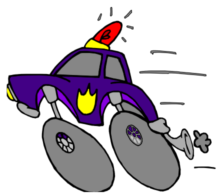
MSP430FR57xx Family
MSP430 with Embedded FRAM

FRAM – Technology Attributes

Photo: forums.wow-europe.com

- **Non-Volatile –** retains data without power



- **Fast Write / Update –** RAM like performance**.** Up to ~ 50ns/byte access times today (> **1000x** faster than Flash/EEPROM)

- **Low Power - Needs 1.5V** to write compared to **> 10-14V** for Flash/EEPROM → no charge pump

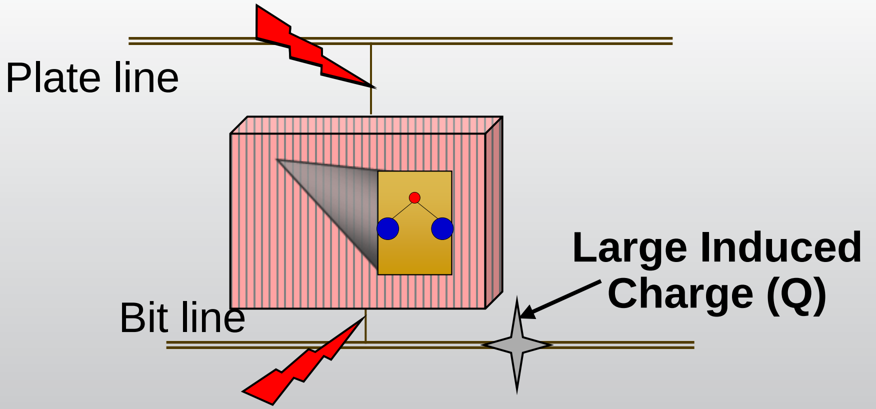
RAMTRON
Automotive F-RAM Memory

- **Superior Data Reliability - 'Write Guarantee'** in case of power loss and > **100 Trillion** read/write cycles

Understanding FRAM Technology

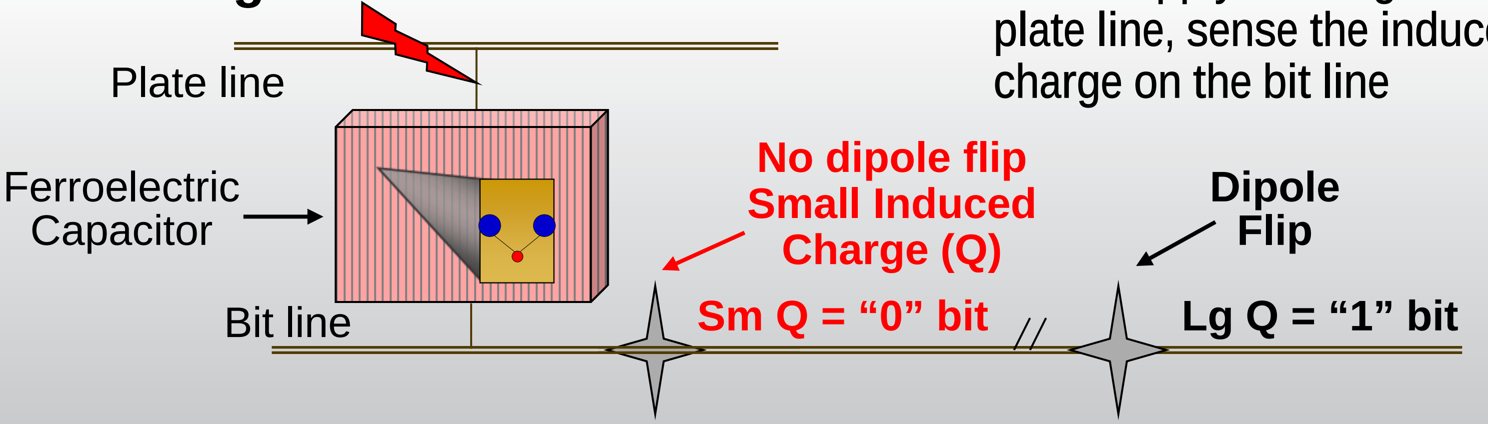
Programming Data to FRAM

WRITE: Apply voltage to plate line (write ‘0’) or bit line (write ‘1’)



Reading Data from FRAM

READ: Apply a voltage to the plate line, sense the induced charge on the bit line



Target Applications

- Data logging, remote sensor applications (High Write endurance, Fast writes)
- Digital rights management (High Write Endurance – need >10M write cycles)
- Battery powered consumer/mobile Electronics (low power)
- Energy harvesting, especially Wireless (Low Power & Fast Memory Access, especially Writes)
- Battery Backed SRAM Replacement (Non- Volatility, High Write Endurance, Low power, Fast Writes)

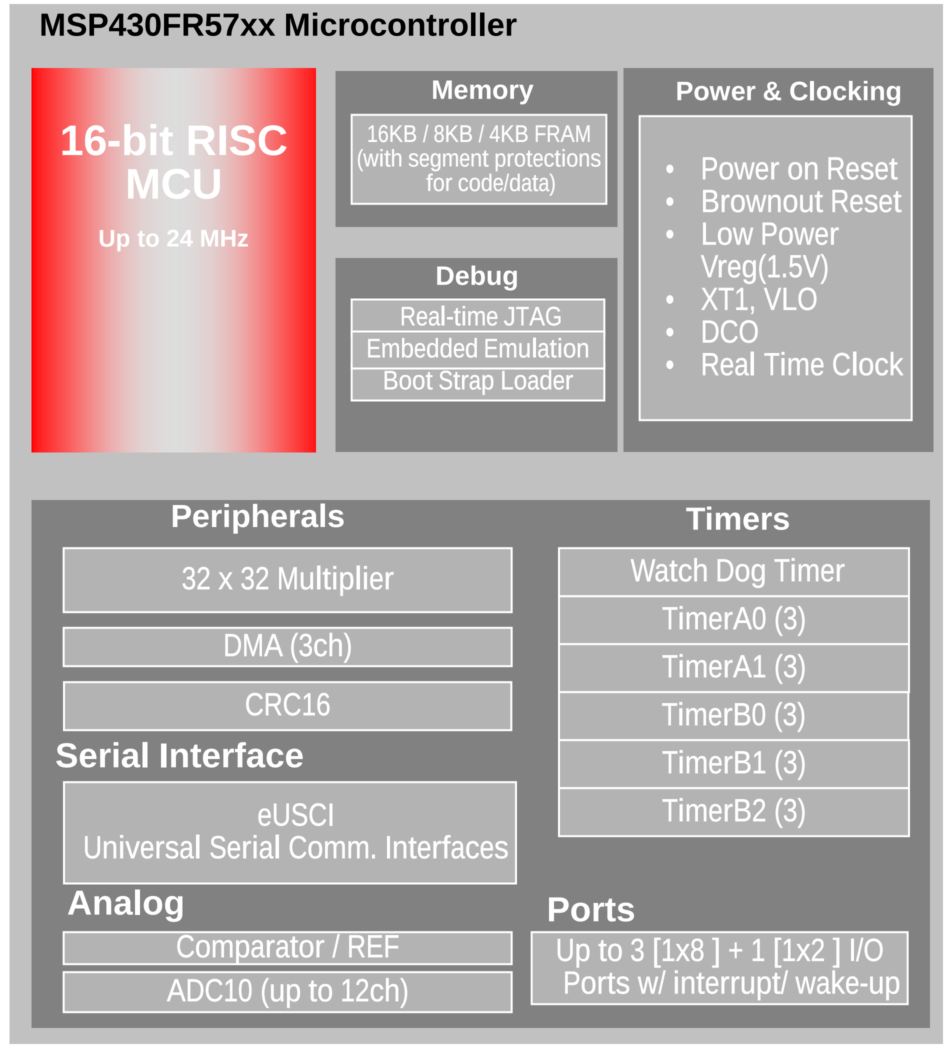
All-in-one: FRAM MCU delivers max benefits

	FRAM	SRAM	EEPROM	Flash
Non-volatile Retains data without power	Yes	No	Yes	Yes
Write speeds	10ms	<10ms	2secs	1 sec
Average active Power [µA/MHz]	110	<60	50mA+	230
Write endurance	100 Trillion+	Unlimited	100,000	10,000
Dynamic Bit-wise programmable	Yes	Yes	No	No
Unified memory Flexible code and data partitioning	Yes	No	No	No

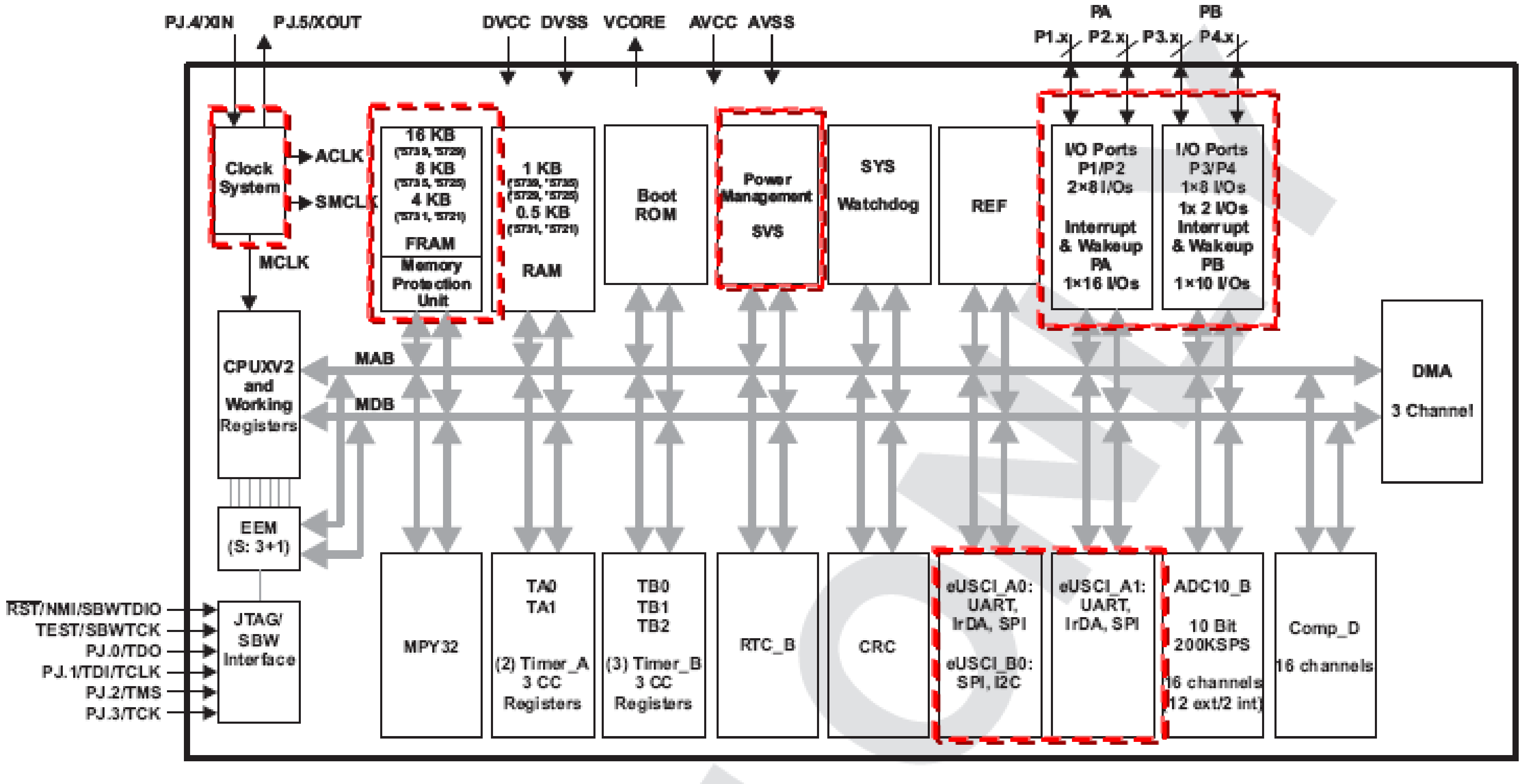
Data is representative of embedded memory performance within device

First ULP, Embedded FRAM MCU – FR5739

- Performance
  - Up to 24MHz (FRAM access @ 8MHz)
- Power Numbers
  - Active Mode: 110 µA/MHz avg.@ 8MHz
  - RTC mode (LPM3.5): ~1.5 µA
  - Standby Mode (LPM3): <7 µA
  - Shutdown Mode (LPM4.5): ~0.3 µA
- Flexible Unified Memory
  - 16/8/4 KB FRAM versions with program code / data memory partitioning
- Package
  - 24/40-Pin QFN, 28, 38-Pin TSSOP
  - Temp Range -40ºC to 85ºC

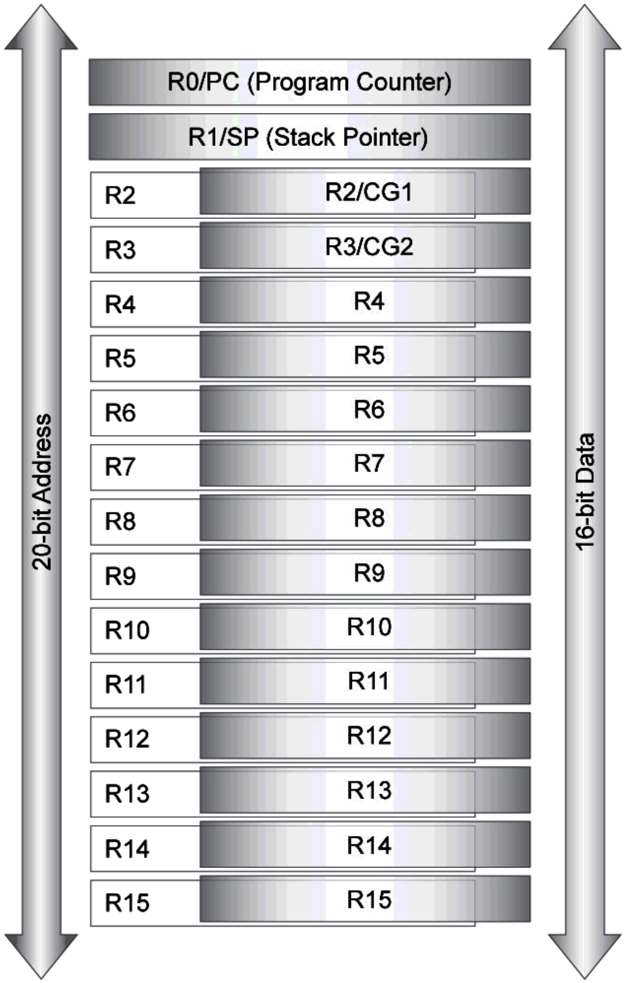


MSP430FR5739 Block Diagram



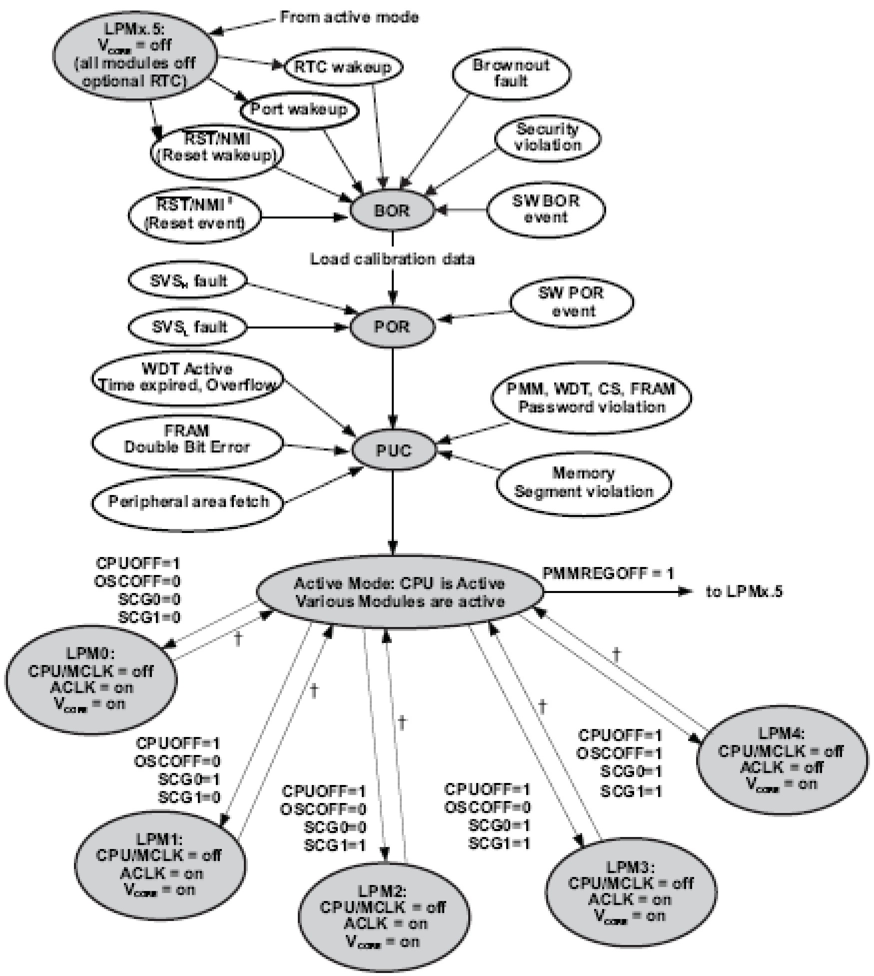
FR57xx Architecture & Core Peripherals

MSP430xv2 Orthogonal CPU



- No changes from the F5xx CPU!
- C-compiler friendly
- Memory address access up to 1MB
- CPU registers 20-bit wide
- Address-word instructions
- Direct 20-bit CPU register access
- Atomic (memory-to-memory) instructions
- Instruction compatible w/previous CPU
- Cycle count optimization for certain instructions

Operating Modes



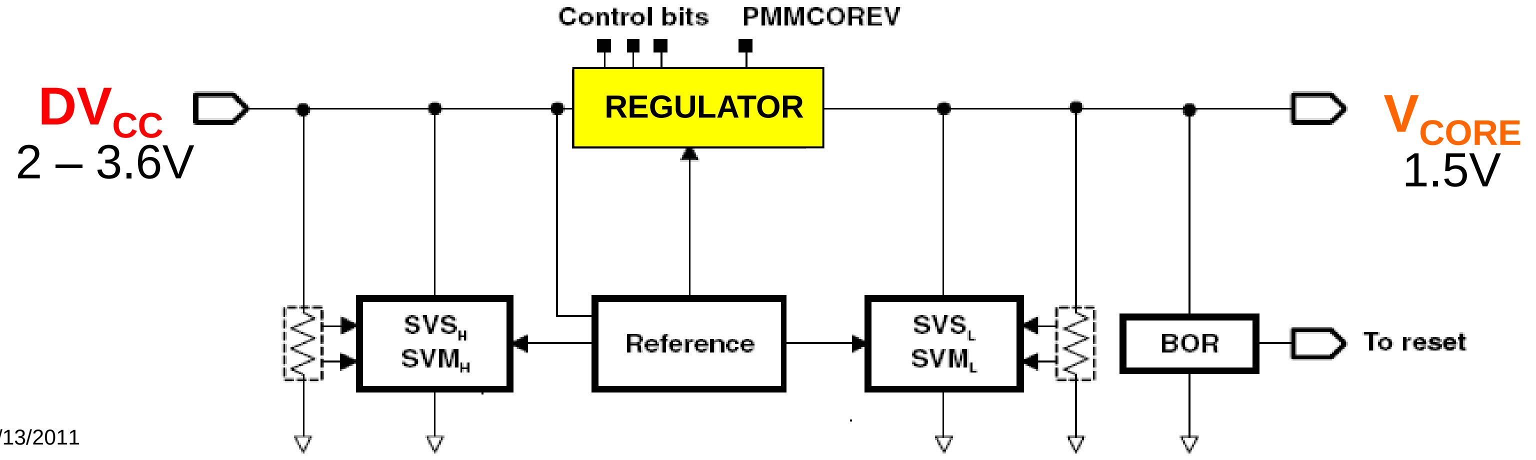
- **Active Mode – 110 µA/MHz!**
  - CPU active
  - Fast Peripherals Enabled
  - 32 kHz Peripherals Enabled - RTC
- LPM0 – 170 µA
  - CPU disabled, Fast Peripherals Enabled
  - Fast Wake up
  - 32 kHz Peripherals Enabled – RTC
- LPM3 – 6.4 µA
  - CPU disabled, Fast Peripherals Disabled
  - Slow wake up
  - 32 kHz Peripherals Enabled
    - RTC, Watchdog & SVS protection
- LPM4 – 5.9 µA
  - All clocks disabled
  - Wake on interrupt
- LPM3.5 – 1.5 µA
  - Regulator & all clocks disabled
  - Complete FRAM retention
  - BOR on nRST/NMI or Port I/O or RTC
- LPM4.5 – 0.32 µA

LPM & Wakeup Time Comparison

Parameter	F2xx	F5xx	FR57xx
LPM0-LPM4	Yes	Yes	**Yes**
LPMx.5	No	Yes	**Yes**
$t_{WAKEUP-LPM0}$	1µs	6µs	**1µs**
$t_{WAKEUP-LPM1,2}$	1µs	6µs	**11µs**
$t_{WAKEUP-LPM3,4}$	1µs	6µs/ 150µs	**100µs**
$t_{WAKEUP-LPMX.5}$	N/A	2000µs	**700µs**

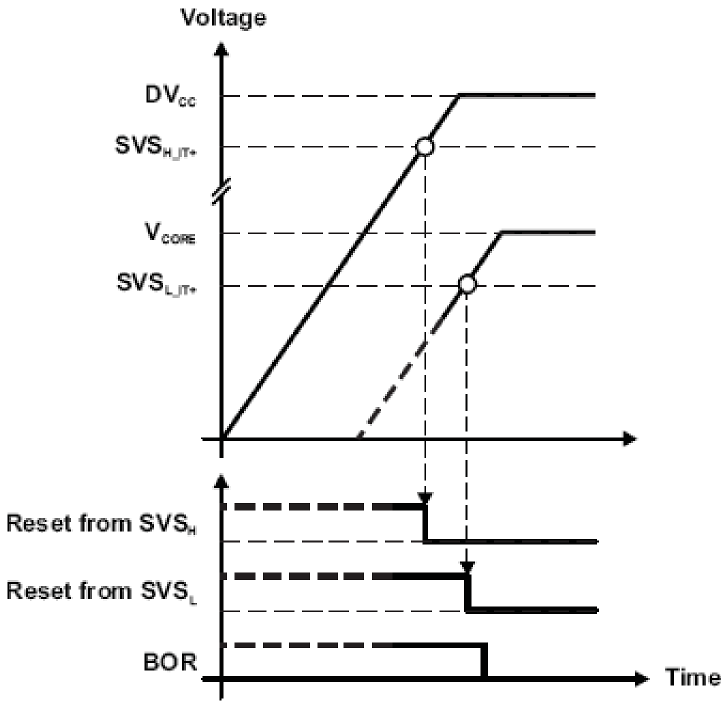
PMM & Core Voltage

- What is V_{CORE}?
  - Integrated LDO provides a regulated voltage
  - V_{CORE} powers digital core (CPU, memory, digital modules)
- Is this any different from the F5xx family?
  - Yes, FR57xx has only one core level [1.5V]
- Any recommendations?
  - DO put a 470nF cap on the V_{CORE} pin
  - DO NOT load the V_{CORE} pin externally
  - DO NOT connect the V_{CORE} pin to any other pins on the device



Supply Voltage Supervision (SVS)

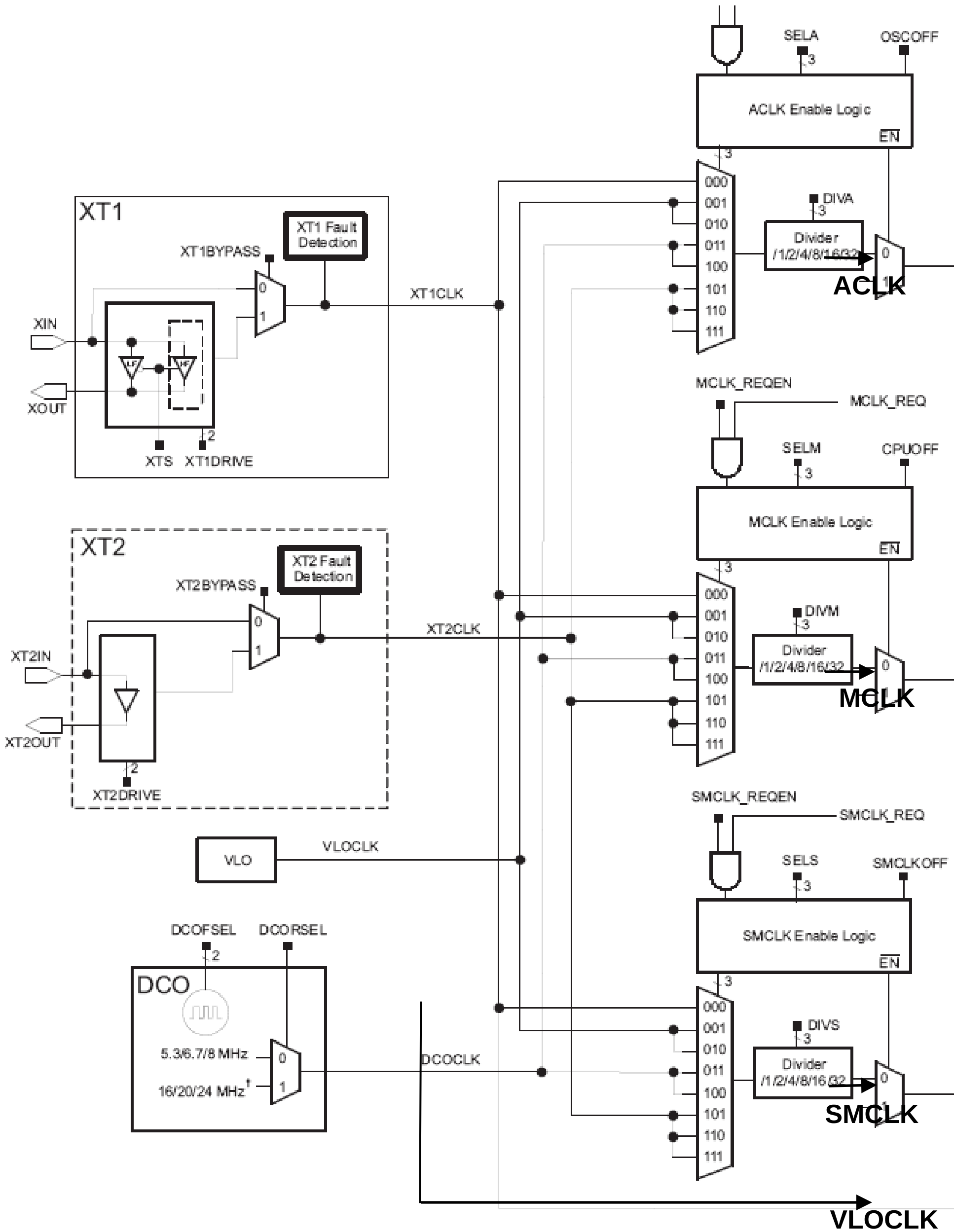
- Supply voltage supervision highly simplified compared to F5xx family
- Individually enabled for high (supply)/ low (core) sides
- Hard-coded threshold levels
- Device reset tracks with SVSH
- SVSH
  - Enabled in all modes, cannot be disabled
  - Disabled in LPM4.5
- SVSL
  - Enabled in active, LPM0, cannot be disabled
  - Can be disabled in LPM1,2 (default enabled)
  - Disabled in LPM3,4,x.5



PMM Action at Device Power-up

Clock System (CS)

- Five independent clock sources
  - Low Freq
    - LFXT1 32768 Hz crystal
    - VLO 10 kHz
  - High Freq
    - XT1 4 – 24 MHz crystal
    - XT2 4 – 24 MHz crystal
    - DCO Specific CAL range
- **Default DCO = 8MHz**
  - **MCLK = DCO/8 = 1MHz**
- ACLK / SMCLK / MCLK tree is fully orthogonal
- MODOSC provided to modules
  - ADC10
- Failsafe
  - XT1LF: VLO
  - XT1HF or XT2: MODOSC



CS: Digitally Controlled Oscillator

- Six frequency settings
- Not programmable (add comment)
- Factory Calibrated

  ±2% accuracy from 0-50C

  ±3.5% accuracy from -40 to 85C

DCOFSEL	Nominal DCO frequency, MHz	
	DCORSEL = 0	DCORSEL = 1 +
00, 10	5.33	16
01	6.67	20
11	8	24

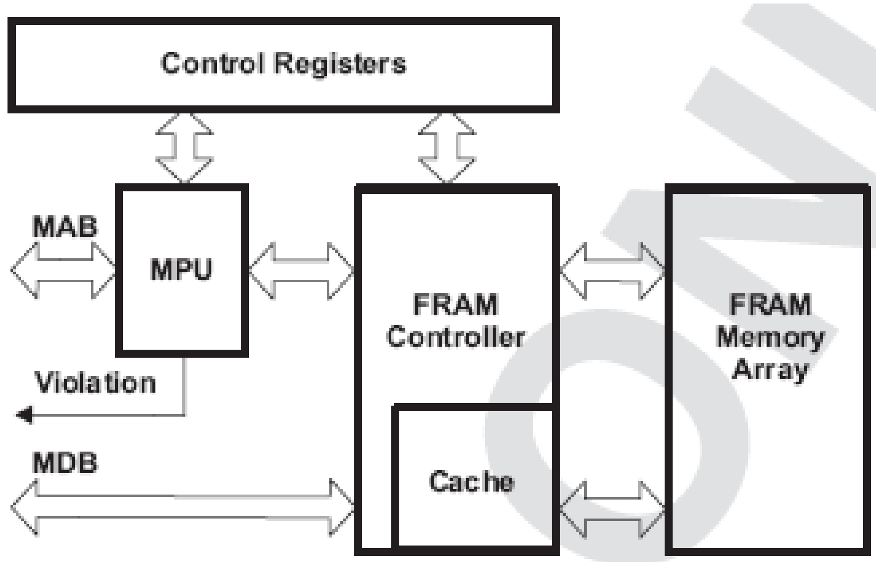
DCO Frequency Selection

+*Higher frequency ranges for FR573x family only*

FRAM Controller (FRCTL)

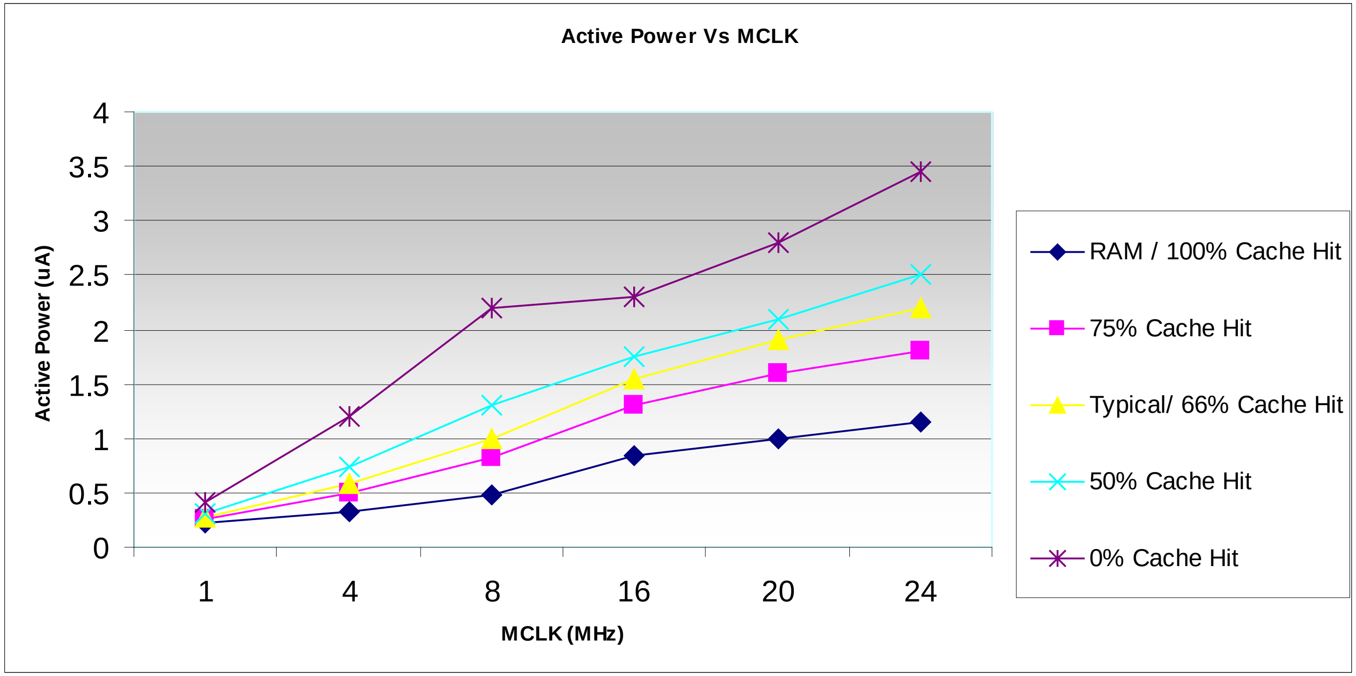
Functions of FRCTL:

- FRAM reads and writes like standard RAM (but)
- Read/Write frequency $\leq$ 8MHz
- For MCLK > 8MHz, wait states activated
  - Manual or automatic
- Seamless and transparent integration of cache
- Error checking and correction (ECC) built into FRAM read/write cycle



FRAM and the Cache

- Built-in 2 way 4-word cache; transparent to the user, always enabled
- Cache helps:
  - Lower power by executing from SRAM
  - Increase throughput overcoming the 8MHz limit set for FRAM accesses
  - Increase endurance specifically for frequently accessed FRAM locations e.g. short loops (JMP$)



So far we've covered…

- FRAM Technology Attributes
- Introduction to the MSP430FR57xx Family
- FR57xx CPU, Operating Modes & Wake up times
- Core Module Overview
  - PMM
  - SVS
  - CS, DCO
  - FRCTL
  - Impact of Cache in the system

Now we are ready for a lab!

Lab 1A

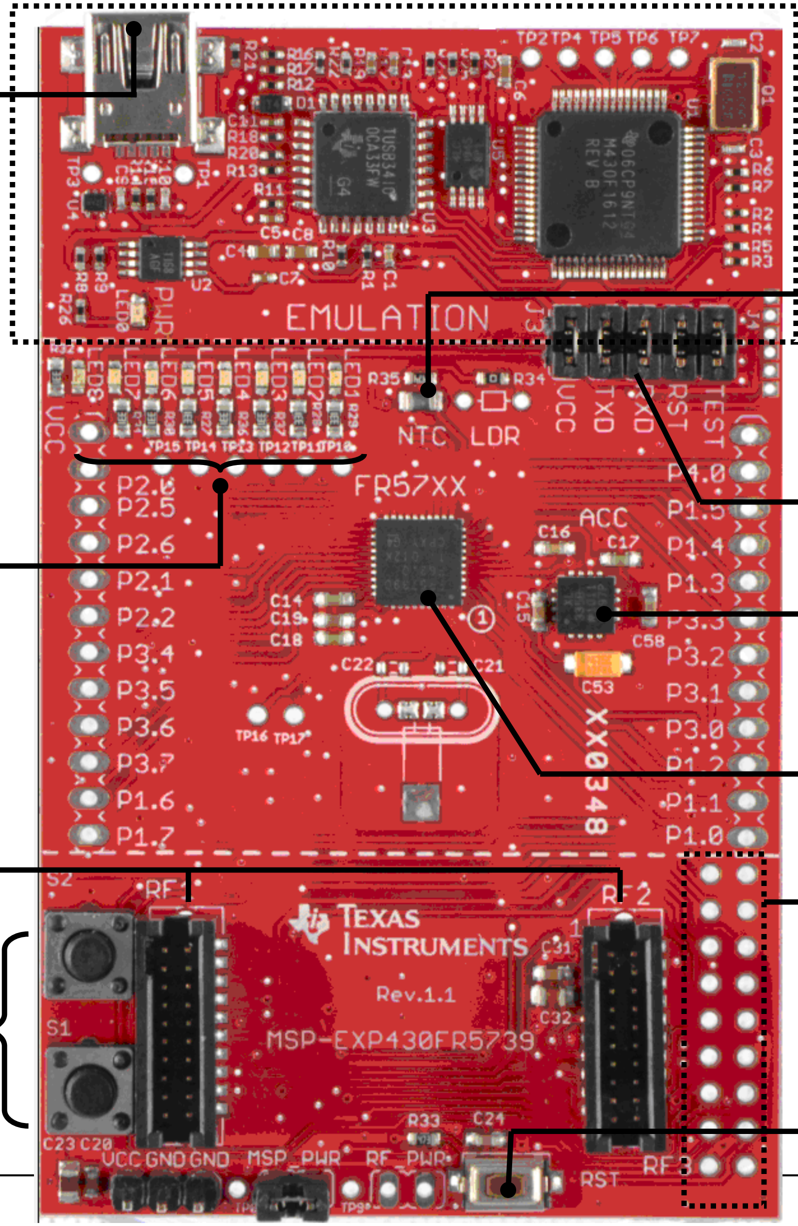
Goals:

1) Unboxing the FRAM Experimenter’s Board
2) FRAM EXP feature overview
3) FRAM EXP Graphical User Interface and User Experience

Obtaining Lab Software

- Software for this lab can be obtained from the MSP430 FRAM Training Wiki
- The link is: http://processors.wiki.ti.com/index.php/MSP430_FR57xx_3_HR_Lab
- Download the zip file
  - The folder 'FR-EXP User Experience' contains the FRAM EXP User Experience Code and the Graphical User Interface used in Lab1
  - The folder 'LabWorkspace' contains the CCS workspace location and the source files for executing the labs 2 & 3
- By default the board is programmed with the User Experience code

MSP-EXP430FR5739 Experimenter's Board

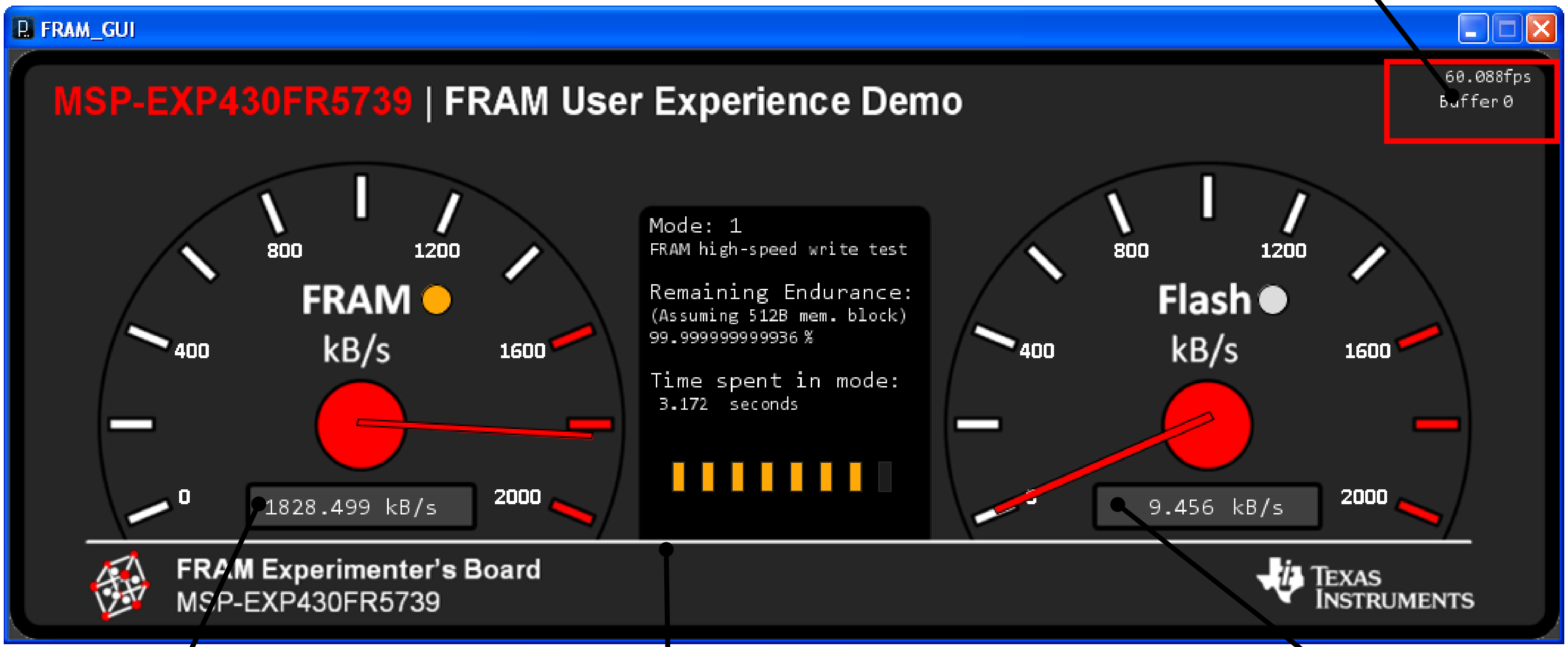


EXP Board: Out-of-the-box Experience

- Four Demo Modes:
  - High Speed FRAM write mode
  - Flash Emulation mode
  - Accelerometer sample and store mode
  - Temperature sensor sample and store mode
- Use S1 to select a mode and S2 to enter
- When inside a mode, toggle S2 to turn display/ UART on/off
- To exit and return to menu press S1
- Demo package comes with a graphical user interface
  - …OBE\ FRAM_GUI\FRAM_GUI.exe

EXP Board: Out-of-the-box Experience

MCUTTTFRAM \ FRAM_GUI\FRAM_GUI.exe



PC Debug data

FRAM Write speed

LED Tracking

Emulated flash Write speed

EXP Board: Out-of-the-box Experience

Step 1: Double click to open FRAM_GUI.exe

Step 2: Plug in EXP Board to computer

Step 3: Select Mode 1 on EXP board. Observe FRAM speed in kB/s

Step 4: Select Mode 2. Observe emulated flash speed in kB/s

Step 5: Select Mode3. Place the board on a level surface to calibrate the board before entering Mode 3.

Step 6: Observe the GUI track with the tilt of the board as the FR5739 records sample data on-the-fly

Step 7: Select Mode 4. Observe LED sequence based on increasing/ decreasing temperature

Lab 1B

Goals:

1) Setting up a CCS Project

2) Measure active power for different system frequencies

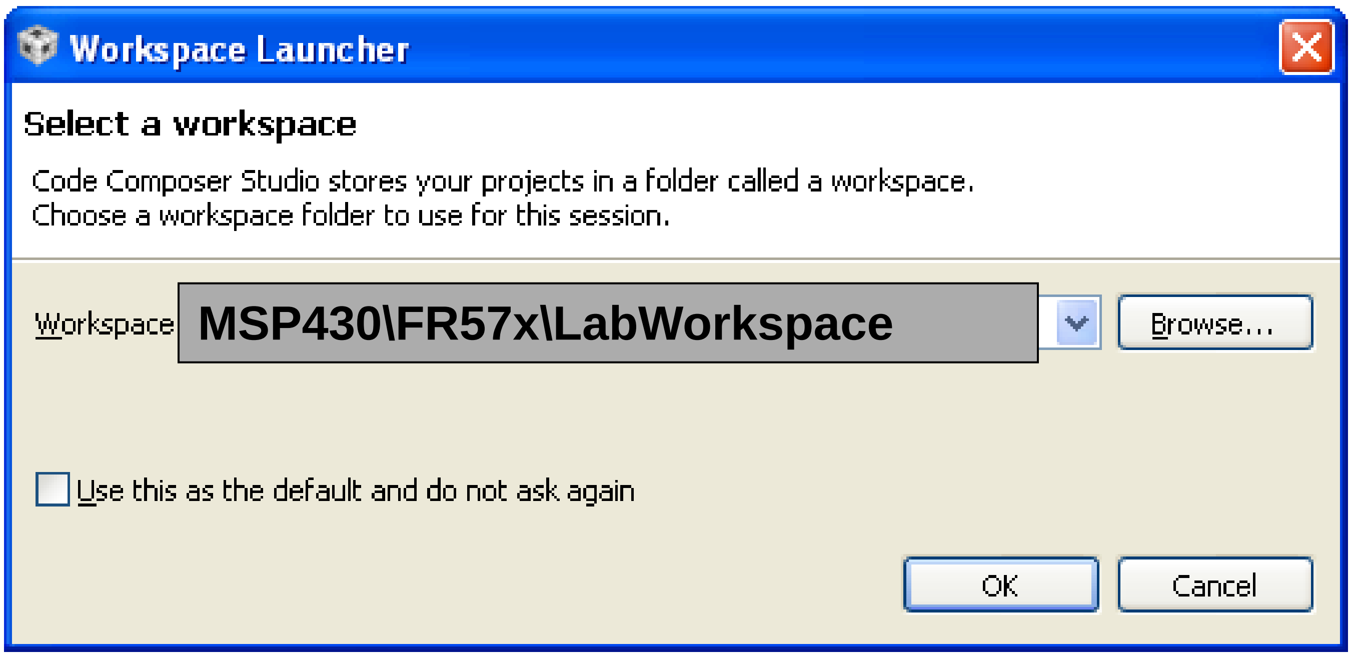
3) Understand the impact of cache on active power

Obtaining Lab Software

- Software for this lab can be obtained from the MSP430 FRAM Training Wiki
- The link is: http://processors.wiki.ti.com/index.php/MSP430_FR57xx_3_HR_Lab
- Download the zip file
  - The folder 'FR-EXP User Experience' contains the FRAM EXP User Experience Code and the Graphical User Interface used in Lab1
  - The folder 'LabWorkspace' contains the CCS workspace location and the source files for executing the labs 2 & 3
- By default the board is programmed with the User Experience code
- If this needs to be re-installed after the labs use the batch file in the folder 'User Experience Programmer'

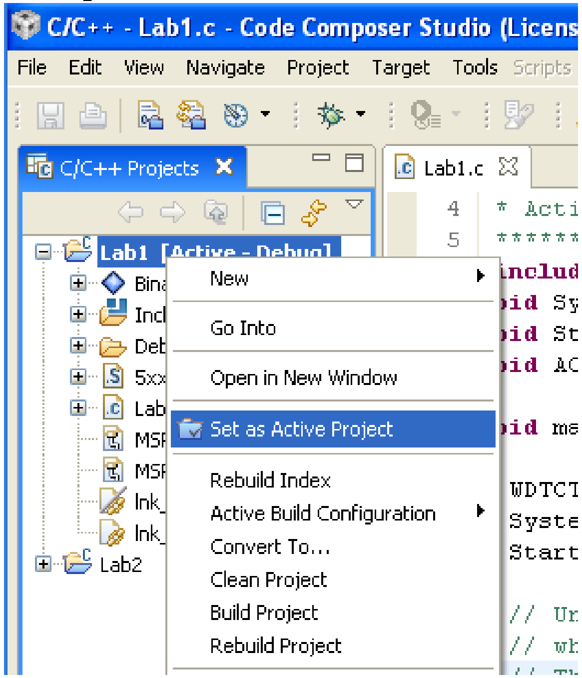
Setting up a Project using CCS

- Open Code Composer Studio (V4.3.2 or later)
- Select **MSP430\FR57x\LabWorkspace** as the workspace location



Setting up a Project using CCS

- Import two projects using Project → Import existing…
- Select the folder ‘LabWorkspace’ as the root dir for the projects
- Ensure Lab1 is marked as [Active-Debug]
- If not, right click on Lab1 and use ‘Set as Active Project’



Using while(1)/ JMP$ to Measure Power

Lab Notes:

- Lab1.c is setup to initialize the board, execute the LED startup sequence
- Ensure that while(1); loop in main() is included
- Build and download active project [Target → Debug Active Project]
- Execute the code [Target → Run]
- Terminate Debug Session [Target → Terminate All]

And the power number is…

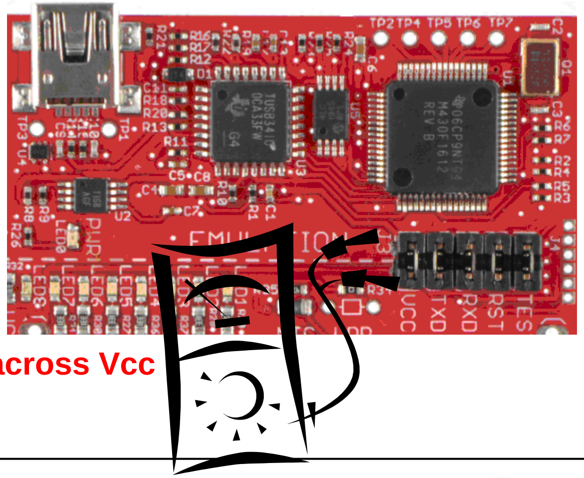
Lab Notes:

- Measure power across V_{CC} jumper of the eZFet
- MCLK = DCO = 8MHz; Meter reads <600µA or ~75µA/MHz

Observations:

- Single word opcode (JMP$) → Code execution is completely within the cache (SRAM)
- Hence the low active power!

Use USB for Power



Connect meter across Vcc

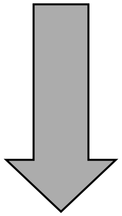
A More Realistic Scenario

- Function Active_mode_test() = combination of RAM, FRAM access + different addressing modes
- Closer to typical application use-case
- Use this function to measure 'real world' active power
- Comment out the while(1); loop
- Include ACTIVE_MODE_TEST() function call
- Rebuild Project
- Download & execute the code, terminate debug session

Note: Remember to reconnect the jumper to program the target or leave the meter ON

Source Code Snapshot

```
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                         // Stop WDT
  dummyvar++;
  SystemInit();                                     // Init the Board
  StartUpSequence();                                // Light up LEDs

  // Uncomment the line below to execute fully from SRAM
  // while(1);
  // This function exercises a mix of RAM and FRAM accesses
  ACTIVE_MODE_TEST();
}
```



Ensure that this function call is included

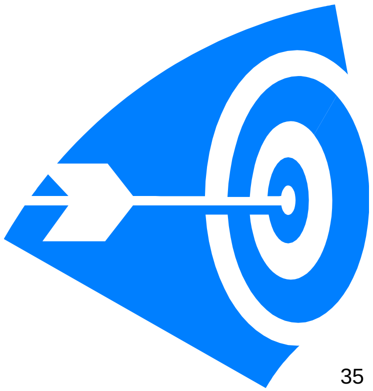
And now we measure…

Lab Notes

- MCLK = DCO = 8MHz
- Meter reads <750µA or <<100µA/MHz

Observations:

- As # of cache misses increase, active power increases
- Cache hit/miss ratio is completely application dependent
- Tighter, shorter loops = fewer cache misses



f_{SYSTEM} vs. Active Power

LAB1C

- Measure Power with different system clock frequencies
- Use MCLK = 16MHz and/or MCLK = 24MHz
- Set CSCTL1 registers → DCORSEL, DCOFSELx bits according to table below
- Follow previously provided instructions for code download

DCOFSEL	Nominal DCO frequency, MHz	
	DCORSEL = 0	DCORSEL = 1
00, 10	5.33	16
01	6.67	20
11	8	24

Texas Instruments

f_{SYSTEM} vs. Active Power

Lab Notes

- Verify increased system clock by speed of startup sequence
- Active Power @ 16MHz <1.3mA
- Active Power @ 24MHz < 2mA

Checklist:

✓ Measure active power @ 8MHz

✓ Setup DCO for 16MHz and 24MHz

✓ Compare active power numbers for 8,16, 24MHz

FR57xx Peripheral Additions & Enhancements

eUSCI_A: UART

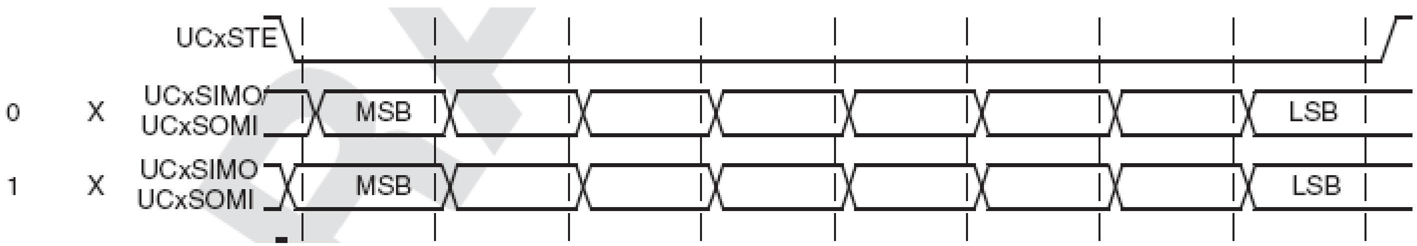
NEW!

- 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

eUSCI_A: 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 9MHz @ 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
- ACK/NACK selectable in software

eUSCI_B: I2C

Multiple Slave Addresses

- Support for four slaves in hardware
- 4 unique slave address registers: UCBxI2COAx
- Each slave address has a corresponding UCOAEN
- Independent interrupt vector pairs for TX and RX flags
- Shared status flags
- Dedicated DMA channels
- Example application: EEPROM + sensor

```
UCB0I2COA0 = 0x48; // EEPROM
UCB0I2COA1 = 0x40; // ADC
#pragma vector = USCI_B0_VECTOR
__interrupt void USCI_B0_ISR(void)
{
  switch()
  {
  case  0: break;
  case  2: break;
   …….
   case 20: // UTXIFG0 EEPROM TX
   case 22: // URXIFG0 EEPROM RX
   case 24: // UTXIFG1 ADC TX
   case 26: // URXIFG1 RX
   ……..
   default: break;
   }
}
```

eUSCI_B: I2C

Clock Low Timeout

- SCL being held low for a time> timeout interval causes flag to be set
- Interval timer based on MODOSC
- 3 selectable intervals ~25, 30, 35ms
- Interrupt: UCCLTOIE
- In LPMs high power LDO is automatically requested
- Available for both master and slave
- User is required to determine post-timeout activity such as reset
- Allows for SMBus compatibility without using a timer resource
- Can be leveraged for hot-plug issues

```
UCB0CTLW1 |= UCCLTO_2; // 25ms
UCB0IE |= UCCLTOIE;
#pragma vector = USCI_B0_VECTOR
__interrupt void USCI_B0_ISR(void)
{
  switch()
  {
  case  0: break;
  case  2: break;
   …….
   case 28: // clock low timeout
   UCB0CTL0 |= UCSWRST;
   UCB0CTL0 &= ~UCSWRST;
   break;
  }
}
```

eUSCI_B: I2C

Byte Counter & Auto Stop

- RX and TX bytes are counted in hardware
- The counter increments for every byte that is *on the bus*
- Available in master (active) and slave (passive) mode
- In master mode when used with auto stop – eliminates the need for software counters.
- Master sends Stop condition when BCNT threshold is hit

```
// Master TX Mode
UCB0CTLW1 |= UCASTP_2; //
UCB0TBCNT |= 0x05; // 5 bytes
#pragma vector = USCI_B0_VECTOR
__interrupt void USCI_B0_ISR(void)
{
  switch()
  {
  case  0: break;
    …….
   case 20: // UTXIFG0
   UCB0TXBUF = *Data_ptr;
   Data_ptr++;
   break;
  }
}
```

eUSCI_B: I2C

Early Transmit Interrupt

- USCI module clock stretches in TX mode if TX ISR is not serviced immediately
- eUSCI offers a preload feature
- TXBUF is loaded on detection of start edge prior to address compare
- Software must take care of the unloading in case of an address mismatch.

```
// Master TX Mode
UCB0CTLW1 |= UCETXINT; //
#pragma vector = USCI_B0_VECTOR
__interrupt void USCI_B0_ISR(void)
{
  switch()
  {
  case  0: break;
    …….
   case 20: // UTXIFG0
   UCB0TXBUF = *Data_ptr;
   Data_ptr++;
   break;
  }
}
```

eUSCI_B: I2C Migration Considerations

- HW clear of interrupt flags no longer available
  - USCI_B has 4 sets of flags with associated clearing events
  - Customers who have previously used the USCI will be assume this is still available (→ Migration document)
  - TXIFG cleared by NACK
    - In master mode NACKIFG can be used to clear last TXIFG
    - In slave mode STPIFG can be used. TXIFG could likely be already serviced and user needs to ensure data pointers are re-adjusted
  - STPIFG ↔ STTIFG
    - Needs to be included by user in S/W
  - NACKIFG cleared by STP
    - master mode only
    - NACKIE needs to be enabled if clearing is needed (no STPIFG in master mode)

ADC10_B

Feature Enhancements

- Significant power savings
  - 150µA Vs 1.2mA on F2xx
- Up to 200ksps
- REF – unique module
  - 1.5V, 2V and 2.5V
- DTC replaced by DMA
- Up to 12 external input channels
- Window Comparator
  - Hi, low and middle interrupts

```
// Configure Thresholds
ADC10HI = High_Threshold;
ADC10LO = Low_Threshold;
#pragma vector = ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  switch()
  {
   …….
   case 6: // ADC10MEM > ADC10HI?
    //…
   break;
  case 8: // ADC10MEM < ADC10LO?
    //…
   break;
 case 10:
    // ADC10HI <ADC10MEM < ADC10LO?
    //…
   break;
}}
```

RTC_B and Comp_D

RTC_B

- Calendar mode only
- LFXT1 32768Hz required
- Advanced interrupt capability – alarms, OF fault, RTCREADY and RTCEV
- Selectable BCD format
- Calibration
- Multiple Alarms
- Operation in LPM3.5

COMP_D

- Interrupt driven for low power
- Uses the REF module like ADC10_B
- Up to 15 external input channels
- Software selectable RC filter
- Selectable reference voltage generator
- Voltage Hysteresis generator

JTAG and BSL

JTAG

- Security can be achieved by:
- Fuse is in software

1) JTAG lock and unlock
   - Access granted only if tool chain supplies correct password
2) JTAG fuse blow
   - Access only via BSL if password is know
   - JTAG can be re-enabled via BSL
3) JTAG fuse blow + BSL disable
   - No further access to device is possible

BSL

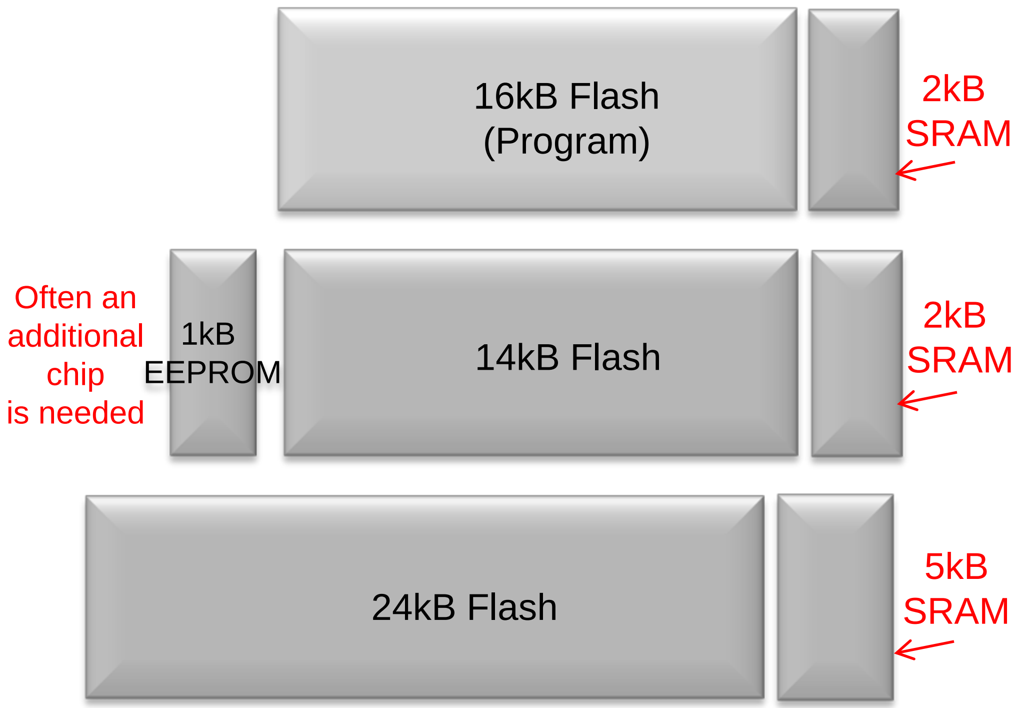
- Similar to F5xx BSL but
- Code in Boot ROM – cannot be modified
- Peripheral Interface: HW UART
- BSL Entry and signature same as F5xx

Using FRAM on the FR57xx

Unified Memory

Before FRAM

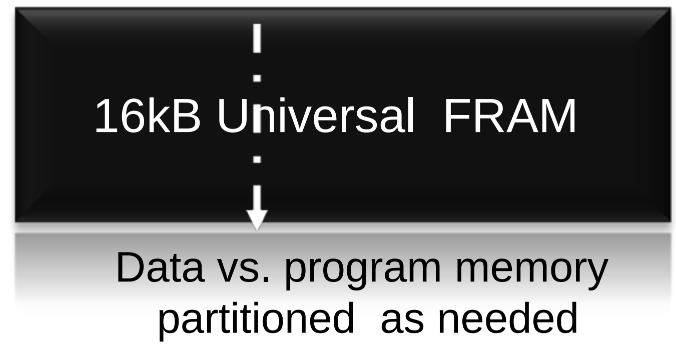
Multiple device variants may be required



To get more SRAM you may have to buy more FLASH ROM

With FRAM

One device supporting multiple options "slide the bar as needed"



- Easier, simpler inventory management
- Lower cost of issuance / ownership
- Faster time to market for memory modifications

Setting Up Code and Data Memory

- Let's analyze the linker command file for this device in CCS
- Open lnk_msp430fr5739.cmd from the project Lab1
- Study lab1.map from the project (located in the Debug folder) for RAM/FRAM usage

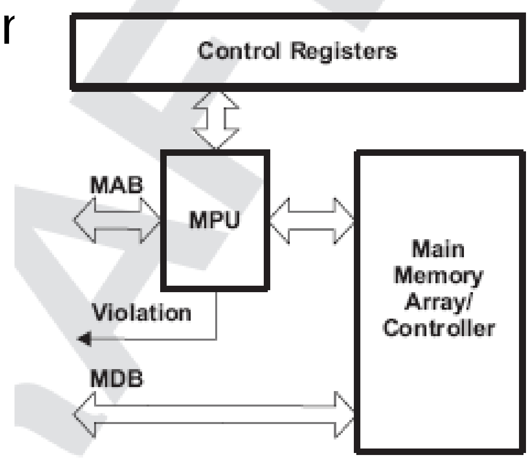
```
GROUP(ALL_FRAM)
{
   GROUP(READ_WRITE_MEMORY): ALIGN(0x0200) RUN_START(fram_rw_start)
   {
      .cio    : {}                  /* C I/O BUFFER                    */
      .sysmem : {}                  /* DYNAMIC MEMORY ALLOCATION AREA  */
   }

   GROUP(READ_ONLY_MEMORY): ALIGN(0x0200) RUN_START(fram_ro_start)
   {
      .cinit  : {}                  /* INITIALIZATION TABLES           */
      .pinit  : {}                  /* C++ CONSTRUCTOR TABLES          */
      .const  : {}                  /* CONSTANT DATA                   */
   }

   GROUP(EXECUTABLE_MEMORY): ALIGN(0x0200) RUN_START(fram_rx_start)
   {
      .text   : {}                  /* CODE                            */
   }
} > FRAM

.bss       : {} > RAM               /* GLOBAL & STATIC VARS            */
.stack     : {} > RAM (HIGH)        /* SOFTWARE SYSTEM STACK           */

.infoA     : {} > INFOA             /* MSP430 INFO FRAM  MEMORY SEGMENTS */
.infoB     : {} > INFOB

.int00   : {} > INT00               /* MSP430 INTERRUPT VECTORS        */
```

Setting Up Code and Data Memory

- Case 1: all global variables are assigned to FRAM
  - Advantage: All variables are non-volatile, no special handling required for backing up specific data
  - Disadvantage: Uses up code space, increased power, decreased throughput if MCLK > 8MHz
- Case 2: all global variables are assigned to SRAM
  - Advantage: Some variables may need to be volatile e.g. state machine, frequently used variables do not cause a throughput, power impact
  - Disadvantage: User has to explicitly define segments to place variables in FRAM
- Achieving an optimized user experience is a work in progress…

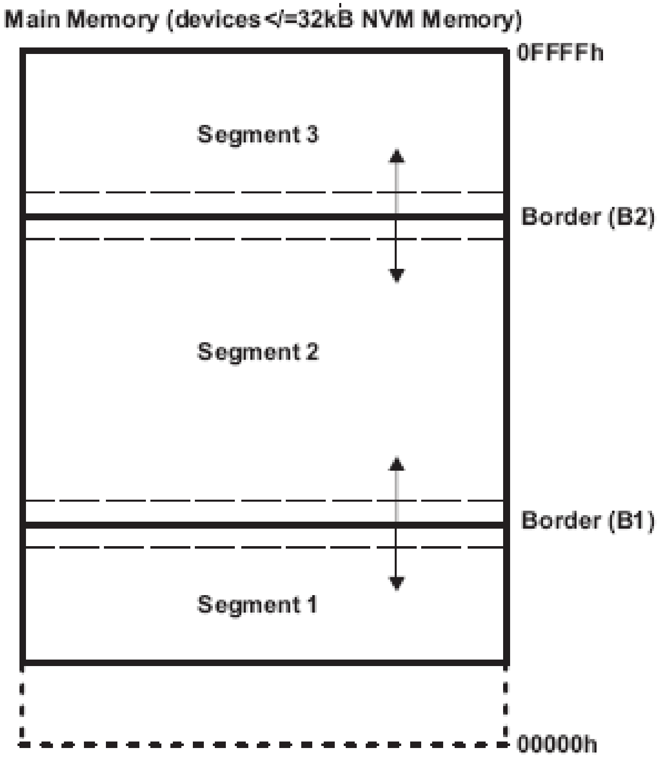
Memory Protection Unit (MPU)

- FRAM is so easy to write to…
- Both code and non-volatile data need protection
- MPU protects against accidental writes [read, write and execute only permissions]
- Features include:
  - Configuration of main memory in three variable sized segments
  - Independent access rights for each segmen
  - MPU registers are password protected



Calculating Segment Boundaries

- Size of segment determined by setting the MPUSB register (Segment Borders)
- Total # of bits = 5
- For 16K device
  – Segment Granularity =16*1024 / 32 = 512 bytes



Texas Instruments

Creating Segments in 4 Easy Steps

Step 1: Decide segment boundaries

Segment 1 = 0xC200 to 0xCDFF

Segment 2 = 0xCE00 to 0xD7FF

Segment 3 = 0xD800 to 0xFFFF

Step 2: Look up User's Guide Table for MPUSBx values

MPUSBx[4:0]	Page_start Address	
0x01	0xC200	
…..	0xCxxx	
0x07	0xCE00	→ B1
…	0xCxxx	
0x0C	0xD800	→ B2

Creating Segments in 4 Easy Steps

Step 3: Write to table values to MPUSEG register

0x0C

15	14	13	12	11	10	9	8
Reserved	Reserved	Reserved	MPUSB2[4]	MPUSB2[3]	MPUSB2[2]	MPUSB2[1]	MPUSB2[0]

7	6	5	4	3	2	1	0
Reserved	Reserved	Reserved	MPUSB1[4]	MPUSB1[3]	MPUSB1[2]	MPUSB1[1]	MPUSB1[0]

0x07

Step 4: Assign rights and violation responses for each segment

15	14	13	12	11	10	9	8
MPUSEGIVS	MPUSEGIXE	MPUSEGIWE	MPUSEGIRE	MPUSEG3VS	MPUSEG3XE	MPUSEG3WE	MPUSEG3RE

7	6	5	4	3	2	1	0
MPUSEG2VS	MPUSEG2XE	MPUSEG2WE	MPUSEG2RE	MPUSEG1VS	]MPUSEG1XE	MPUSEG1XE	MPUSEG1RE

Lab 2

Goals:

1) Study MPU registers

2) Assign Segment Boundaries using the User’s Guide Table

3)Assign segment rights and violation response as indicated

Obtaining Lab Software

- Software for this lab can be obtained from the MSP430 FRAM Training Wiki
- The link is: http://processors.wiki.ti.com/index.php/MSP430_FR57xx_3_HR_Lab
- Download the zip file
  - The folder 'FR-EXP User Experience' contains the FRAM EXP User Experience Code and the Graphical User Interface used in Lab1
  - The folder 'LabWorkspace' contains the CCS workspace location and the source files for executing the labs 2 & 3
- By default the board is programmed with the User Experience code
- If this needs to be re-installed after the labs use the batch file in the folder 'User Experience Programmer'

Configuring the MPU

LAB2

1. Set Lab 2 as active project
2. Fill in the blank spaces in lab2.c to match the following criteria
   - Enable access to MPU register
   - Setup segment boundaries at **0xC800 and 0xD000**
   - Disable write access for the Segment 2
   - Enable reset on violation for Segment 2
3. Once complete, build project Lab2
4. Download and run the code example
5. LED5 should toggle on correct execution

Source Code Snapshot

```
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                   // Stop WDT

  P3OUT &= ~BIT4;
  P3DIR |= BIT4;                              // Configure P3.4 for LED
  P3OUT |= BIT4;
  __delay_cycles(100000);
  P3OUT &= ~BIT4;
  __delay_cycles(100000);

  // Configure MPU
  MPUCTL0 =                                   // Write PWD to access MPU registers
  MPUSEG =                                    // B1 = 0xC800; B2 = 0xD000
                                              // Borders are assigned to segments
  MPUSAM &=                                   // Segment 2 is protected from write
  MPUSAM |=                                   // Violation select on write access

  MPUCTL0 = MPUPW+MPUENA+MPUSEGIE+MPULOCK;  // Enable NMI & MPU protection

  Data = 0x88;
  // Cause an MPU violation by writing to segment 2
  ptr = (unsigned int  *)
  *ptr = Data;

  while(1);
}
```

1. Write password
2. Write segment boundaries
3. Protect segment 2 from write access
4. Configure reset on violation
5. Create a violation

Configuring the MPU

LAB2 Observations:

- MPU is essential to protect code vs. data memory
- MPU can be programmed to reset the device in case of an access violation

Checklist:

✓ Learn about the MPU registers

✓ Configure Segment Boundaries

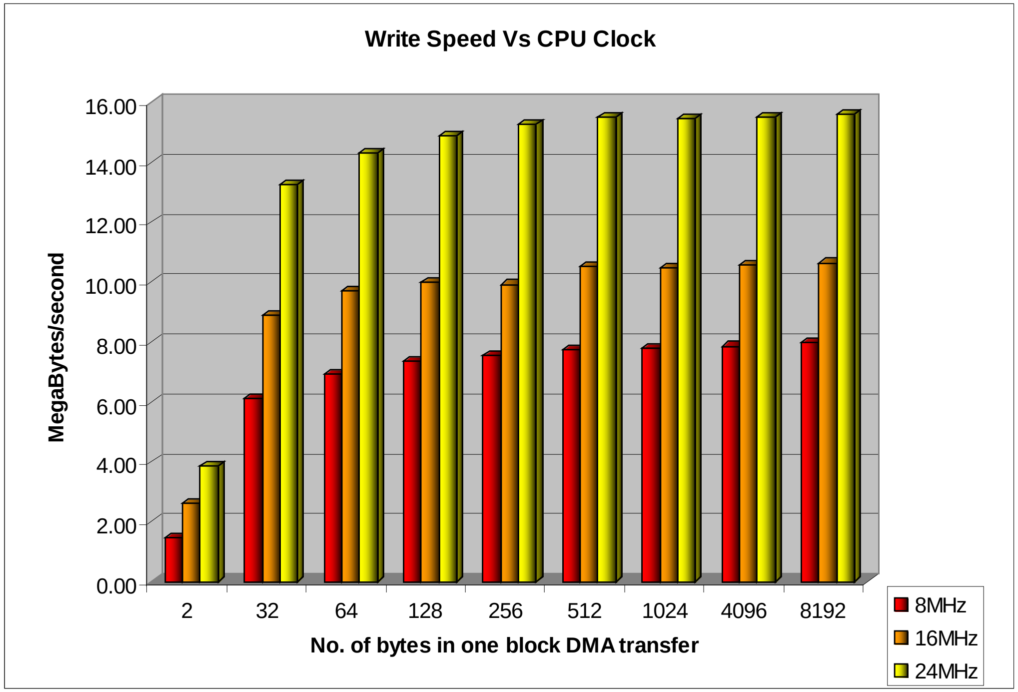
✓ Create individual access rights for each segment

✓ Assign violation response per segment

Maximizing FRAM Write Speed

- FRAM Write Speeds are mainly limited by communication protocol or data handling overhead etc
- For in-system writes FRAM can be written to as fast as **16MBps!**
- The write speed is directly dependent on:
  - The use of DMA
  - System speed and
  - Block size

*Refer to Application Report titled **Maximizing FRAM Write Speed on the MSP430FR573x***

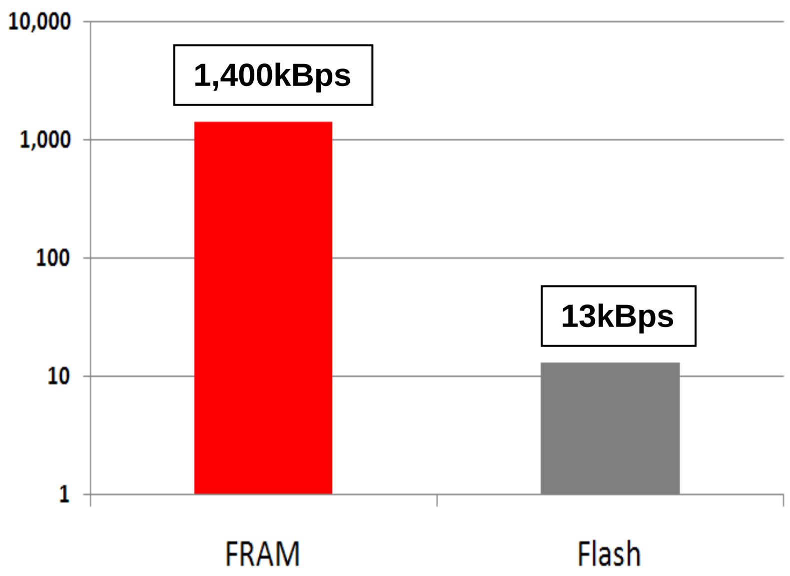


Differentiating with the FR57xx

FRAM = Ultra-fast Writes

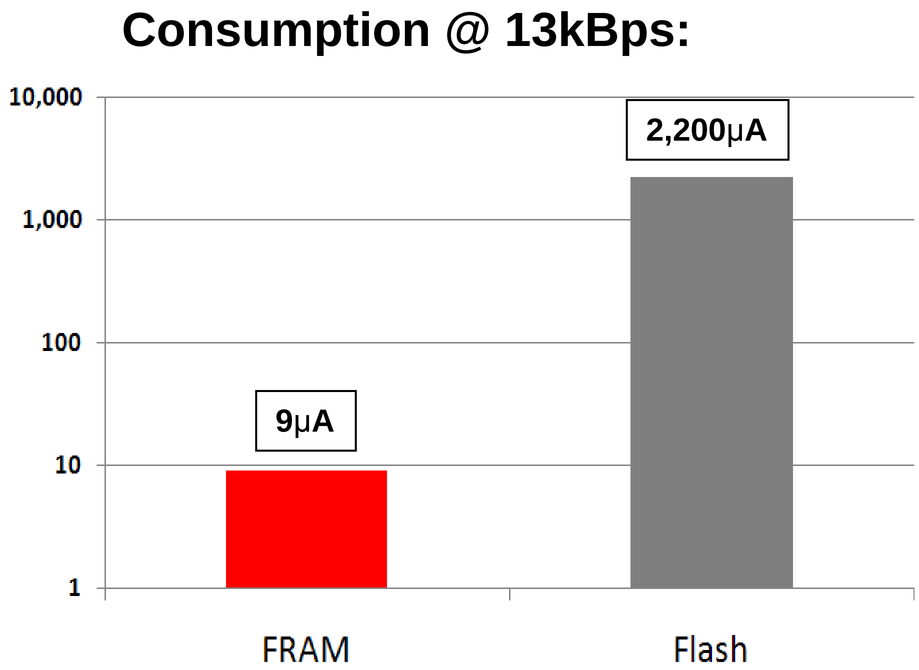
- Use Case Example: MSP430F2274 Vs MSP430FR5739
- Both devices use System clock = 8MHz
- Maximum Speed FRAM = 1.4Mbps [100x faster]
- Maximum Speed Flash = 13kBps

Max. Throughput:



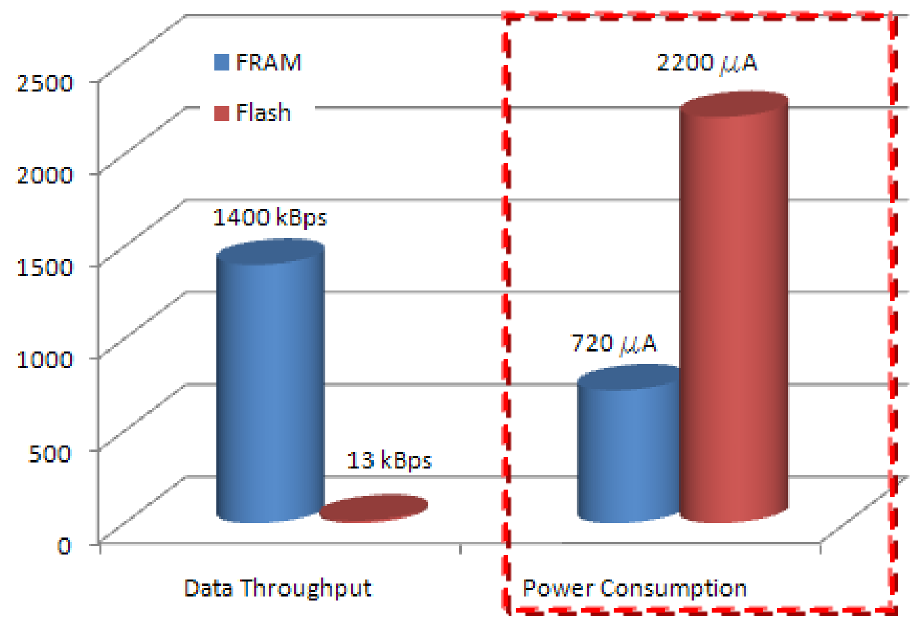
FRAM = Low active write duty cycle

- Use Case Example: MSP430F2274 Vs MSP430FR5739
- Both devices write to NV memory @ 13kBps
- FRAM remains in standby for 99% of the time
- Power savings: >200x of flash



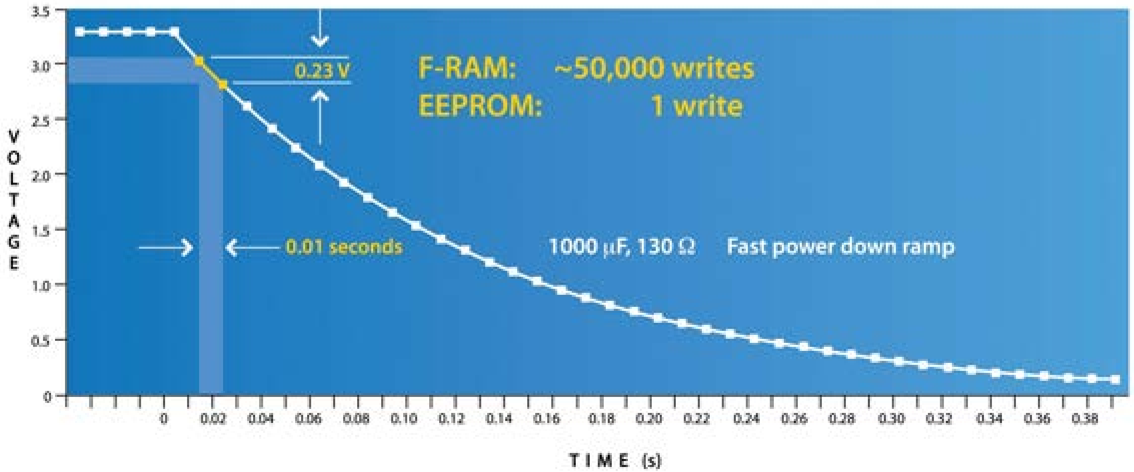
FRAM = Ultra-low Power

- Use Case Example: MSP430F2274 Vs MSP430FR5739
- Average power FRAM = 720µA @ 1.5Mbps
- Average power Flash = 2200µA @ 12kBps
- 100 times faster in half the power
- Enables more unique energy sources
- FRAM = Non-blocking writes
  - CPU is not held
  - Interrupts allowed



FRAM = Increased flexibility

- Use Case Example: EEPROM Vs MSP430FR5739
- Many systems require a backup procedure on power fail
- FRAM IP has built-in circuitry to complete the current 4 word write
  - *Supported by internal FRAM LDO & cap*
- In-system backup is an order of magnitude faster with FRAM



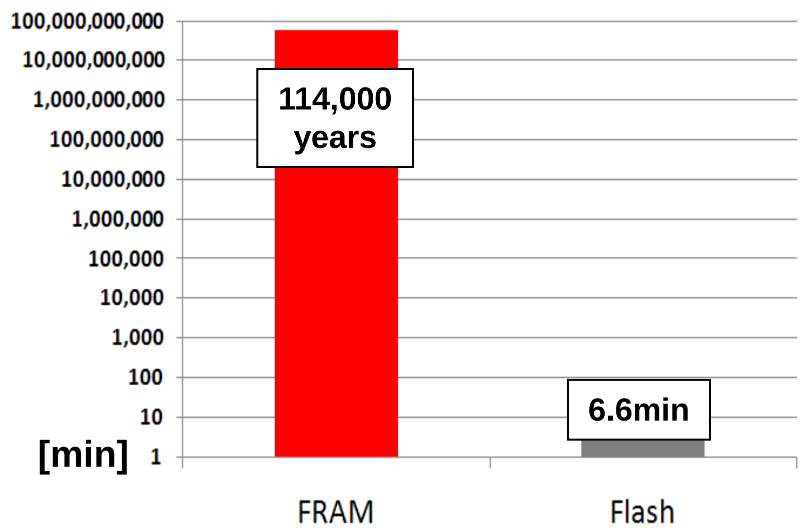
Write comparison during power fail events+

+ *Source: EE Times Europe, An Engineer's Guide to FRAM by Duncan Bennett*

FRAM = High Endurance

- Use Case Example: MSP430F2274 Vs MSP430FR5739
- FRAM Endurance >= 100 Trillion [10^14]
- Flash Endurance < 100,000 [10^5]
- Comparison: write to a 512 byte memory block @ a speed of 12kBps
  - Flash = 6 minutes
  - FRAM = 100+ years!

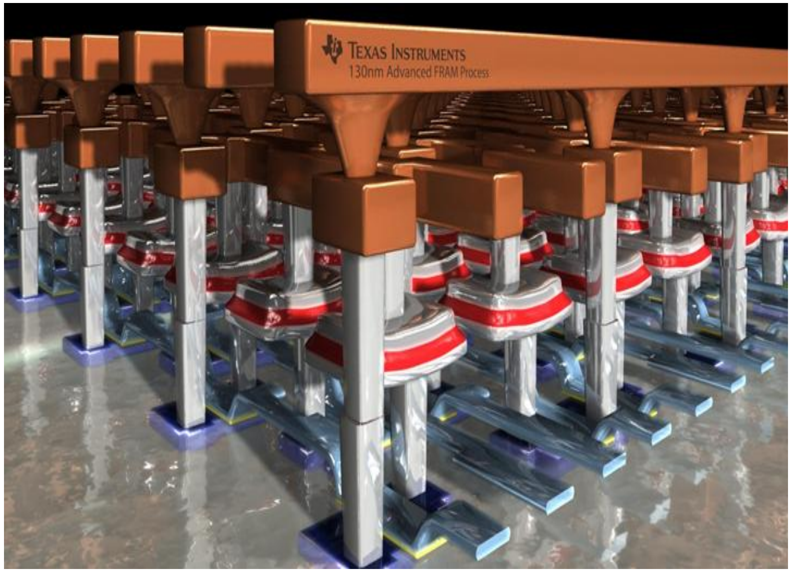
We need a log scale to compare!



FRAM and Reliability

FRAM: Proven, Reliable

- Endurance
  - Proven data retention to 10 years @ 85°C
- Radiation Resistance
  - Terrestrial Soft Error Rate (SER) is below detection limits
- Immune to Magnetic Fields
  - FRAM does not contain iron!

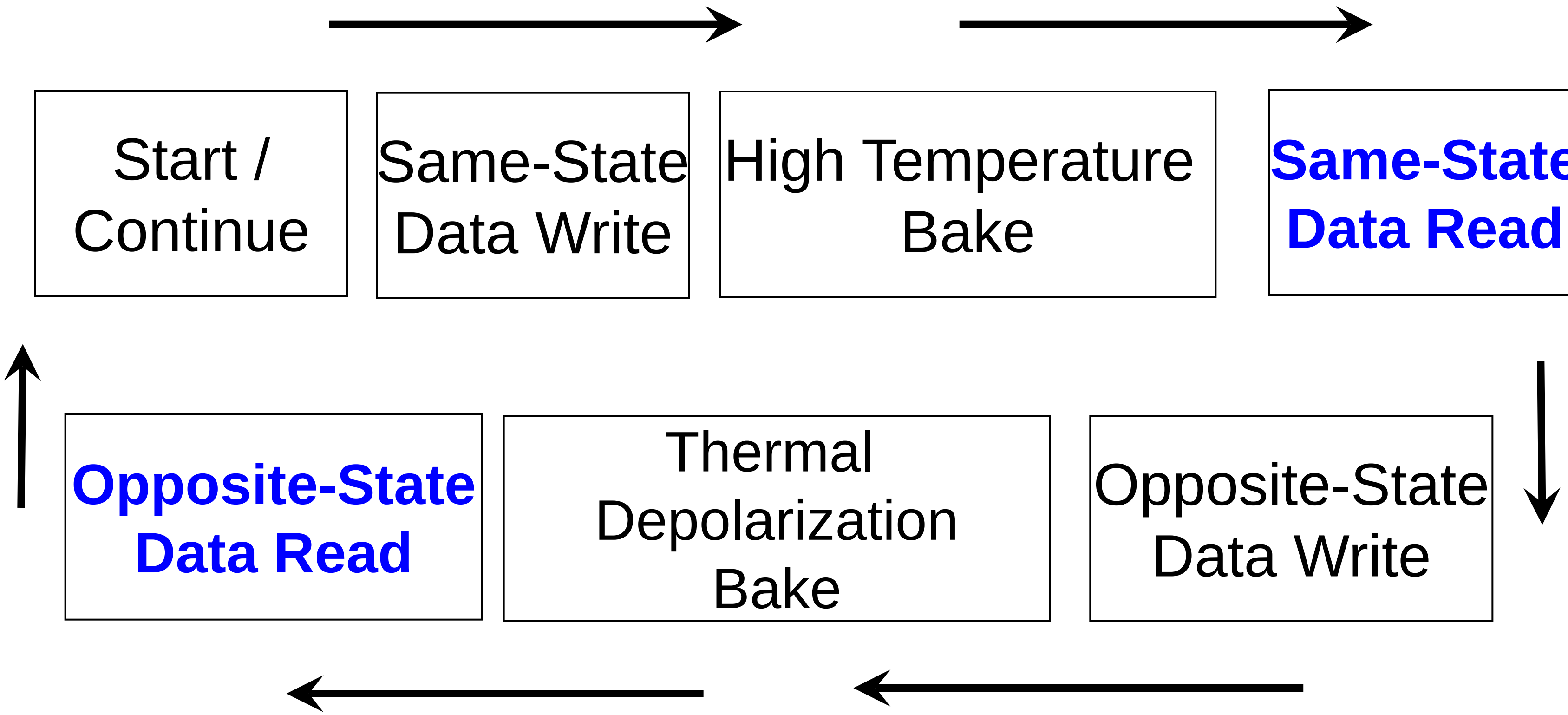


www.ti.com/fram
For more info on
TI's FRAM technology

Data Retention Definitions

- FRAM cells are tested for the following:
  - **Thermal Depolarization** is a reduction of the spontaneous polarization as the sample temperature increases
  - **Imprint** is the stabilization of polarization in a preferred state
    - "Same-State" (SS) retention may strengthen
    - "Opposite-State" (OS) retention may weaken

Data Retention Test Procedure



- **High Temp bake accelerates imprint related signal reduction**
- **Thermal Depolarization: 15-30 minutes at operating Temp**

Summary of Retention and Imprint Tests

- Mechanism is temperature dependent with activation energy ~1.4eV
- 100 hours bake @150°C ~ 1,000 hours @125°C ~ **10 years @85°C**
- Test sequence designed to demonstrate data retention with no fails through 10 years at 85°C
- Bits are long term baked in Same-State to maximize amount of imprint
  - **125°C Same-State bake to 1,000 cumulative hours**
  - **85°C Opposite-State bake at each read point to verify retention at operating condition**

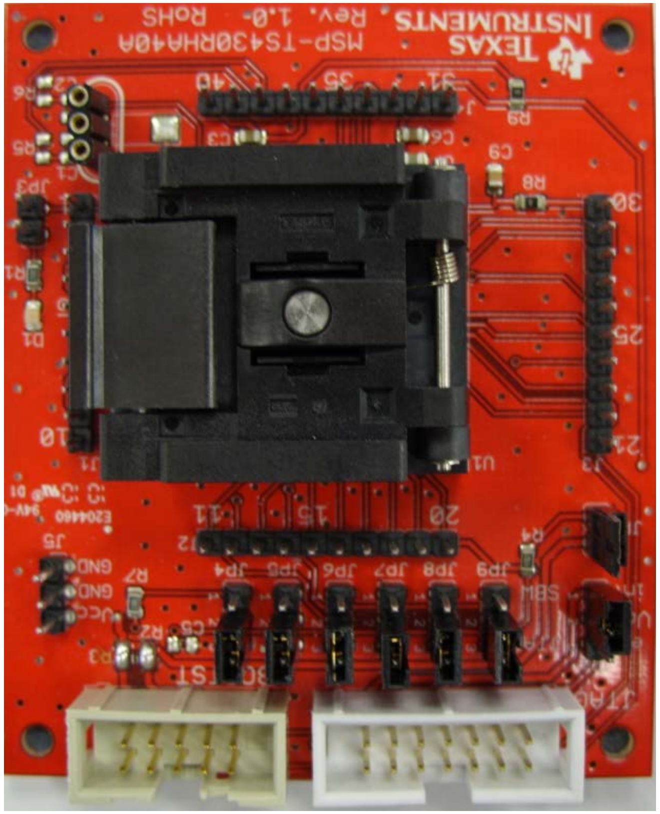
What about Reflow?

- **TI factory programming is not available for the MSP430FR57xx devices**
- **Customer and CMs MUST program post reflow or other soldering is activity**
- **Hand soldering is not recommended. However it can be achieved by following the guidelines**
  - ✓Be mindful of temperature: FRAM can be effected above 260 deg C for long periods of time
  - ✓Using a socket to connect to evaluation board during prototyping is also a best practice

Tools & Resources

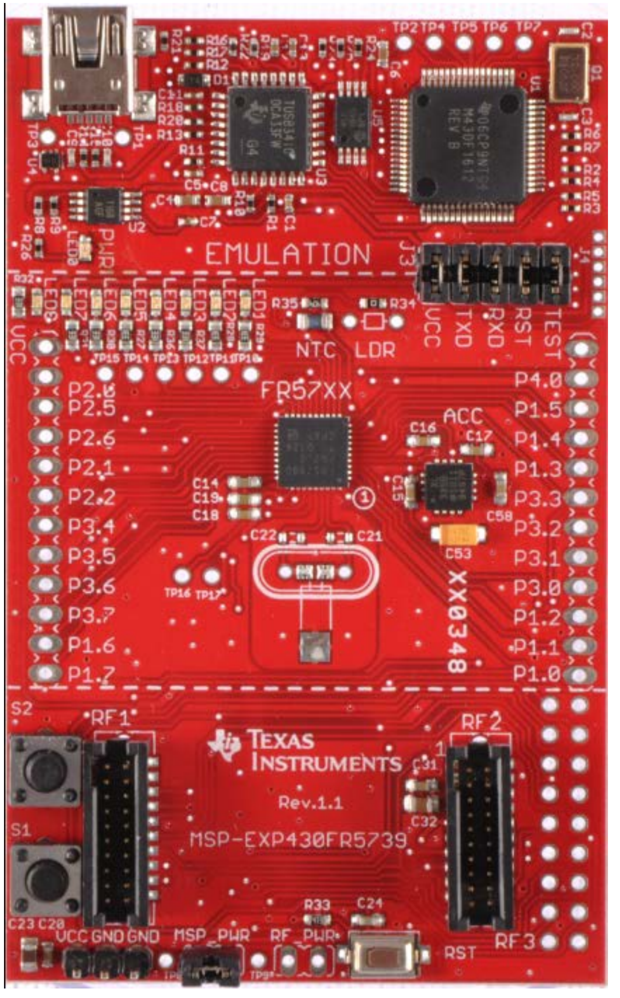
Getting Started with MSP430FR5739

- **MSP430FR5739 Target Board**
- Development board with 40-pin RHA socket (MSP-TS430RHA40A)
- All pins brought out to pin headers for easy access
- Programming via JTAG, Spy-bi-wire or BSL
- $99



Getting Started with MSP430FR5739

- **MSP-EXP430FR5739 FRAM Experimenter's Board**
- $29
- On Board Emulation
- Features
  - 3 axis accelerometer
  - NTC Thermister
  - 8 Display LED's
  - Footprint for additional through-hole LDR sensor
  - 2 User input Switches
- User Experience
  - Preloaded with out-of-box demo code
  - 4 Modes to test FRAM features:
    - Mode 1 - Max FRAM write speed
    - Mode 2 - Flash write speed emulation
    - Mode 3 – FRAM writes using sampled accelerometer data
    - Mode 4 – FRAM writes using sampled Thermistor data



Getting Started with MSP430FR5739

- www.ti.com/fram
- Product Page from www.msp430.com
- Upcoming Collateral:
  - Maximizing FRAM Write Speed
  - FR57xx Migration Guide
  - FR-EXP Tool User's Guide
  - FRAM Reliability Application Report
  - Code Examples
  - Embedded Developers Guide to FRAM
  - FRAM for Dummies by V.C. Kumar

To Summarize

FRAM is real! The world's first ultra-low power catalog FRAM microcontroller is here.

- Top 3 FRAM sellers are:
  - Ultra-fast writes
  - Ultra-low power
  - Super high endurance
- FR5739: Great general purpose MCU enhanced by FRAM
- FR5739: Targets niche applications where *only* FRAM makes sense
- Check out our Demos!