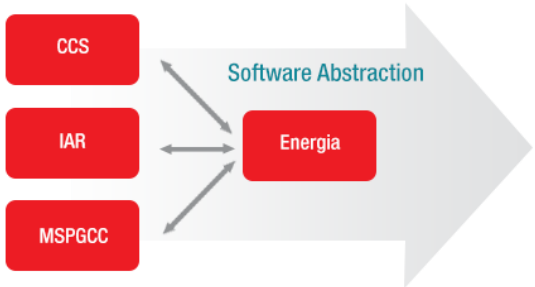
Development Environment

MSP IDEs

- Feature **full debugging** capability, additional **design support** & **optimization** tools

Energia

- Simplified programming environment: **simplify coding** to high level functions



Design Resources

MSPWare

- **Sleek & intuitive GUI** for navigating code examples, docs, APIs & other resources

MSP Driver Library

- **Faster time to market:** abstracted API library eliminates bitwise programming

Software Optimization

Ultra-low-power Software Tools

- **Checks entire project** against a set of ULP rules created & verified by industry ULP experts

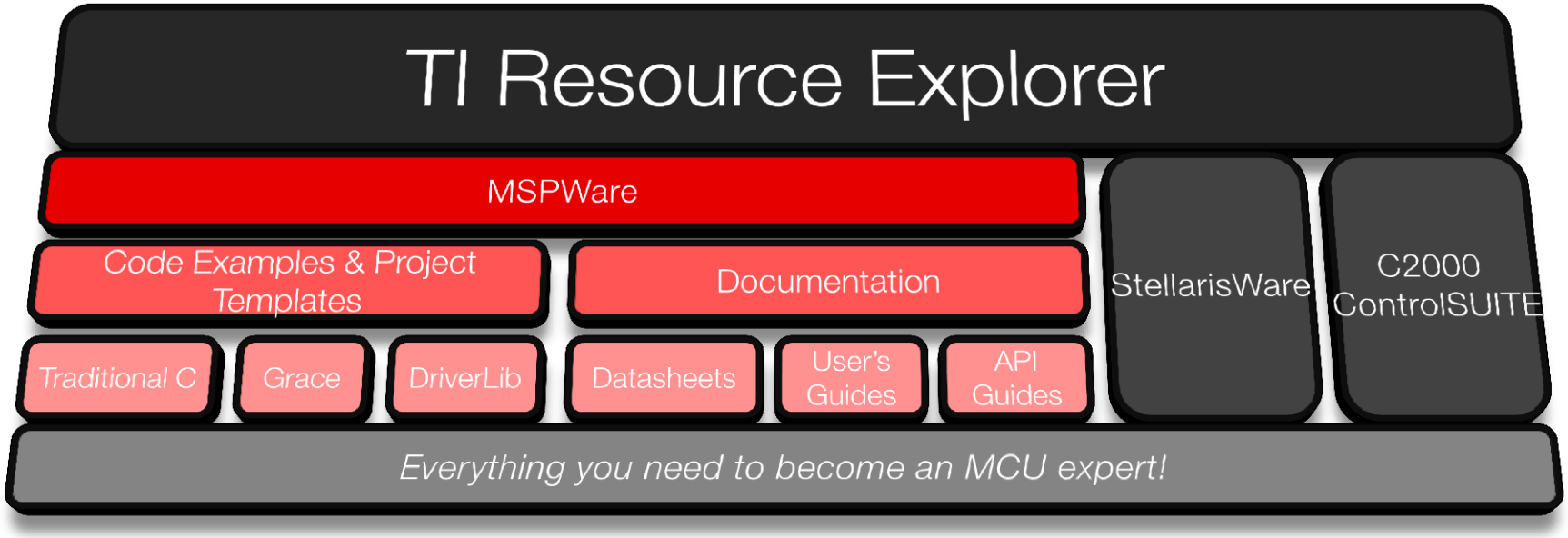
Application Specific Libraries

- **Variety** of application specific libraries: Graphics Library, RF libraries

ULP Advisor™
Code Analysis Tool
for MSP430 Microcontrollers

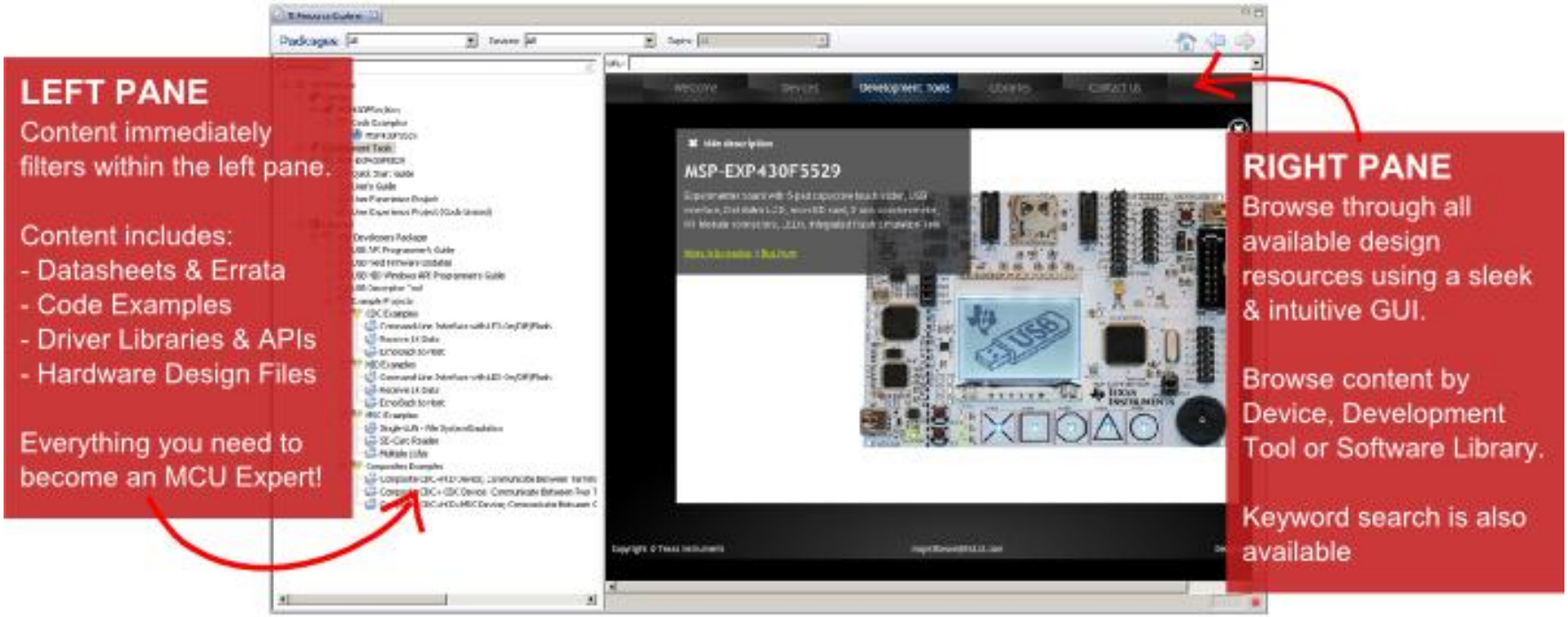
Texas Instruments

Software | TI Resource Explorer with MSPWare



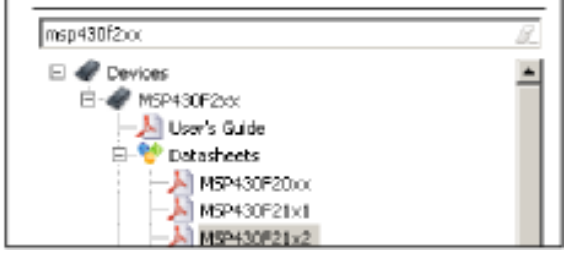
- **Embedded developer's toolbox for "One stop learning"**
  - An intuitive GUI that categorizes and organizes development resources to simplify your creative process.
- **Includes collateral & resource packages for TI microcontrollers**
  - **MSPWare** | StellarisWare | C2000 ControlSUITE | Hercules Tools
- **Automatically updates over the web**
  - Ensure that developers have the latest and greatest documentation & resources
  - Get up-to-date device-support resources as MSP portfolio continues to grow

Software | MSPWare: CCS & Stand-alone



Filter resources by Device, Development Tool or Software Library

Open and read datasheets & user's guides from within the GUI



Import code examples directly into Code Composer Studio

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)
```

Driver Library

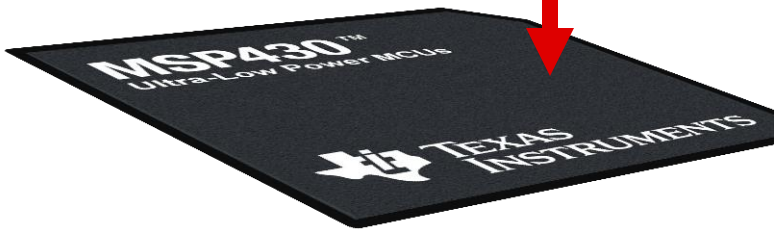
Traditional C code

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

Low level programming

```
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



TI Information – Selective Disclosure

Software | Driver Library Function Calls

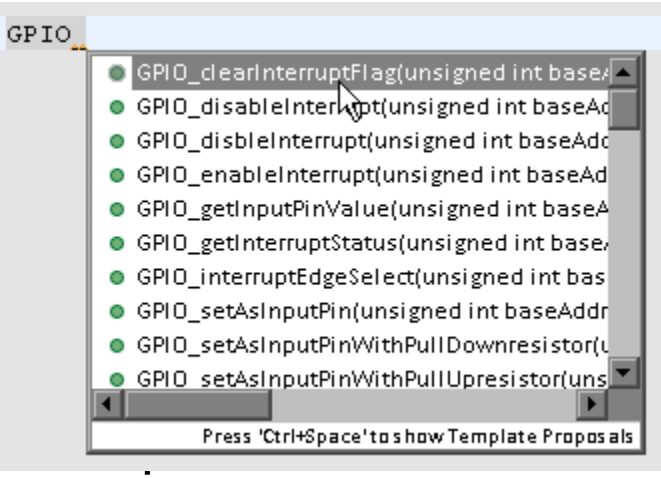
Typical DriverLib Format

Function_name([Instance#] , Param1, Param2, …);

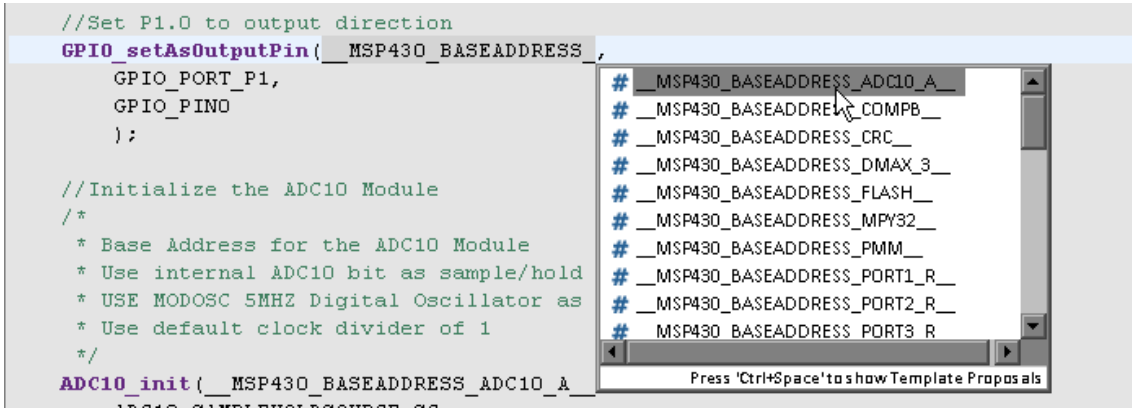
Instance# = Optional parameter, required only for peripherals w/ > 1 instances

Hint: Use CTRL+SPACE for autocomplete...

Suggested functions…



Suggested parameters…

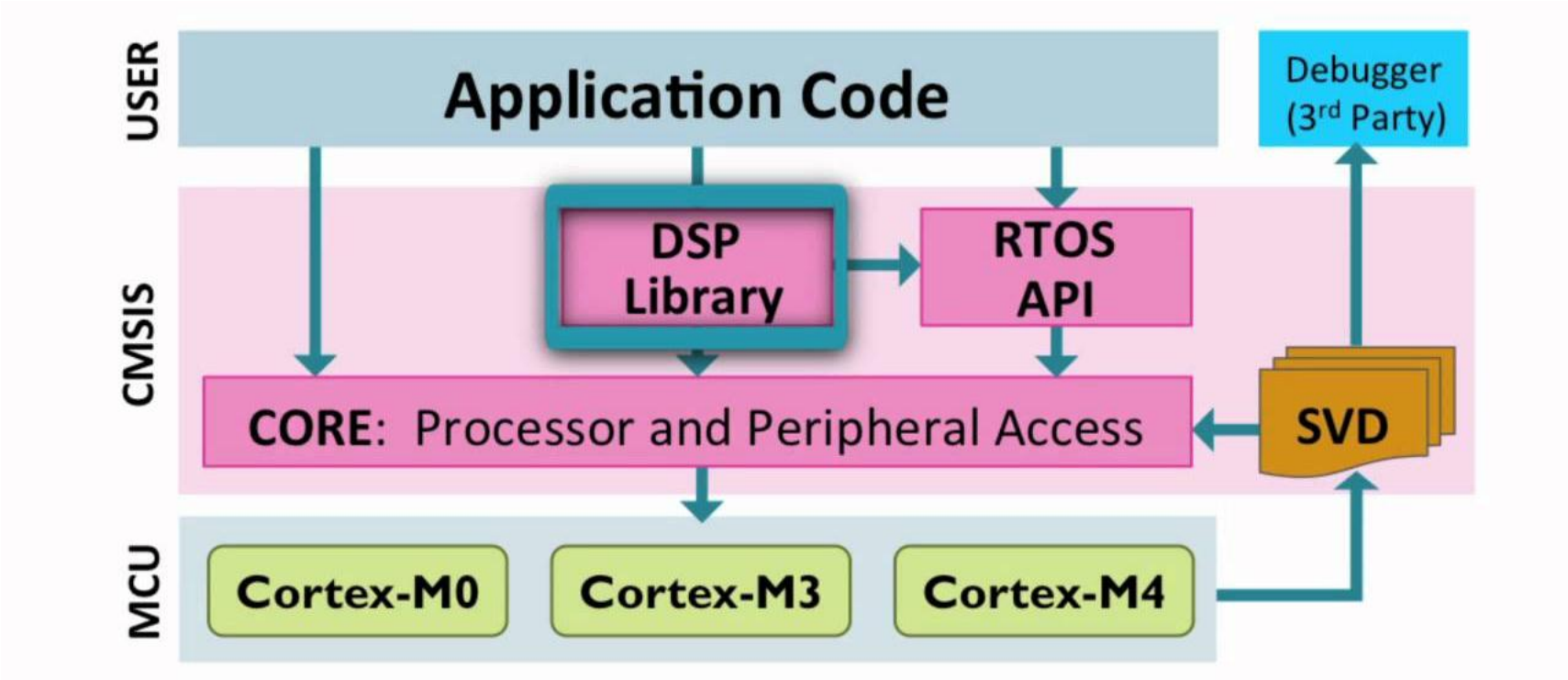


GPIO_setAsOutputPin (GPIO_PORT_P1,
GPIO_PIN1 + GPIO_PIN2);

Texas Instruments

Software | CMSIS

- Cortex Microcontroller Software Interface Standard (CMSIS)
- Standardized hardware abstraction layer for the Cortex-M4 processor series



Software | CMSIS

CMSIS COMPLIANT
ARM® Cortex™ Microcontroller Software Interface Standard

MSP definition

```
#define WDTCTL \
        (HWREG16(0x4000480C))

/* WDTCTL Control Bits */
#define WDTIS0      (0x0001)
#define WDTIS1      (0x0002)
#define WDTIS2      (0x0004)
#define WDTCNTCL    (0x0008)
#define WDTTMSEL    (0x0010)
#define WDTSSEL0    (0x0020)
#define WDTSSEL1    (0x0040)
#define WDTHOLD     (0x0080)
#define WDTPW       (WDTPW_VAL)

#define __WDT_BASE__ (0x40004800)
```

CMSIS definition

```
typedef struct {
  uint8_t RESERVED0[12];
  union {    /* WDT_CTL Register */
    __IO uint16_t reg;
    struct { /* WDT_CTL Bits */
      __IO uint16_t IS    :      3;
      __O  uint16_t CNTCL :      1;
      __IO uint16_t TMSEL :      1;
      __IO uint16_t SSEL  :      2;
      __IO uint16_t HOLD  :      1;
      __IO uint16_t PW    :      8;
    } bit;
  } CTL;
} WDT_Type;
#define WDT ((WDT_Type *) __WDT_BASE__)
```

Software | CMSIS Core & Intrinsics

cmsis_ccs.h

```
// Wait for interrupt, a.k.a. go to sleep/deep-sleep
__attribute__((always_inline)) static inline void __wfi(void)
{
  __asm("  wfi");
}
// Enable Interrupts
__attribute__((always_inline)) static inline void __enable_irq(void)
{
 __asm("  cpsie i");
}
```

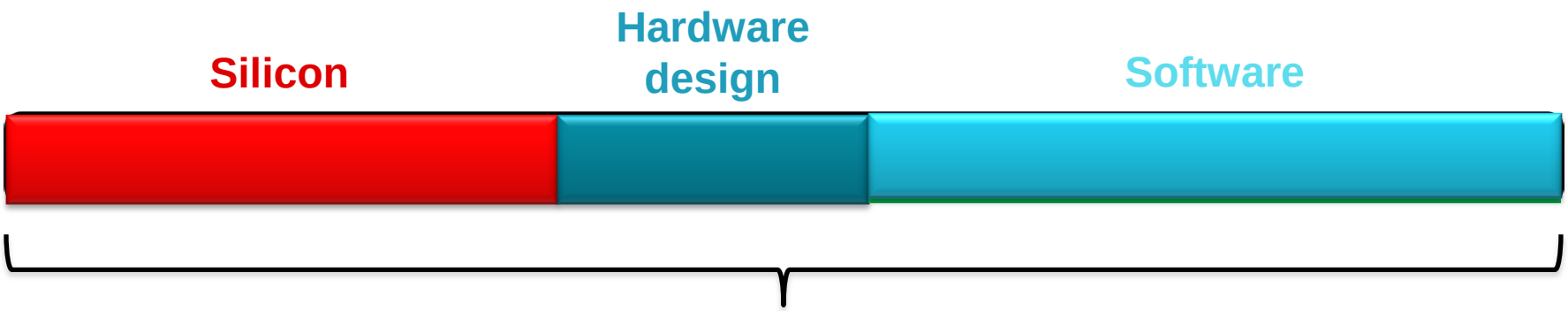
Translated & re-defined for additional MSP ease of use

msp430dna.h

```
#define __sleep()                    __wfi()
#define __enable_interrupt()         __asm("  cpsie i")
#define __disable_interrupt()        __asm("  cpsid i")
```

Software | ULP Advisor™

How ULP is your application?



Power consumed is made up of many factors.

Hardware is only half of the equation. We need Optimized software

ULP Advisor Tool can get you all the way there

TI Information – Selective Disclosure

ULP Advisor | Now with MSP432 Rules

ULP#	Name
ULP 1.1	Ensure LPM usage
ULP 2.1	Leverage timer module for delay loops
ULP 3.1	Use ISRs instead of flag polling
ULP 4.1	Terminate unused GPIOs
ULP 5.1	Avoid processing-intensive operations: modulo, divide. ***(MSP430 only)***
ULP 5.2	Avoid processing-intensive operations: double-precision floating point
ULP 5.3	Avoid processing-intensive operations: (s)printf()
ULP 6.1	Avoid multiplication on devices without hardware multiplier ***(MSP430 only)***
ULP 6.2	Use MATHLIB for complex math operations ***(MSP430 only)***
ULP 7.1	Use local instead of global variables where possible
ULP 8.1	Use 'static' & 'const' modifiers for local variables ***(MSP430 only)***
ULP 9.1	Use pass by reference for large variables
ULP 10.1	Minimize function calls from within ISRs
ULP 11.1	Use lower bits for loop program control flow ***(MSP430 only)***
ULP 11.2	Use lower bits for port bit-banging (MSP430 only)
ULP 12.1	Use DMA for large memcpy() calls
ULP 12.1b	Use DMA for potentially large memcpy() calls
ULP 12.2	Use DMA for repetitive transfer
ULP13.1	Count down in loops / Write loops with the termination test at the bottom
ULP 14.1	Use unsigned variables for indexing
ULP 15.1	Use bit-masks instead of bit-fields
ULP 16.1	Detect for > 4 parameters passed in function call ***(MSP432 only)***
ULP 17.1	Use bit-banding for bit manipulation ***(MSP432 only)***
ULP 18.1	Use natural-size integer variables for computation, convert to smaller size for storage ***(MSP432 only)***

Texas Instruments

EnergyTrace™ Real-Time Power Consumption Technology for MSP430™ and MSP432™ MCUs

Features/Devices	All MSP430 MCUs	MSP432 MCUs	FR5969 (based on Wolverine platform)
Current Monitoring	X	X	X
CPU State (PC)		X	X
Peripheral/System State			X
	EnergyTrace™ technology	EnergyTrace+™ technology	EnergyTrace++™ technology

Current Monitoring: Real-time monitoring of total energy usage of MCU

CPU State: Real-time monitoring of energy usage according to code being handled by CPU

Peripheral/System State: Real-time monitoring of energy usage by specific peripheral

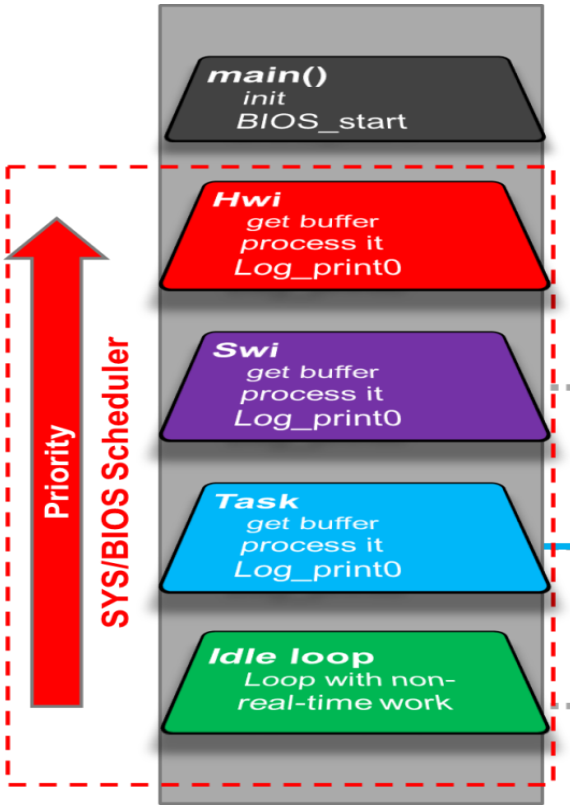
Verbose version:
EnergyTrace+[CPU States]+[Peripheral States]

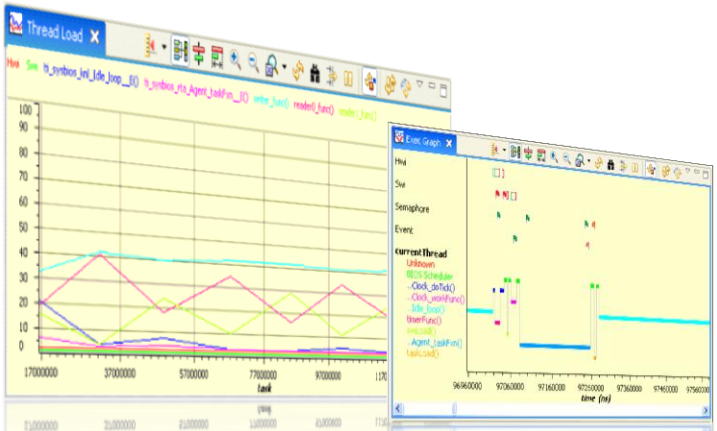
Software | Real-Time Operating Systems (RTOS)

As MSP devices expand to larger memory footprints and high integration, an RTOS can provide deterministic timing to your application & ensures that the highest priority thread is executed

TI-RTOS available for MSP432, future 3rd parties:

- FreeRTOS
- Micrium uC/OS-II
- embOS,TinyOS
- And more!



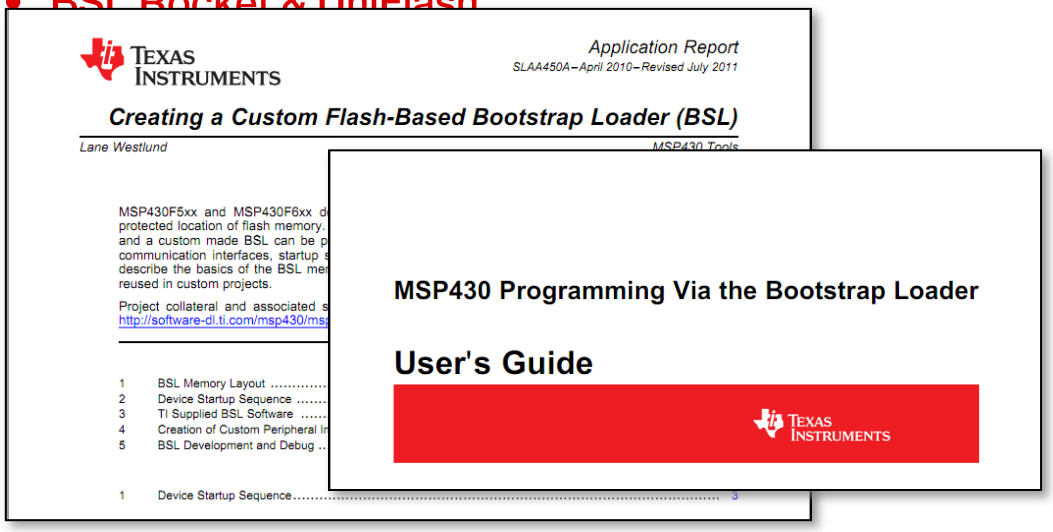


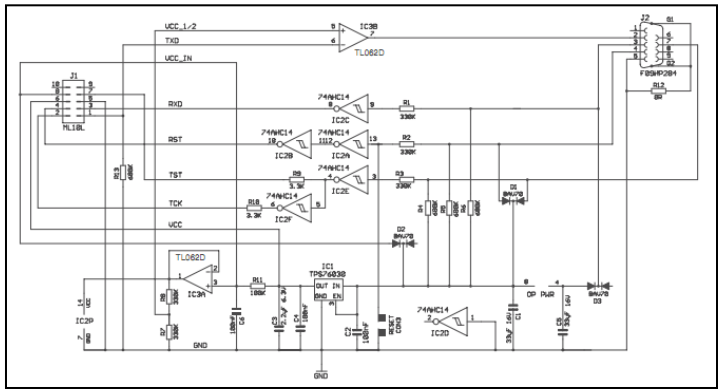
GUI-based monitoring & debug tools also available to simplify RTOS development!

TI Information – Selective Disclosure

Software | MSP Bootstrap Loader (BSL)

- The Bootstrap Loader, or BSL, is a program built into an MSP device designed to communicate with the device, primarily for the purpose of reading and writing to memory.
- This is often done via commands send through a serial link (UART), but can be done through other protocols, such as USB or RF (custom protocols or future devices).
- A complete MSP BSL User's Guide is provided, which fully documents the BSL shipped in all TI devices, enabling developers to fully utilize the BSL for production programming or other purposes.
- An application note is also available for customers interested in creating their own custom BSL
- BSL Rocket & UniFlash





Software and hardware circuit diagrams are provided, which can be built to interface directly to the MSP430 BSL.

Learn more about MSP430 BSL

Lab 5 | Software

Lab 5 | Software: DriverLib & CMSIS

Lab Goals

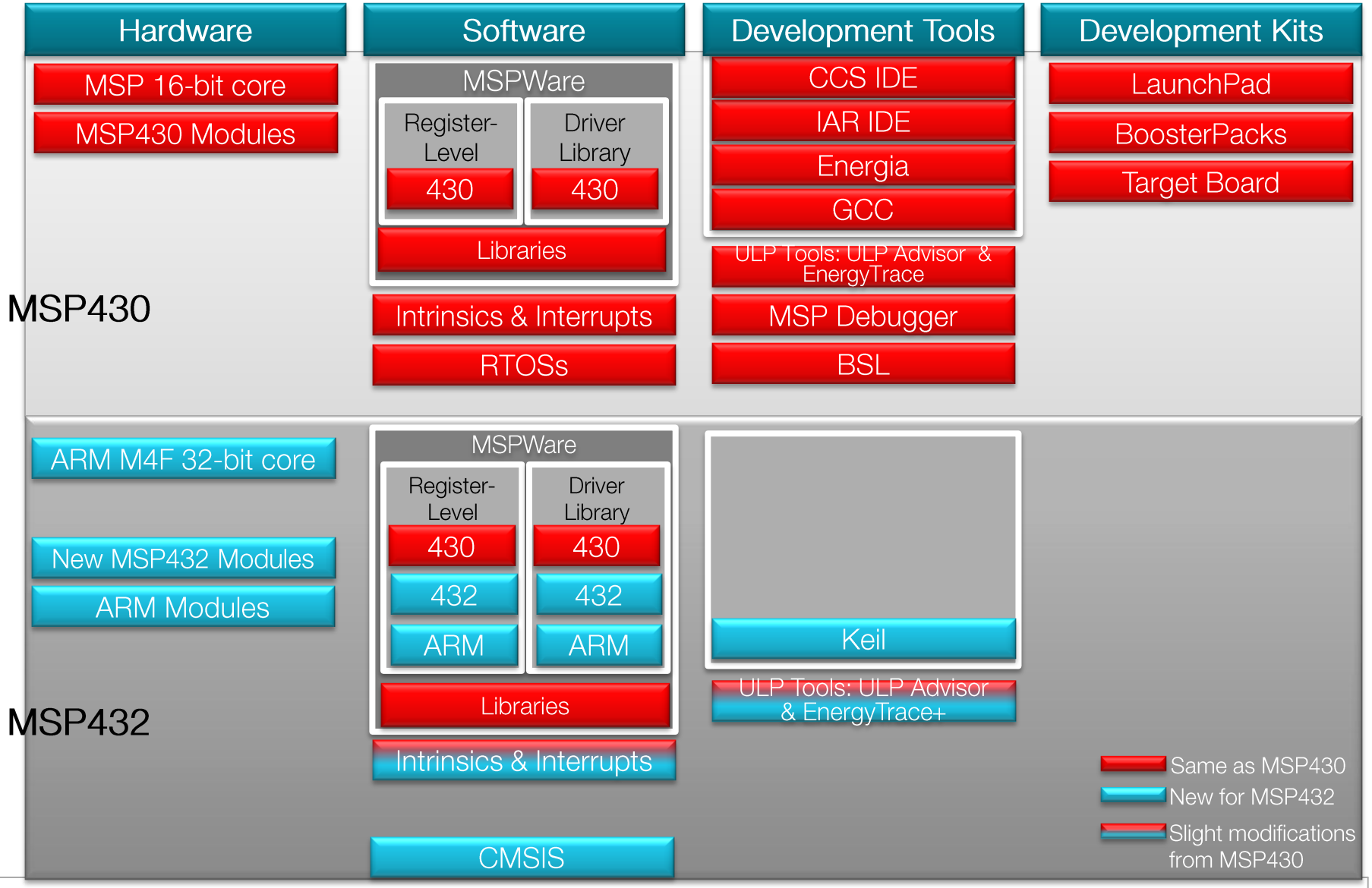
- Learn how to leverage different software solutions available at your finger tips: register-access, DriverLib, CMSIS, and MSP-DNA intrinsics.
- Learn how to navigate through DriverLib User's guide & API guide
- Leverage the CCS' autocomplete & Ctrl+Click features to quickly obtain the desired API
- Make educated assessment, comparison, trade-off, and porting effort between different software styles/techniques/solutions.

Instructions

- Based on Lab 4, this lab requires you to convert the existing register-access code to DriverLib API calls to exercise the same modules as Lab 4 (ADC14, Timer_A, etc.)
- Bonus: convert the register-access code to CMSIS-style code

MSP430 <-> MSP432 Porting

MSP Platform Portability



Porting | Code Compatibility

	Module	C-code	DriverLib	Recommendation
CORE SYS	Core/CPU	New	New	msp430dna.h, intrinsic support available
	Power	New	New	Develop new code using DriverLib APIs
	CS	New	New	Develop new code using DriverLib APIs
	Flash & RAM	New	New	Develop new code using DriverLib APIs
	NVIC			
	ISR			
	Intrinsics			
MSP430	Digital: T GPIO, A			
	ADC14			
	COMP_			
	AES, CR			
ARM	DMA	New	New	Use DriverLib APIs
	Timer32	New	New, similar to TivaC	Simple modules: use C-code, DriverLib, or CMSIS
	SysTick	New	New, similar to TivaC	Simple modules: use C-code, DriverLib, or CMSIS
	Application Code			Take care of data types (16-bit, 32-bit, native int, signs, etc.) Try to use explicit C99 types (uint32_t) if possible.

- Header file definitions, register names, bits compatible to MSP430
- Existing register-access code compatible on MSP432
- MSP430 DriverLib APIs also compatible on MSP432
- See interrupt/NVIC component below for specific interrupt changes

- New core system configuration code required.
- No need to develop code from scratch; take advantage of DriverLib APIs and

- Register access code available, but DriverLib APIs recommended for complex module such as DMA
- DriverLib APIs Similar to Tiva C DriverLib APIs
- CMSIS register definitions compatible with other ARM devices.

Porting | Code & SW

- Bare-metal
  - MSP430 modules 100% compatible
    - Register names & access for MSP430 peripherals retained
      - Existing MSP430 register-level C code compatible on MSP432
      - Minimize code migration from F5xx/6xx/FR5xx to MSP432
  - New modules → recommend DriverLib
  - Intrinsic support: msp430dna.h
    - Applicable MSP430 intrinsic functions carried over
  - Interrupt code migration
    - MSP430 interrupt vector usage supported in CCS
      - #pragma = __interrupt_vector
- DriverLib
  - MSP430 modules 100% compatible
  - ARM modules 100% compatible with TivaWare
- Other SW libraries
  - TI-RTOS, GrLib

Porting | Code & SW: NEW way

- Core Modules → New effort
- MSP430 Peripherals
  - DriverLib
  - Register-access code
- ARM Peripherals
- Interrupts
- Data types
- Intrinsics
- Assembly
- Other SW libraries
  - TI-RTOS, GrLib

Texas Instruments

Porting | Direct Register Access

MSP430

```
P1DIR |= 0x01;                          // P1.0 output
TA0CCTL0 = CCIE;                        // CCR0 interrupt enabled
TA0CCR0 = 50000;
TA0CTL = TASSEL_2 + MC_1 + TACLR;       // SMCLK, upmode, clear TAR

__bis_SR_register(LPM0_bits + GIE);     // Enter LPM0, enable interrupts
__no_operation();                       // For debugger
```

MSP432

```
P1DIR |= 0x01;                          // P1.0 output
TA0CCTL0 = CCIE;                        // TACCR0 interrupt enabled
TA0CCR0 = 50000;
TA0CTL = TASSEL_2 + MC_1 + TACLR;       // SMCLK, UP mode, clear TAR

__enable_interrupt();
__deepsleep();
```

TI Information – Selective Disclosure

Porting | DriverLib I2C Master Initialization

MSP430

```
//Initialize Master
USCI_B_I2C_masterInit(USCI_B0_BASE, USCI_B_I2C_CLOCKSOURCE_SMCLK,
        UCS_getSMCLK(UCS_BASE), USCI_B_I2C_SET_DATA_RATE_400KBPS);

//Specify slave address
USCI_B_I2C_setSlaveAddress(USCI_B0_BASE, SLAVE_ADDRESS);

//Set Master in receive mode
USCI_B_I2C_setMode(USCI_B0_BASE, USCI_B_I2C_RECEIVE_MODE);

//Enable I2C Module to start operations
USCI_B_I2C_enable(USCI_B0_BASE);
```

MSP432

```
//Initialize Master
USCI_B_I2C_masterInit(USCI_B0_BASE, USCI_B_I2C_CLOCKSOURCE_SMCLK,
        UCS_getSMCLK(UCS_BASE), USCI_B_I2C_SET_DATA_RATE_400KBPS);

//Specify slave address
USCI_B_I2C_setSlaveAddress(USCI_B0_BASE, SLAVE_ADDRESS);

//Set Master in receive mode
USCI_B_I2C_setMode(USCI_B0_BASE, USCI_B_I2C_RECEIVE_MODE);

//Enable I2C Module to start operations
USCI_B_I2C_enable(USCI_B0_BASE);
```

TI Information – Selective Disclosure

Porting | CMSIS Support

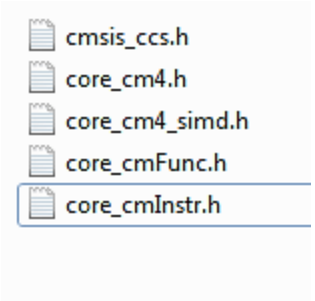
- CMSIS-style device header file
  - msp432_cmsis.h

```
//*****************************************************************************
// TA0 Registers
//*****************************************************************************
typedef struct {
  union {    /* TA0_CTL Register */
    __IO uint16_t reg;
    struct { /* TA0_CTL Control Bits */
      __IO uint16_t TAIFG                    :   1;  /* TimerA interrupt flag */
      __IO uint16_t TAIE                     :   1;  /* TimerA interrupt enable. This i
      __IO uint16_t TACLR                    :   1;  /* TimerA clear. Setting this bit
                                           /* count direction. The TACLR bit is automat
                                           /* . */
      __IO uint16_t RESERVED2                :   1;  /* Reserved */
      __IO uint16_t MC                       :   2;  /* Mode control. Setting MCx = 00
      __IO uint16_t ID                       :   2;  /* Input divider. These bits along
                                           /* input clock. */
      __IO uint16_t TASSEL                   :   2;  /* TimerA clock source select */
      __IO uint16_t RESERVED1                :   6;  /* Reserved */

    } bit;
  } CTL;
  union {    /* TA0_CCTL0 Register */
    __IO uint16_t reg;
    struct { /* TA0_CCTL0 Control Bits */
      __IO uint16_t CCIFG                    :   1;  /* Capture/compare interrupt flag
      __IO uint16_t COV                      :   1;  /* Capture overflow. This bit ind
                                           /* reset with software. */
      __IO uint16_t OUT                      :   1;  /* Output. For output mode 0, this
      __I  uint16_t CCI                      :   1;  /* Capture/compare input. The sel
      __IO uint16_t CCIE                     :   1;  /* Capture/compare interrupt enab
                                           /* corresponding CCIFG flag. */
      __IO uint16_t OUTMOD                   :   3;  /* Output mode. Modes 2, 3, 6, an
                                           /*   */
```

- CMSIS core header files
  - Supplied from ARM
  - Provide intrinsic/instruction support

- CMSIS DSP & Math library



Porting | MSP430 DNA & Intrinsic Support

- Enabled using msp430dna.h & CMSIS standard library
- Intrinsic functions:
  - __no_operation();
  - Low-power mode intrinsics
  - Stack pointer, registers, etc.
  - __delay_cycles(); → Supported in CCS

- Incompatible intrinsics are detected & recommended for MSP432 replacement
  - Detects all incompatibility
  - Guides developers through replacement & migration

Porting | Interrupts Declaration on MSP432

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                   */
    intDefaultHandler,                  /* 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               */
```

msp432_startup_ccs.c

ISR Handlers: stubs defined in table.
Treated as regular function, code in user's application

Option 2: MSP430 method
Use **#pragma vector**

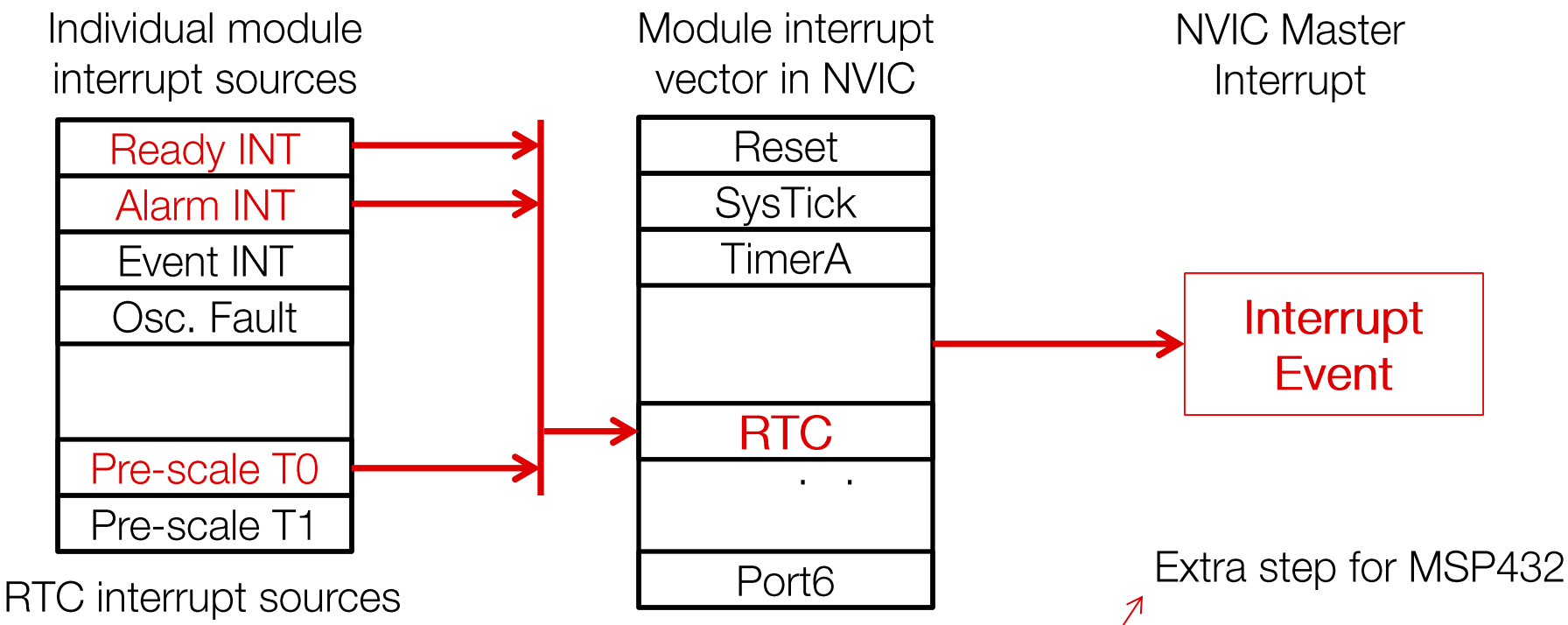
```
#pragma vector = USCI_B0_VECTOR
__interrupt void USCI_B0_ISR(void)
{
  switch(__even_in_range(UCB0IV,12))
  {
  case  0: break;
  ……
}

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

1. MSP430 code re-use
2. Interrupt vector & handler function defined together

Texas Instruments

Porting | Interrupts Usage on MSP432



1. Enable module interrupt sources

```
RTCPS1CTL |= RT1PSIE;
```

2. Enable module interrupt in NVIC

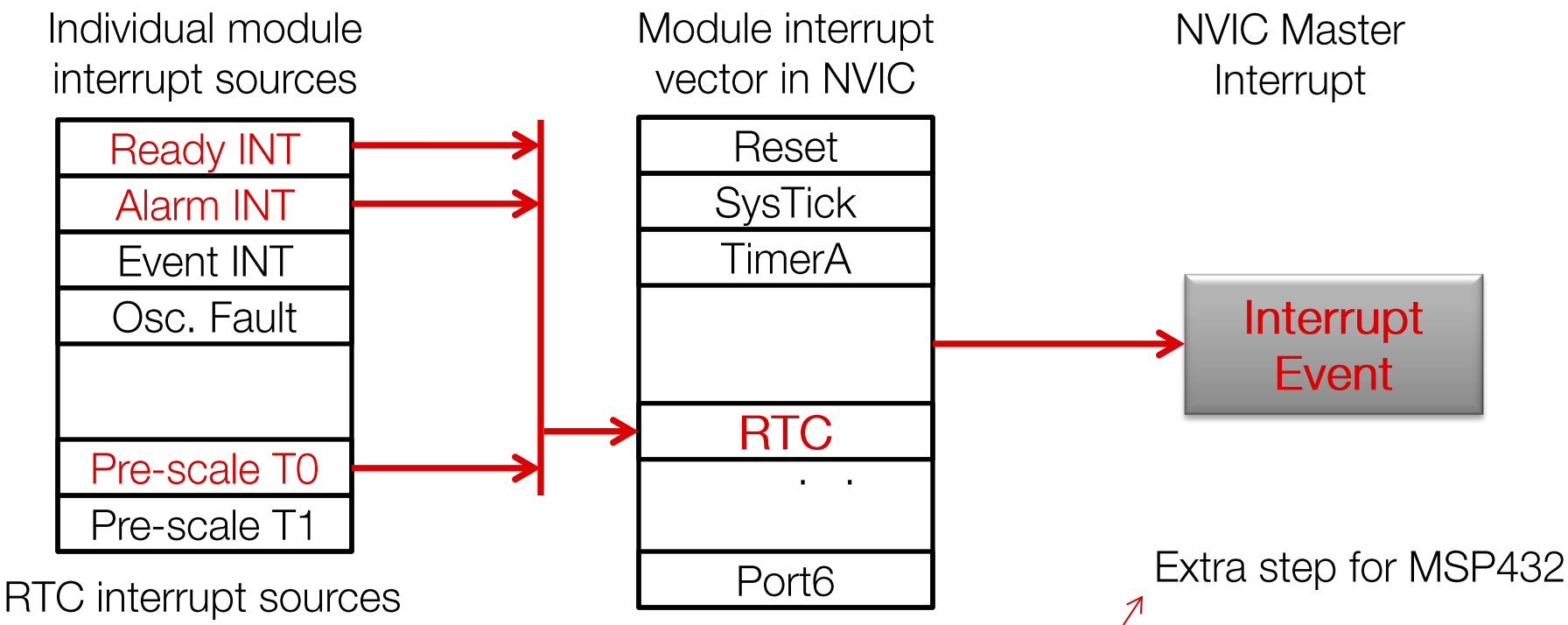
```
SCS_NVIC_ISER0 = INT_RTC_BIT;
```

3. Enable NVIC Master Interrupt

```
__enable_interrupt();
```


Texas Instruments

Porting | Interrupts Usage on MSP432



1. Enable module interrupt sources

```
RTC_enableInterrupt(
  RTC_CLOCK_READ_READY_INTERRUPT |
  RTC_CLOCK_ALARM_INTERRUPT |
  RTC_PRESCALE_TIMER1_INTERRUPT);
```

2. Enable module interrupt in NVIC

```
Interrupt_enableInterrupt(
INT_RTC);
```

3. Enable NVIC Master Interrupt

```
Interrupt_enableMaster();
```


Texas Instruments

Porting | Hardware: Power System

- Regulators
  - Some MSP430 families only have one regulator: LDO
  - MSP432 applications need to consider when to use LDO vs. DC-DC
- VCORE Levels
  - 2 VCore levels instead of 4 (5xx/6xx) or 1 (FRxx)
  - 0-24MHz vs. 24-48MHz frequency throttling
- LDO vs. DC-DC
  - Additional inductor required for DC-DC
- Simplified SVS

Texas Instruments

Porting | Hardware: Clock System

- MSP432 DCO= F5xx FLL + F2xx DCO
- DCO = jitter-free & calibrated 2xx + tunable 5xx
- External resistor for improved DCO accuracy
- HFXT up to 48MHz

Texas Instruments

Porting | Hardware: Memory

- Flash
  - Dual-bank allows for concurrent erase & execution
  - Consideration for larger sector size (512B → 4kB)
  - Individually protectable sectors (MSP432) for flexible mass erase

- RAM
  - Dynamically enable/retain SRAM → minimize current consumption
    - SW needs to be more aware of memory locations & allocations

Texas Instruments

Porting | Hardware: Other Improvements

- Similar peripheral usage/behavior
  - Majority of peripherals are retained & carried over
  - GPIO, Timer_A, eUSCI, RTC, COMP_E, ADC, etc.

- Improvements in new peripheral:
  - Certain modules received improvements while retaining majority of MSP430 functionalities.
    - ADC12 → ADC14
    - GPIO with high-drive
    - GPIO with glitch-filtering to suppress transient noise

Texas Instruments

Lab 6 | Platform Porting

Texas Instruments

Lab 6 | Platform Porting

- Port MSP430F5529 project (*lab6_platform_porting_msp430f5529*) to MSP432 (*lab6_platform_porting*) (incorporating different SW components.
- Attendees need to fill in "migrated" code in different subsets of the code
  - Modify the new startup_ccs.c file & insert the ISR handlers into the interrupt vector table.
  - Bare-Metal: port TimerA configuration code
  - DriverLib: port Clock, Power, UART code
  - CMSIS: a quick taste of ARM standard (Peripheral access & intrinsic)
  - Porting MSP430 Intrinsic to MSP432:
    - Go to LPM0/3 → __wfi()/__deep_sleep()
    - Delays → __delay_cycles() [Available soon]
  - Replacing MSP430 Interrupt with MSP432 interrupt usage

Porting Steps

1. Core module configuration
   - Set up Power & VCore
   - Set up Flash wait-state based on the system frequency
   - Set up DCO frequency
2. Peripheral code: 3 flavors of MSP432 code
   - Port register-access code for Timer_A modules
   - Port DriverLib APIs for ADC12 module to ADC14 module
     - Note the use of MAP_ADC14_xxxxx APIs instead of ADC14_xxxxx
   - Port RTC module mode using CMSIS
3. Interrupt-related configuration code
   - Module interrupt enable instructions to be done using both register-access & DriverLib APIs
   - Enable module interrupt on NVIC to be done
     - Using DriverLib API: Interrupt_enableInterrupt for ADC14
     - Using register-access: SCS_NVIC_ISERx register for RTC
   - Enable NVIC master interrupt (equivalent to GIE bit on MSP430)
     - Using DriverLib API
     - Using MSP432 intrinsic function
4. Add ISR stubs to interrupt vector table

Porting Steps

5. Core, CPU, and system-related functions
   - How to go to LPM0 or sleep
   - How to go to LPM3 or deep sleep
   - Control when/how to wake up to active from ISR
   - Control when/how to go back to sleep from ISR
7. Tie all pieces together & debug
   - Anything else?

Getting Started with MSP432

Getting Started | Code Composer Studio as MSP432 Portal

- One-stop shop for all MSP432 related information
- MSP430-DNA enabled
- Ability to switch between TI & GCC Compilers
- MSPWare (Libraries, code examples)
- ULP Advisor
- Compiler Optimizer Assistant
- Memory Allocation
- Compiler Enhancements
- EnergyTrace+

Getting Started | Leverage ARM & Cortex-M Eco-system

CMSIS standard

- CMSIS header files & definitions
- CMSIS libraries
- CMSIS SVD & package for 3rd-parties ensuring similar experience across IDE platforms
- CMSIS-DAP debugger interface

ARM-enabled IDEs

- CCS
- IAR
- Keil
- GCC

Cortex-M Debuggers

Thank you!

Texas Instruments