Dallas, TX - November 2002

Last Minute Notes

Agenda - Dallas, November 2002

Optional Introductory Session

MSP430 Basics:

- Ultra-low power, high-performance analog and RISC CPU
- Roadmap
- Product family comparison

Embedded Applications:

- SAR and slope analog-to-digital converter comparison
- MCU best practice coding techniques
- Tools and resources
- “Flashing the LED” tool refresher

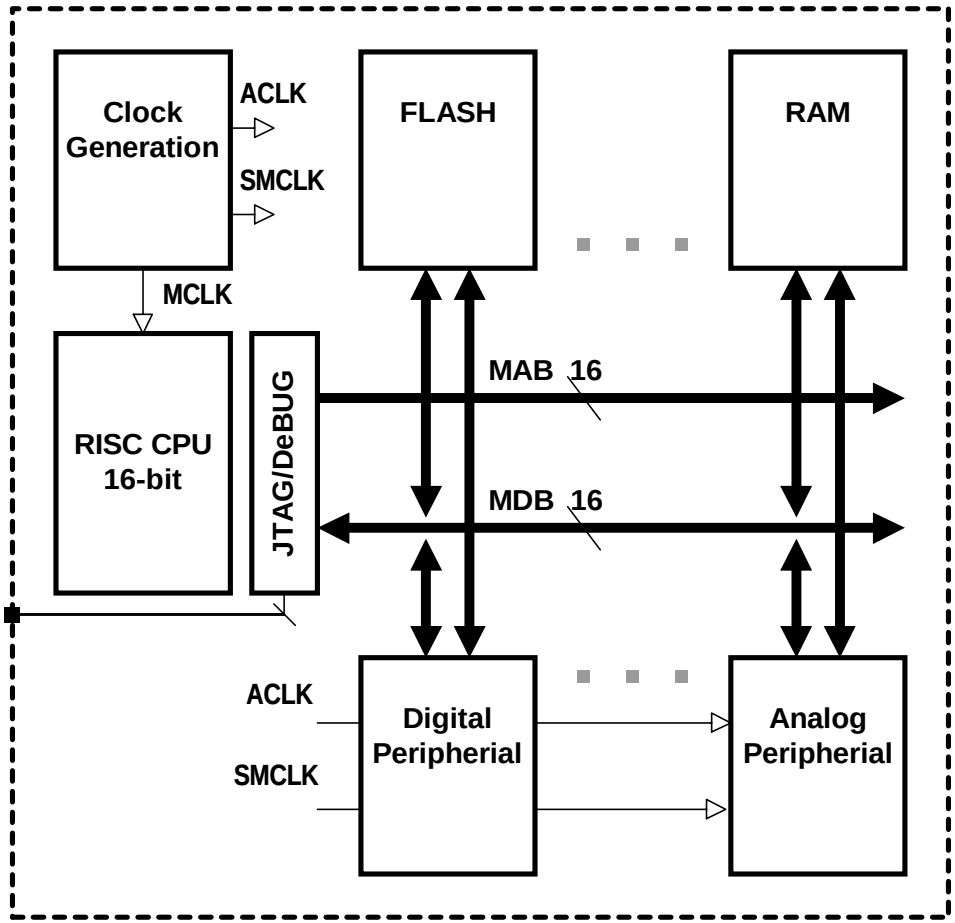
MSP430 Basics

- ***Ultra-low power*** **architecture**
  - 0.1uA RAM retention
  - 0.8uA real-time clock mode
  - 250uA / 1MIPS active
  - 1.8 - 3.6V operation
  - Multiple oscillator clocking
  - Vectored interrupt capability
  - -40 - +85C industrial temperature
- ***High-performance analog*** **for precision measurement**
- ***Modern RISC CPU*** **enables new applications**
- **In-system programmable Flash**
- **Complete $99 integrated development environment**

MSP430 Modular Architecture

von-Neumann common bus connects CPU to all memory and peripherals

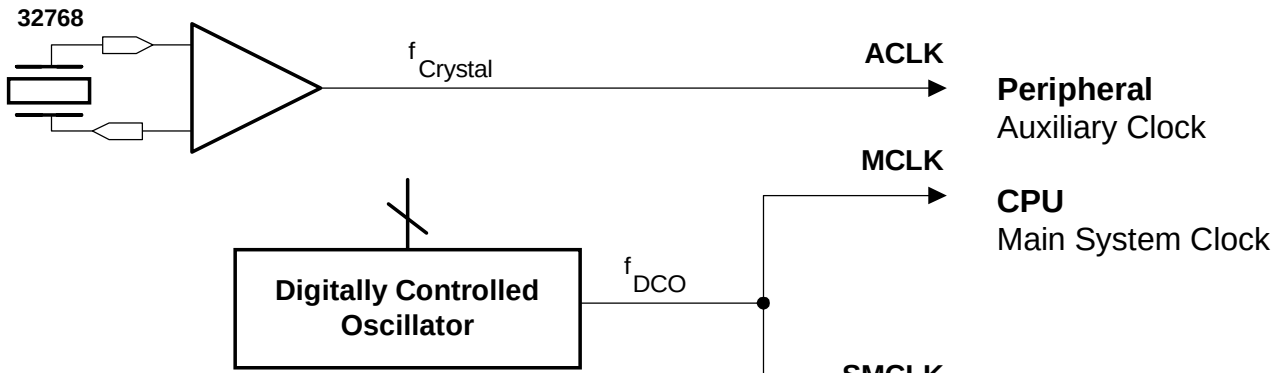
Embedded emulation accessed in-application with JTAG



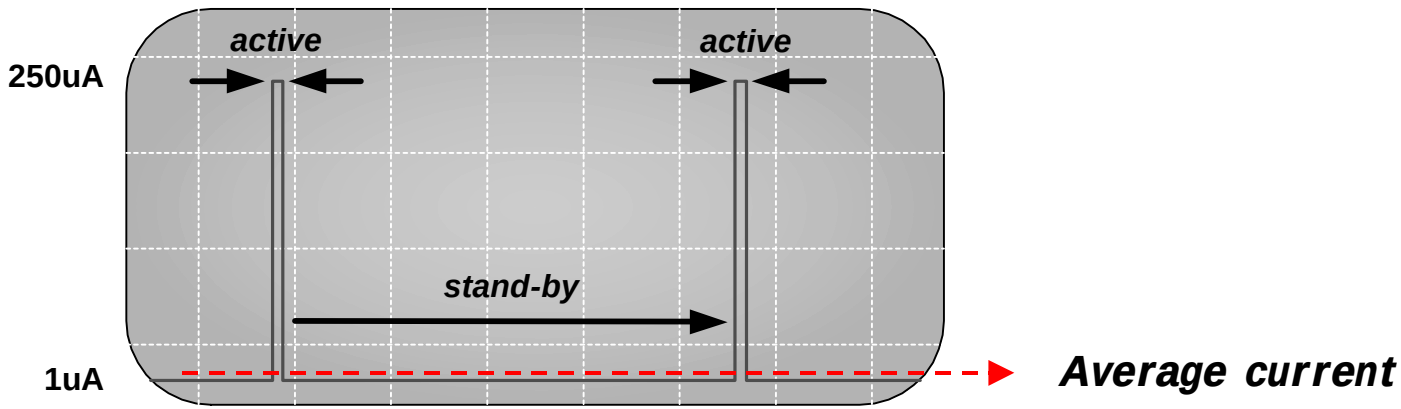
Architecture reduces power consuming, noise generating fetches to memory

16-bit bus handles wide-width data much more effectively

MSP430x1xx Basic Clock System Concept



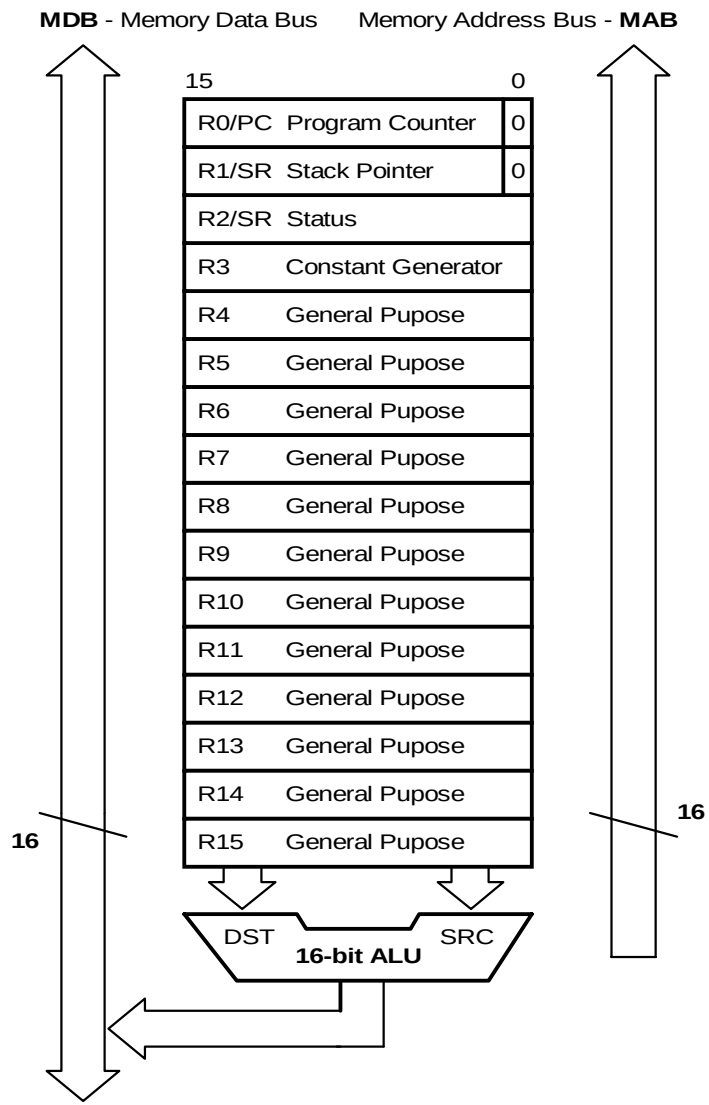
Activity Profile



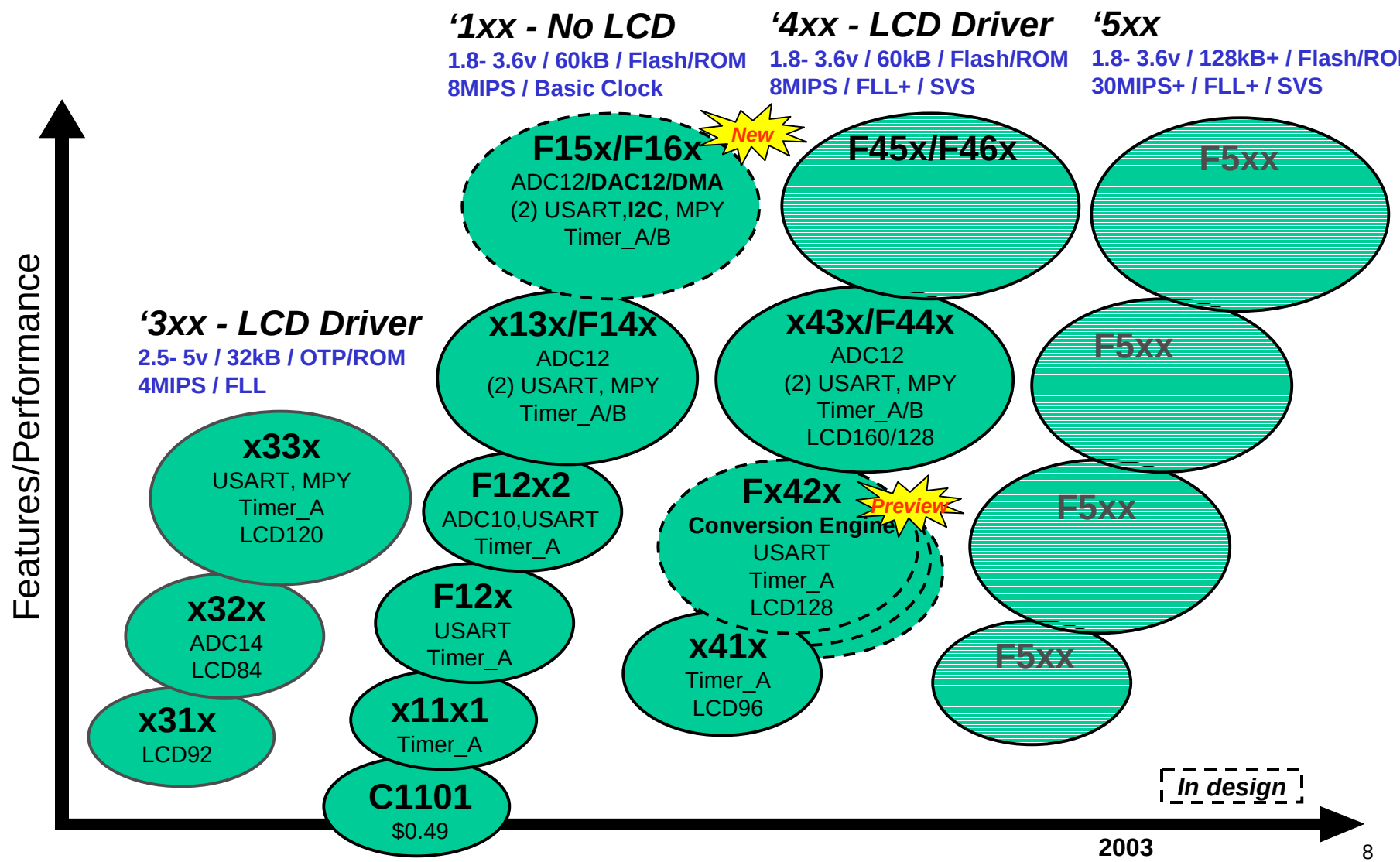
<6 µs Digitally Controlled Oscillator (DCO) start-up allows systems to remain in a low-power mode as long as possible extending battery life

MSP430 Orthogonal 16-bit RISC CPU

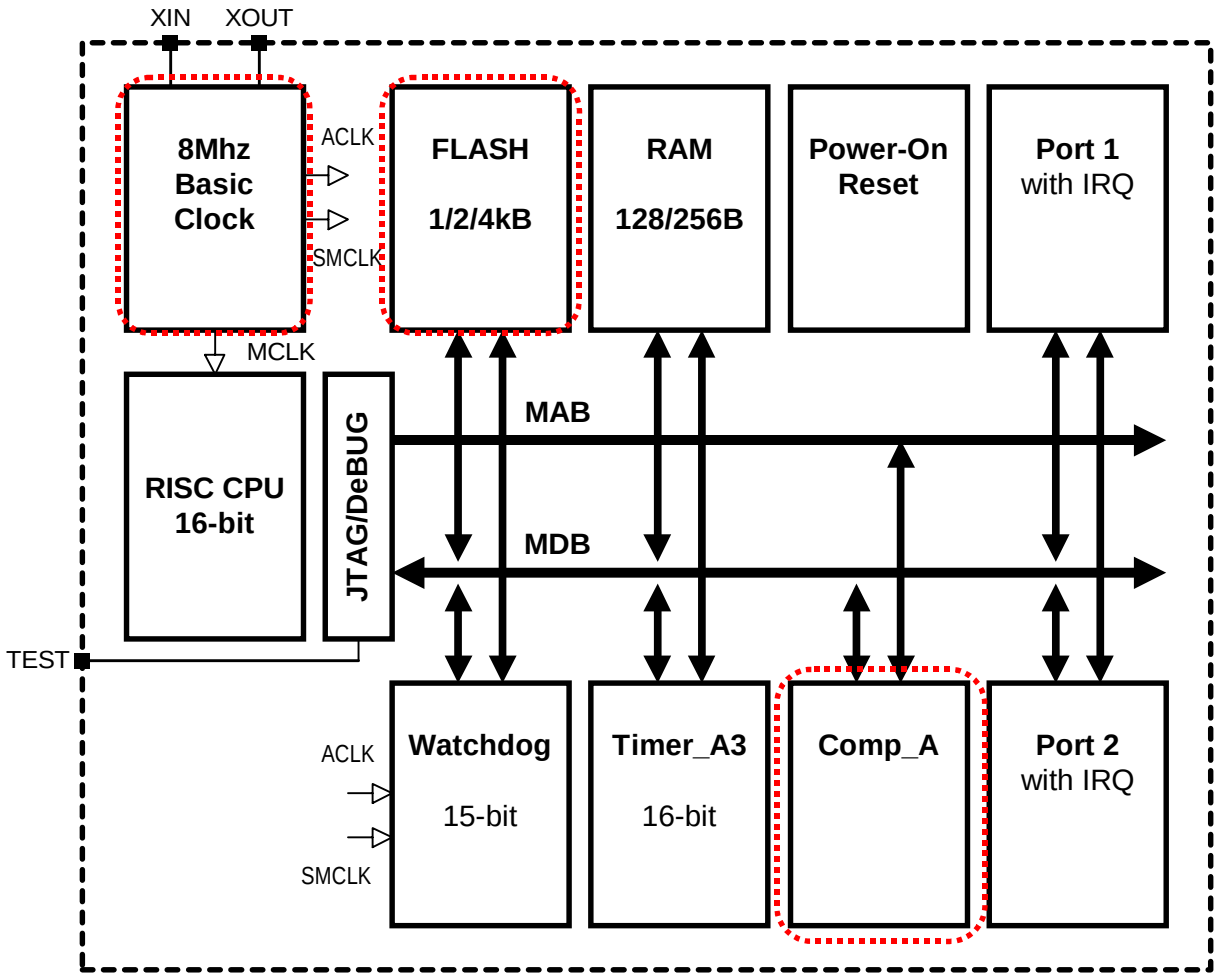
- Large 16-bit register file eliminates single accumulator bottleneck
- High-bandwidth 16-bit data and address bus with no paging
- RISC architecture with 27 instructions and 7 addressing modes
- Single-cycle register operations with full-access
- Direct memory-memory transfer designed for modern programming
- Compact silicon 30% smaller than an ‘8051 saves power and cost



MSP430 Roadmap



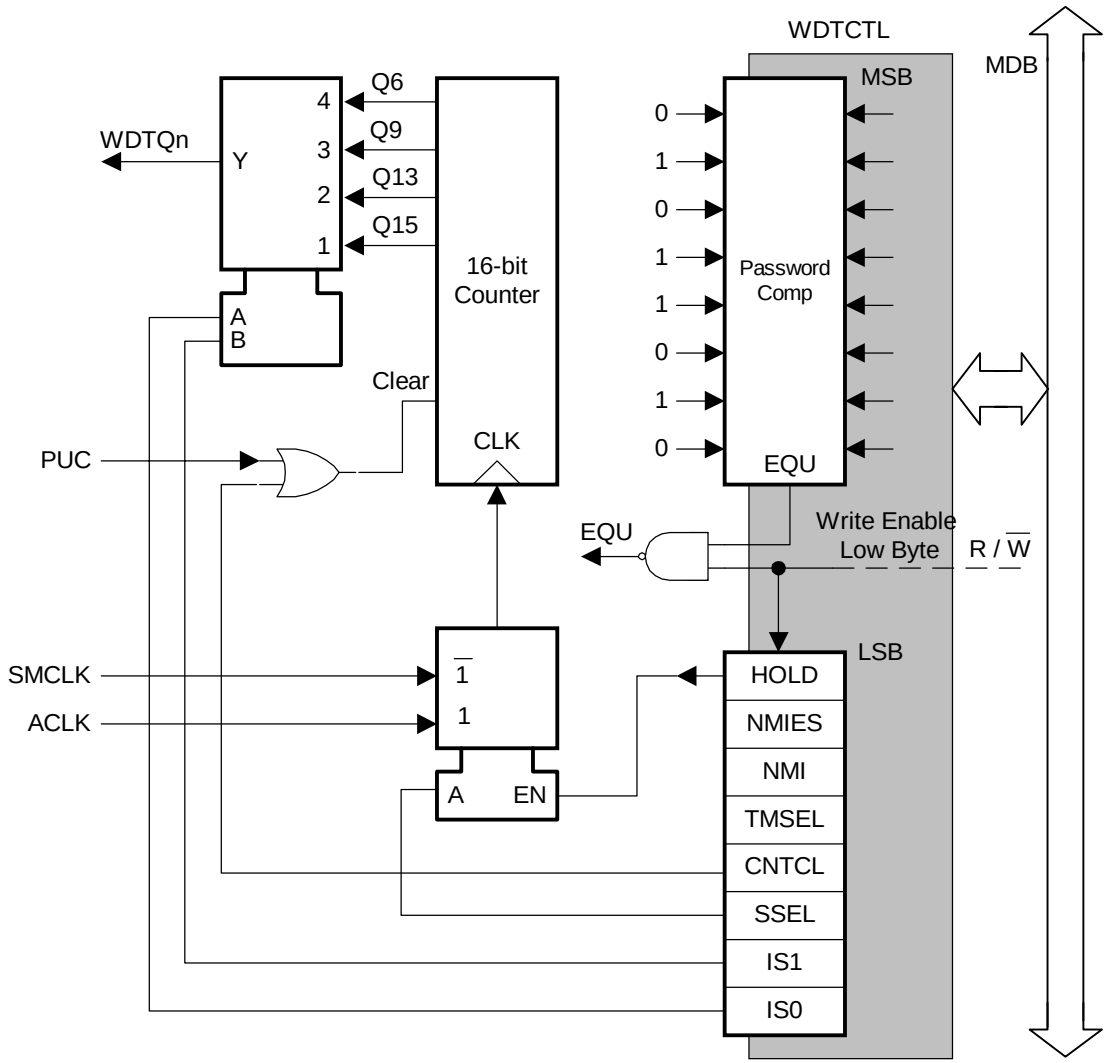
MSP430x11x(1)



20 TSSOP, SOIC

Watchdog Timer

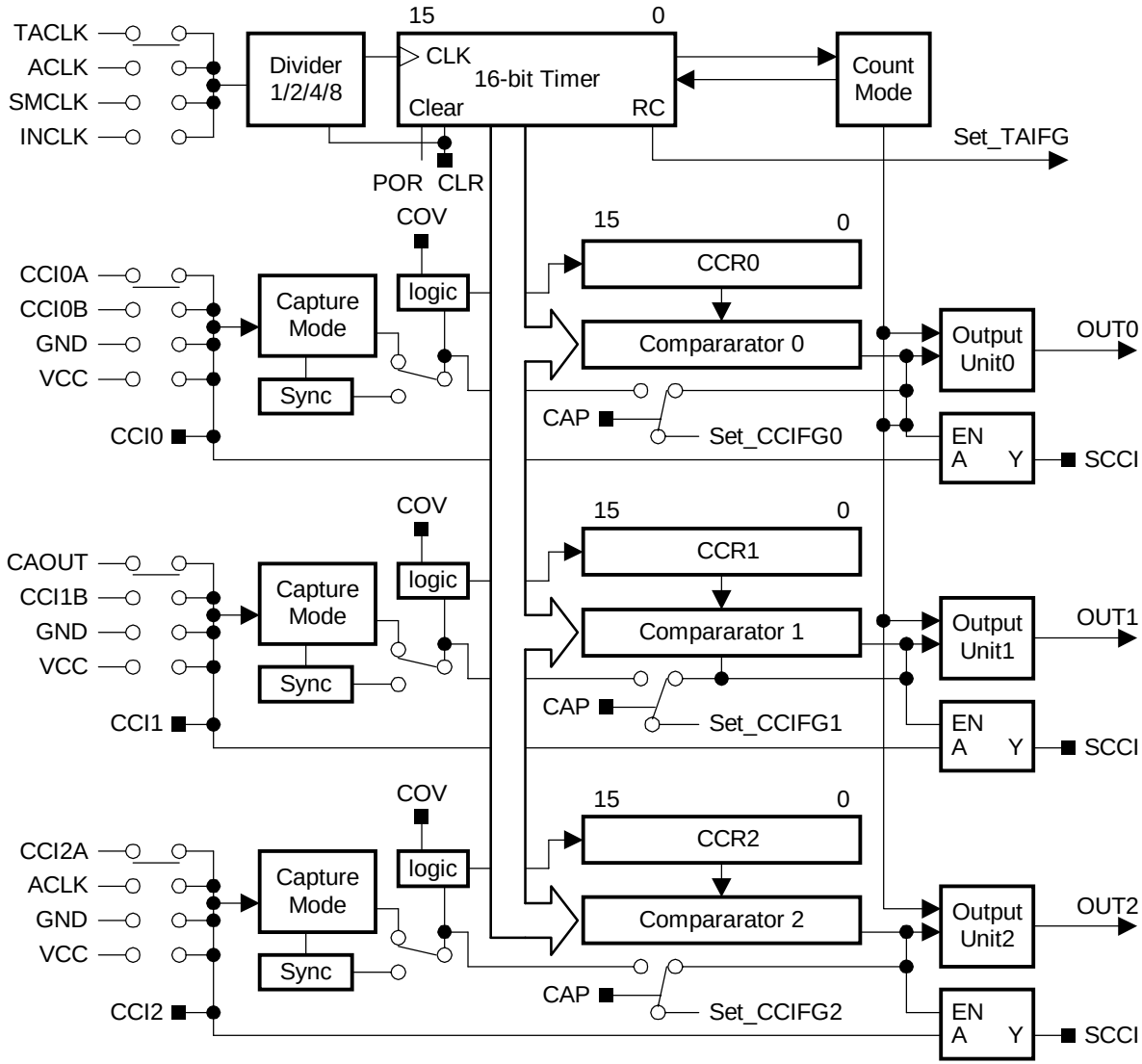
- Software watchdog or interval timer
- Eight software selectable intervals
- Access to WDTCTL must include 05A00h password
- One interrupt vector with enable and flags in SFR's



Caution: Powers up active as watchdog ~32ms reset

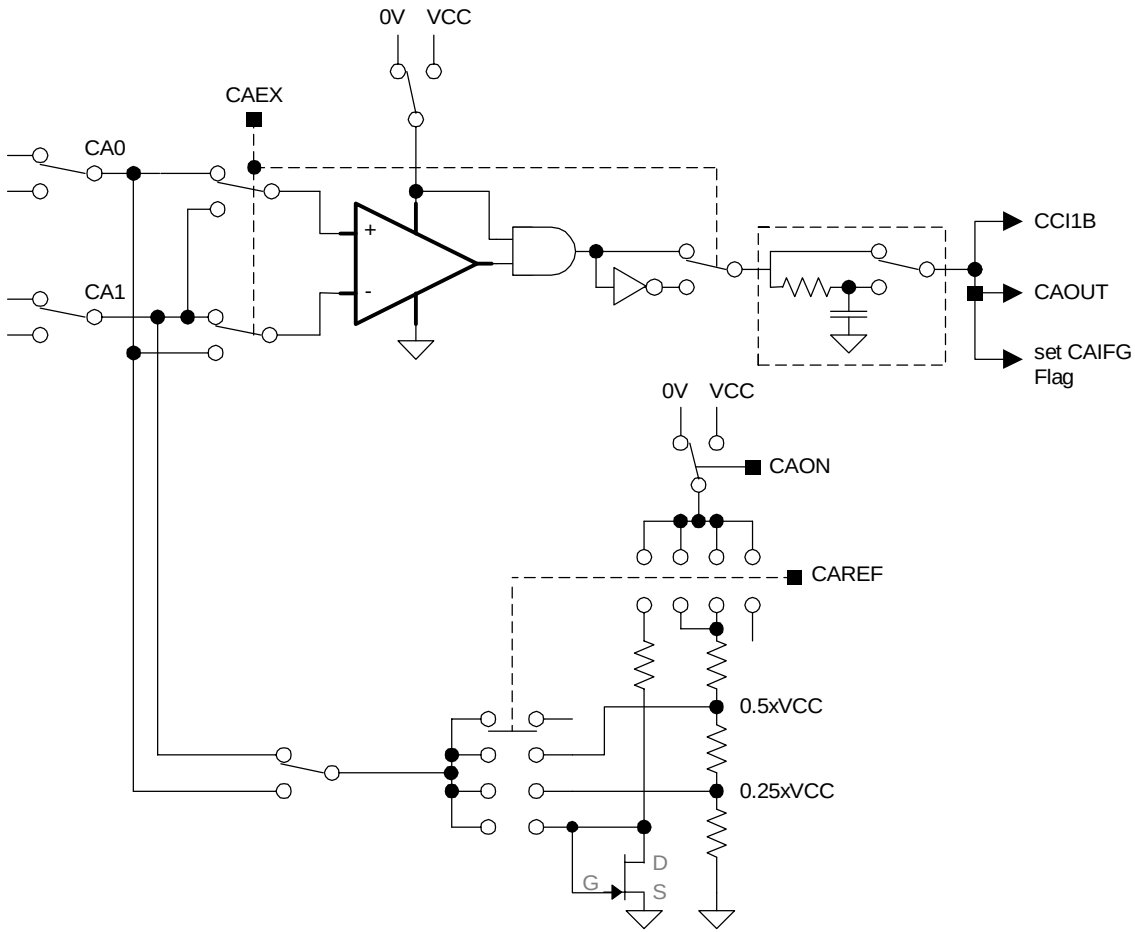
Timer_A3 Introduction

- 16-bit timer or counter
- Interval timer
- Capture external events
- Compare PWM mode
- SCCI latch for asynchronous communication
- Two interrupt vectors with enable and flags in module registers



Comparator_A

- References usable internally and externally
- Low-pass filter selectable by software
- Input terminal multiplexer
- One interrupt vector with enable and flags in module register

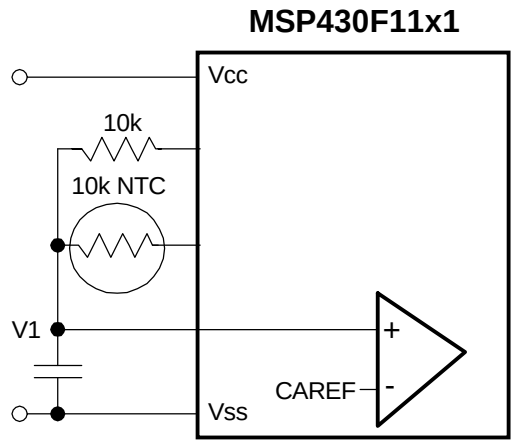


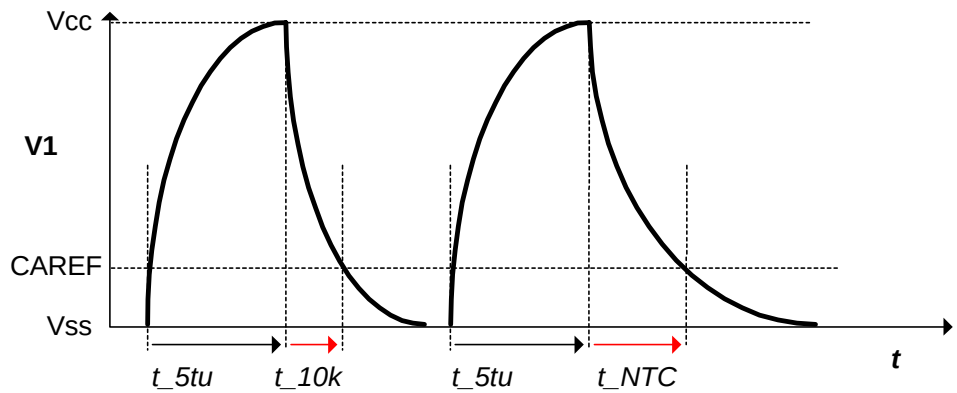
What is Slope ADC.. Using Comparator_A Ω Example

$$t\_10k = 10k \times C \times \ln\frac{V_{CAREF}}{Vcc}$$

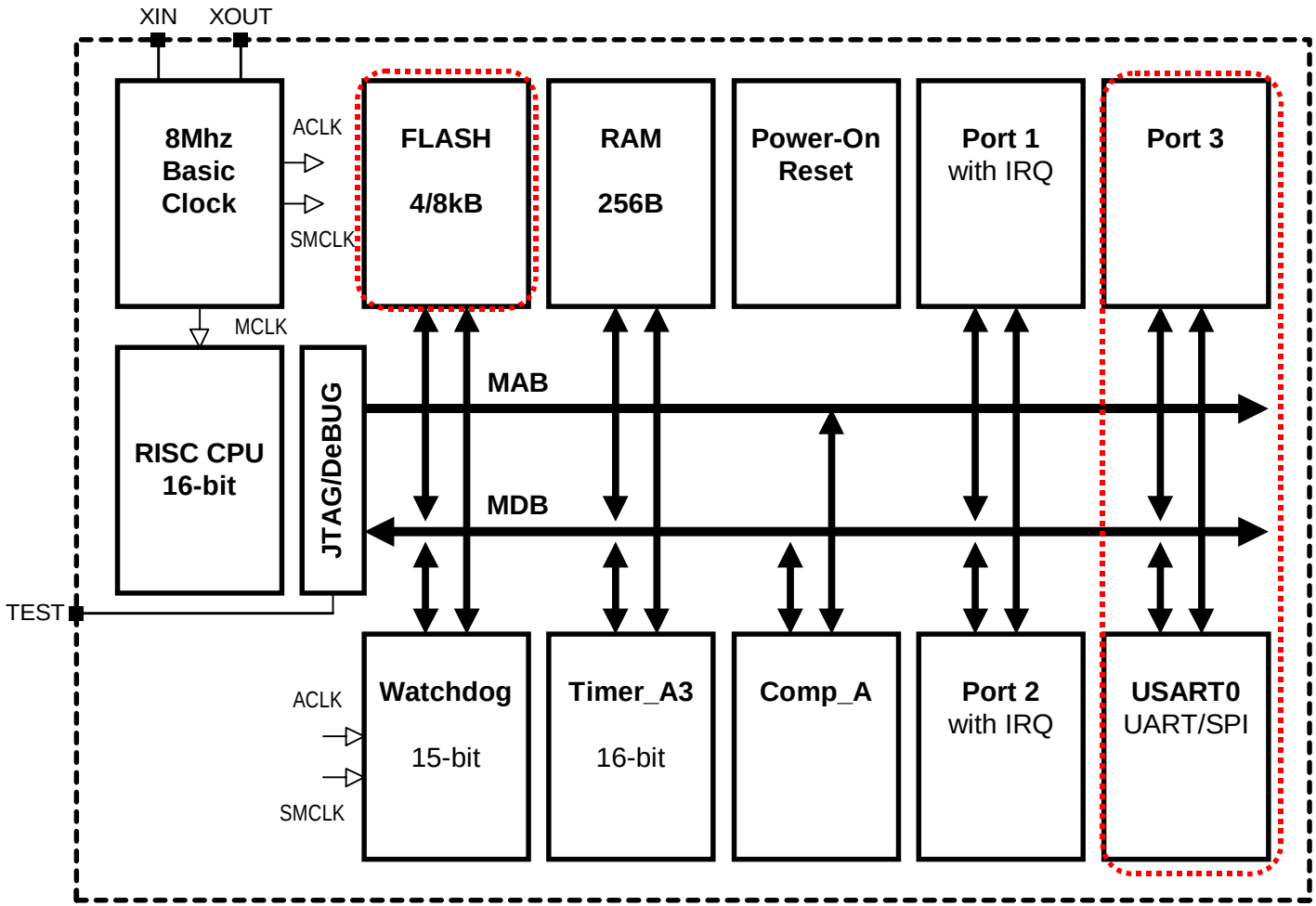
$$\frac{R\_NTC}{10k} = -\frac{t\_10k}{C \times \ln\frac{V_{CAREF}}{Vcc}} - \frac{C \times \ln\frac{V_{CAREF}}{Vcc}}{t\_10k}$$

$$R\_NTC = 10k \times \frac{t\_NTC}{t\_10k}$$





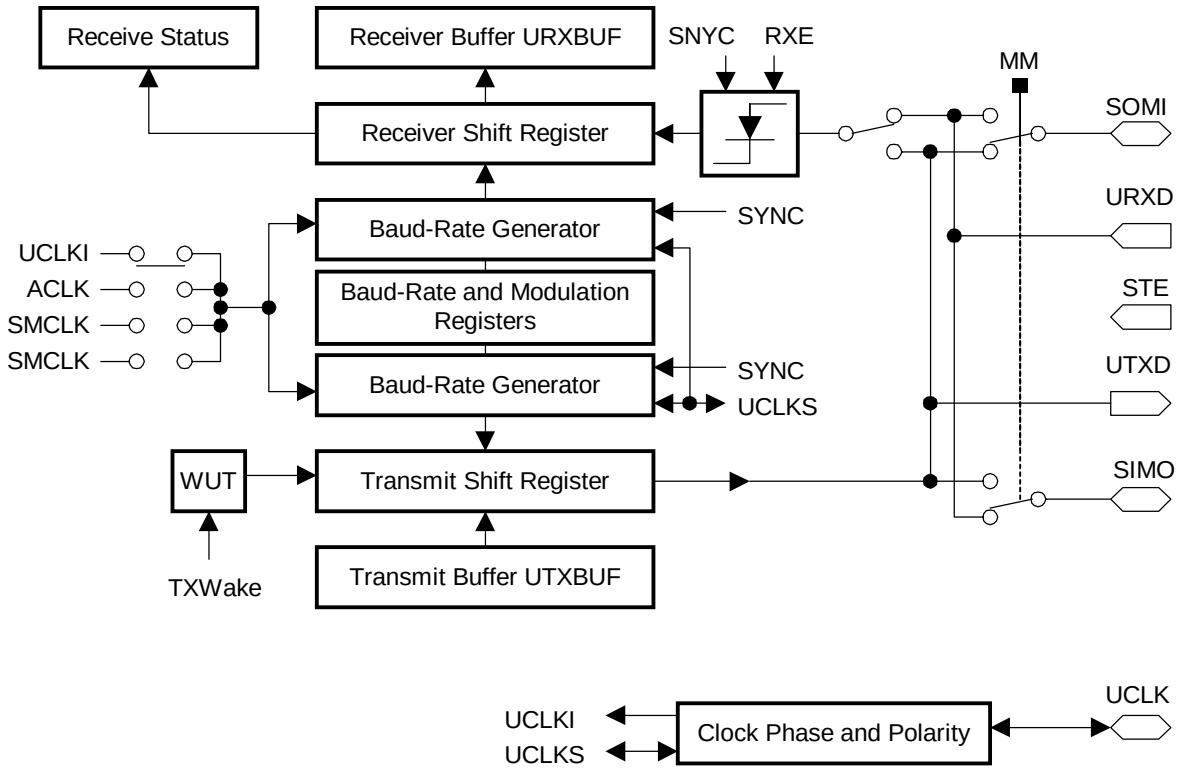
MSP430F12x



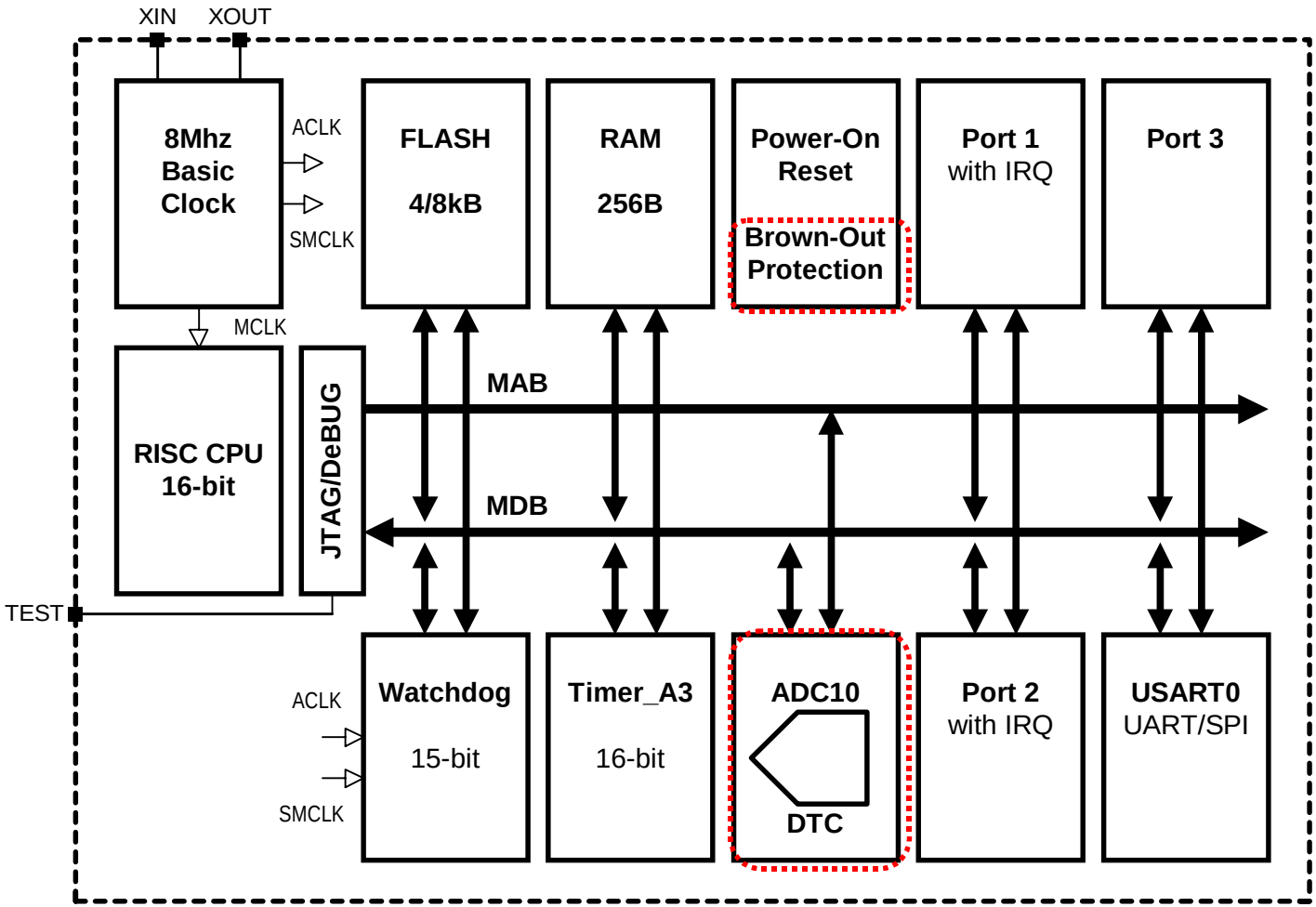
28 SOIC/TSSOP

USART Serial Port Introduction

- Software selectable UART or SPI
- Auto-start from any LPMx
- Double buffered RX and TX shift registers
- Baud-rate generator
- 7 or 8-bit data
- 9-bit addressing mode available
- Error detection and suppression
- Two interrupt vectors with enable and flags in SFR and module register



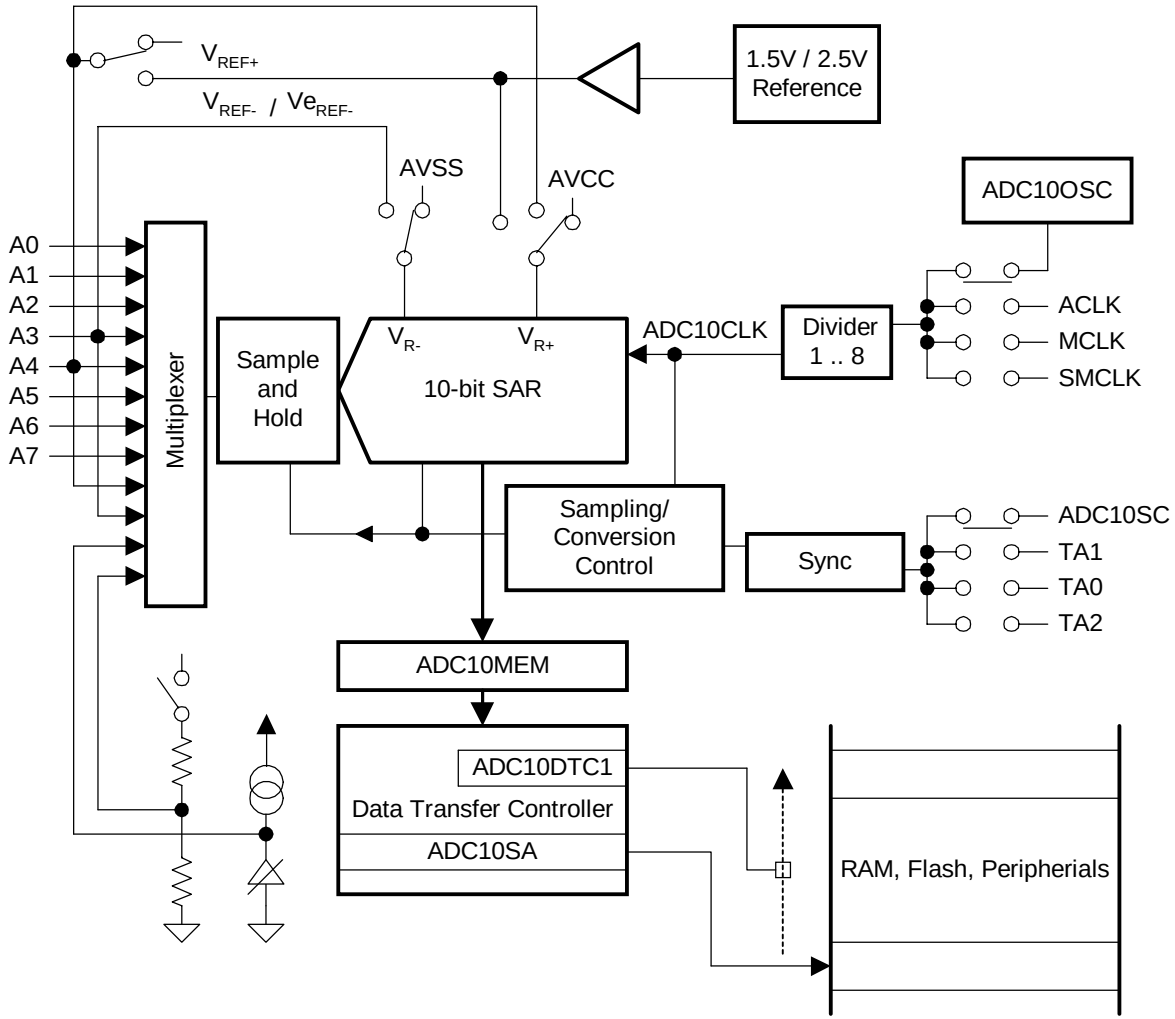
MSP430F12x2



28 SOIC/TSSOP

ADC10 Introduction

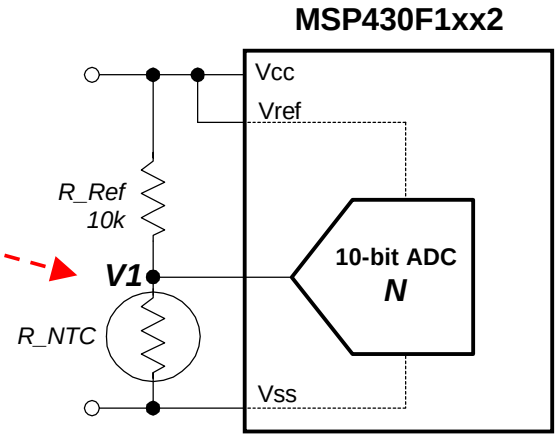
- 200ksps+
- Programmable sample & hold
- Band-gap Vref
- Temp. sensor
- Low-battery detect
- Signed/unsigned justification
- Autoscan
- Data Transfer Controller
- One interrupt vector with enable and flags in module register



Ratiometric Measurement Using ADC10

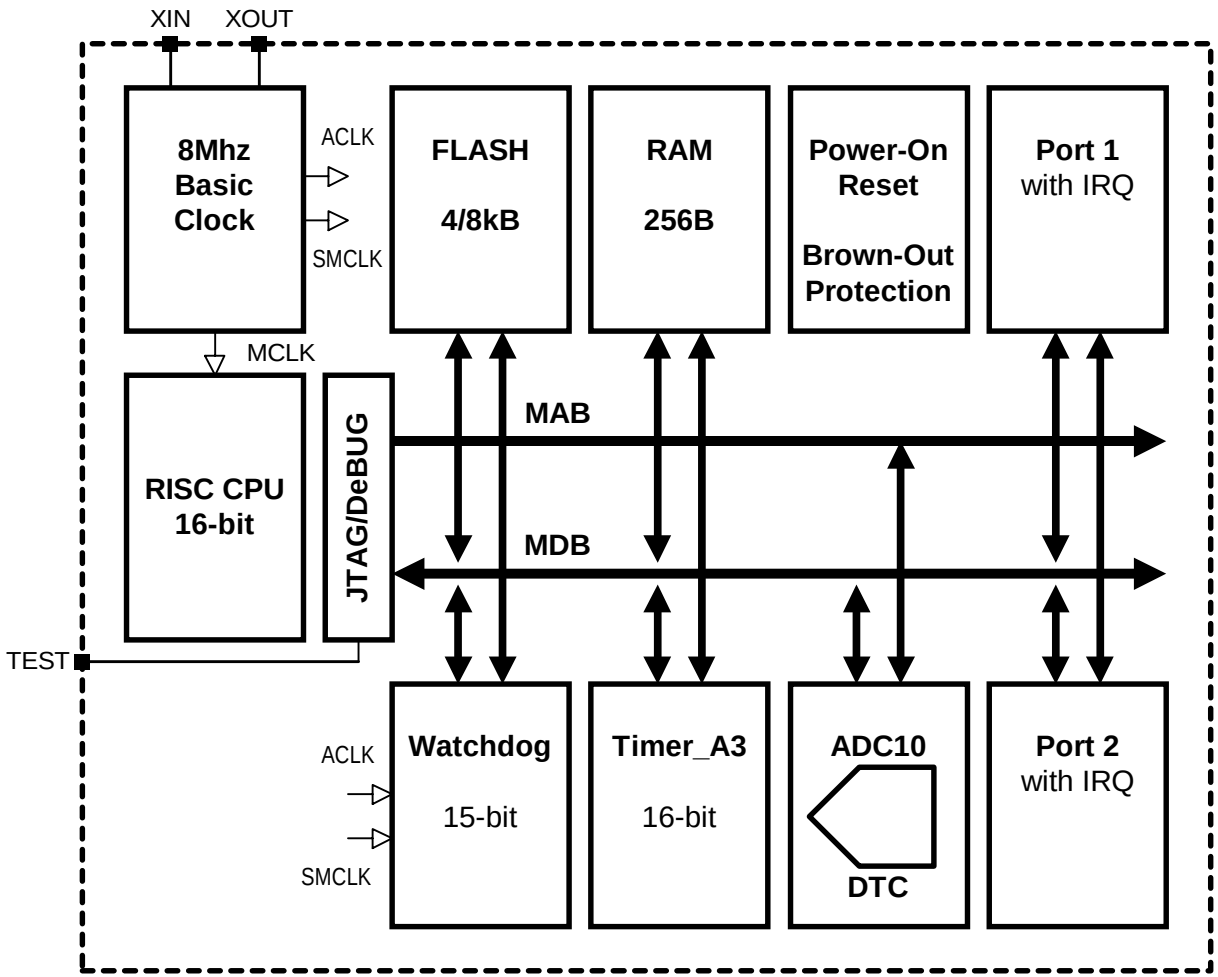
$$V1 = Vcc \times \frac{R\_NTC}{R\_NTC + R\_Ref}$$

$$N = \frac{V1}{Vcc} \times 1024 = \frac{R\_NTC}{R\_NTC + R\_Ref} \times 1024$$



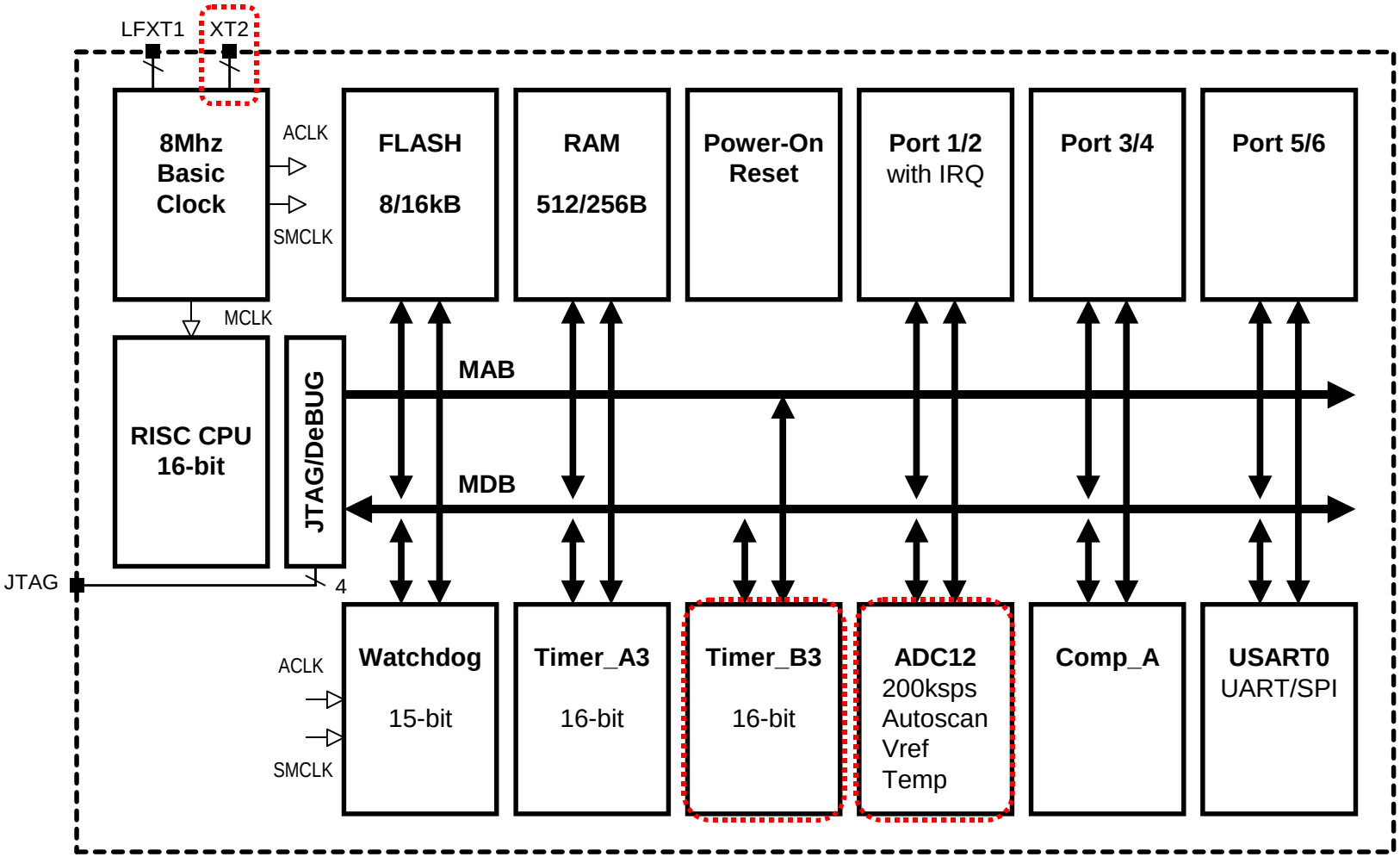
$$R\_NTC = Rref \times \frac{N}{1024 - N}$$

MSP430F11x2



20 SOIC/TSSOP

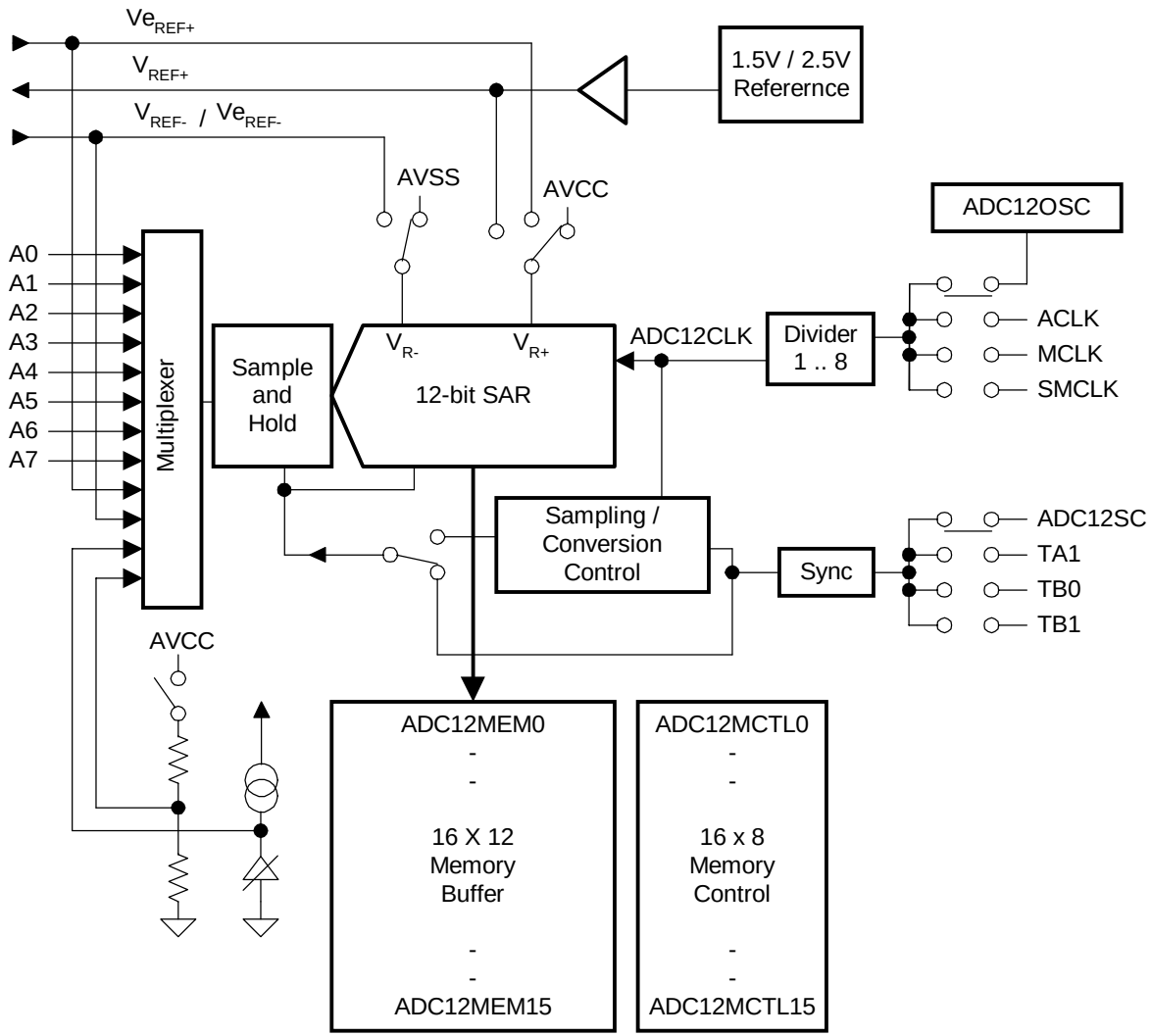
MSP430x13x



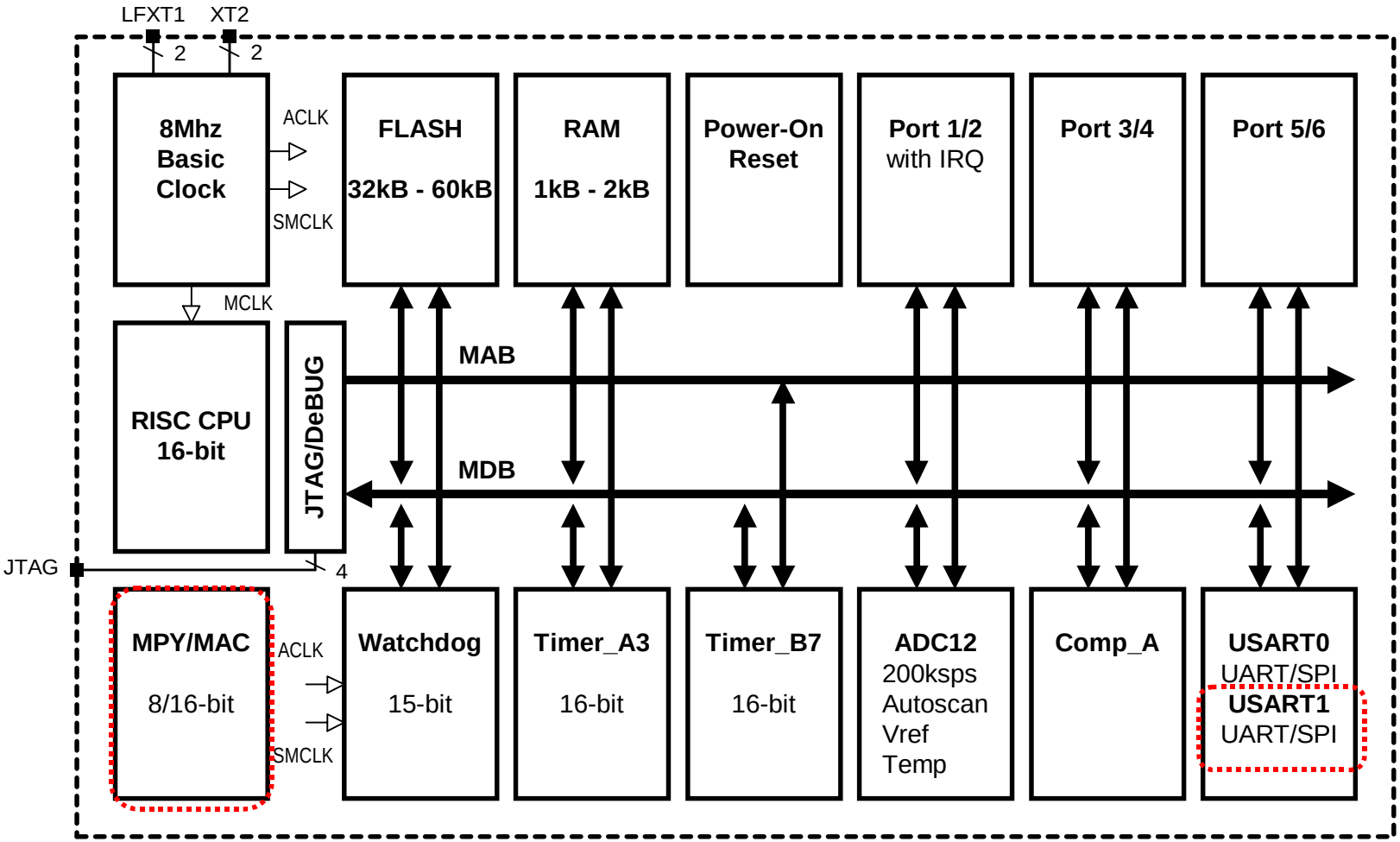
64 TQFP

ADC12 Introduction

- 200ksps+
- Programmable sample & hold
- Band-gap Vref
- Temp. sensor
- Low-battery detect
- Auto-scan
- 16 word conversion buffer
- One interrupt vector generator with enable and flags in module register



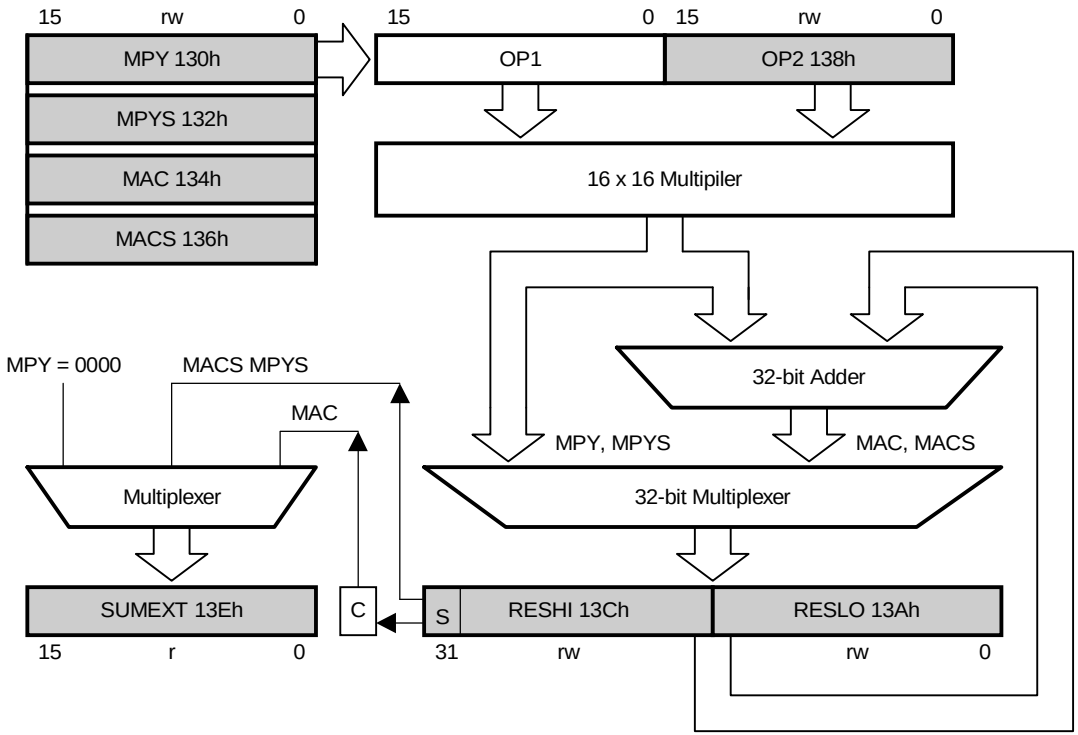
MSP430x14x



64 TQFP

'14x - Only

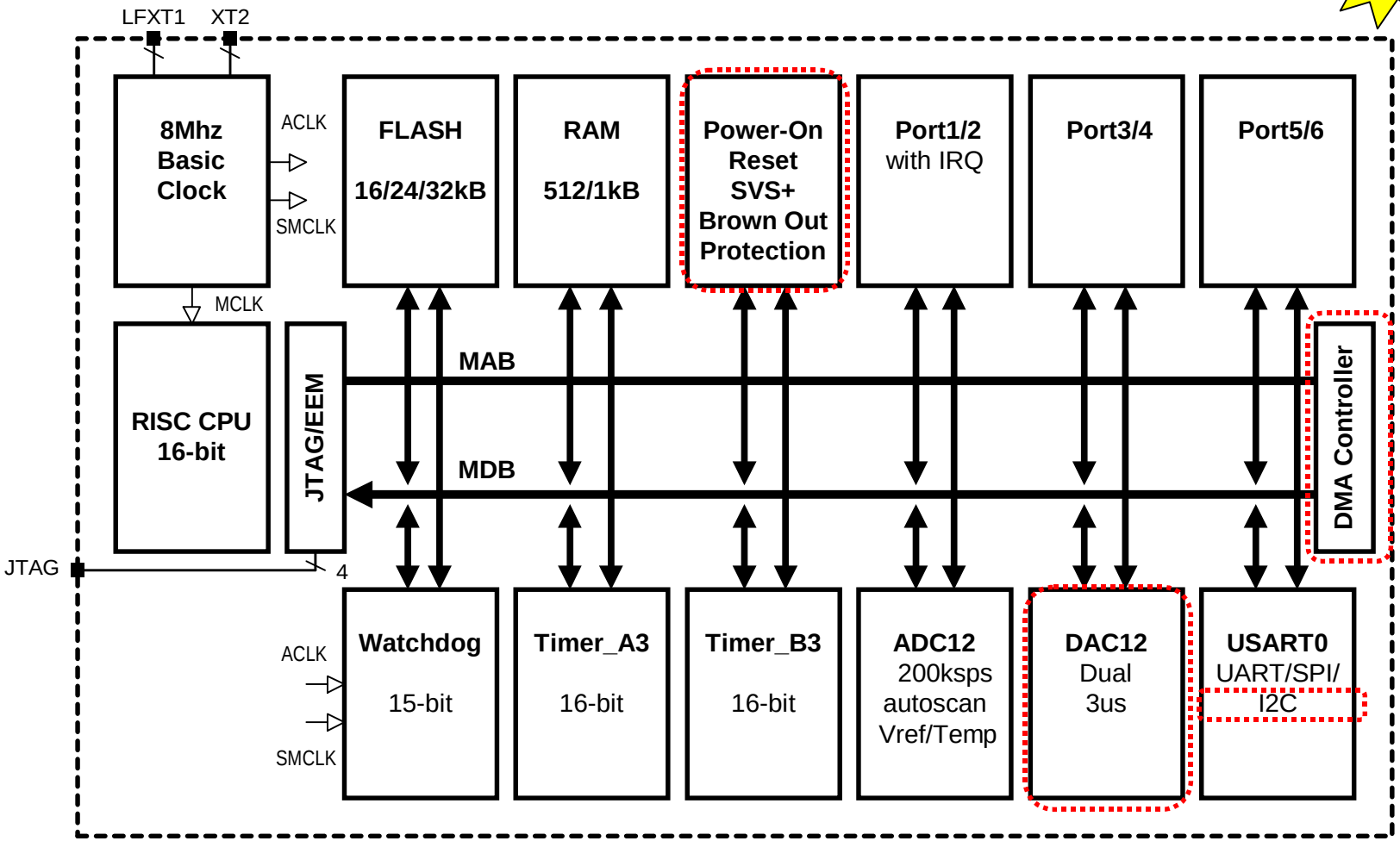
Hardware Multiplier




```
MACS = 1234;        // Load first Operand
OP2 = 4567;         // Load second Operand

    ; Done Result is ready in RESHI / RESLO
```

MSP430F15x

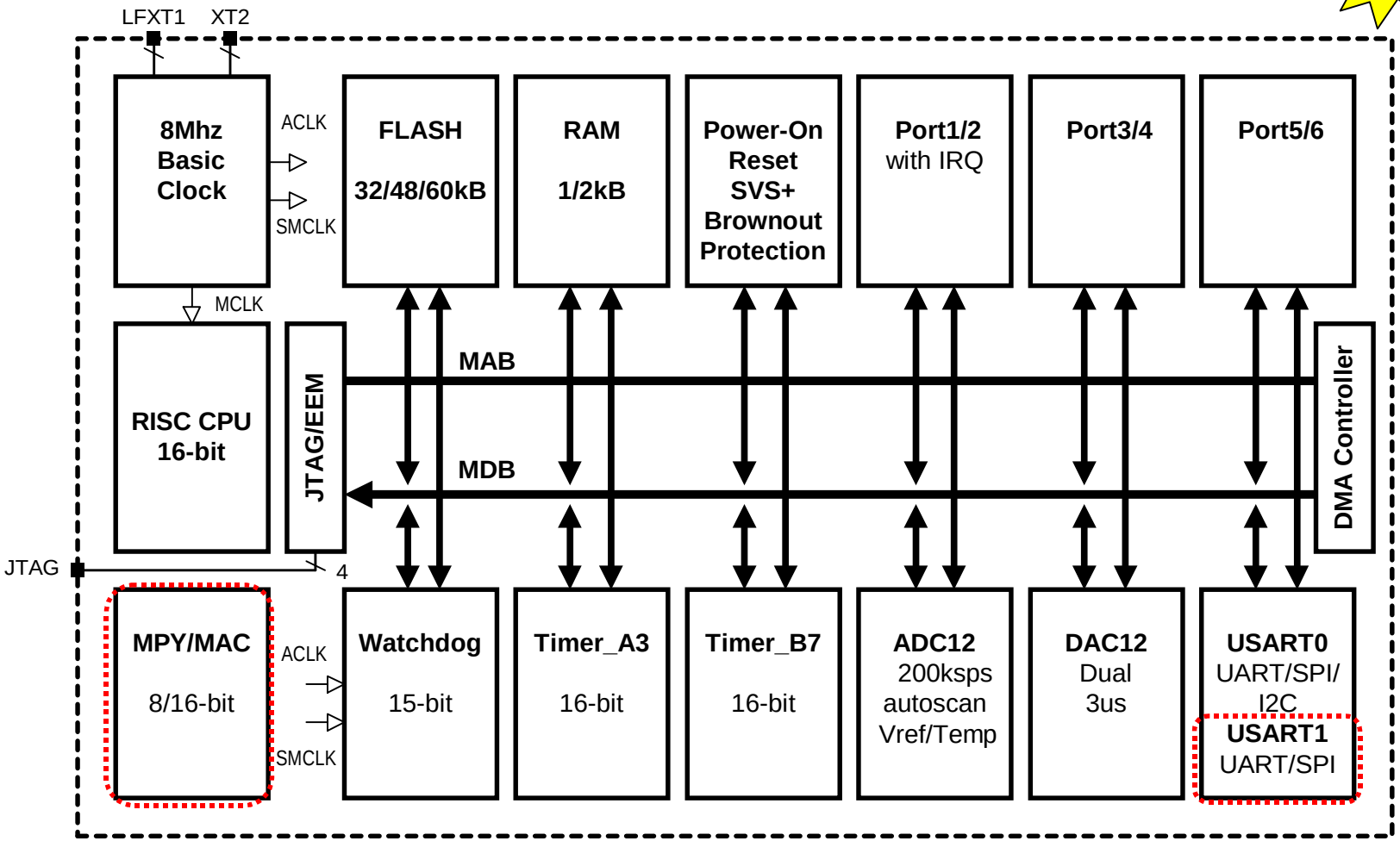
Preview



64 TQFP

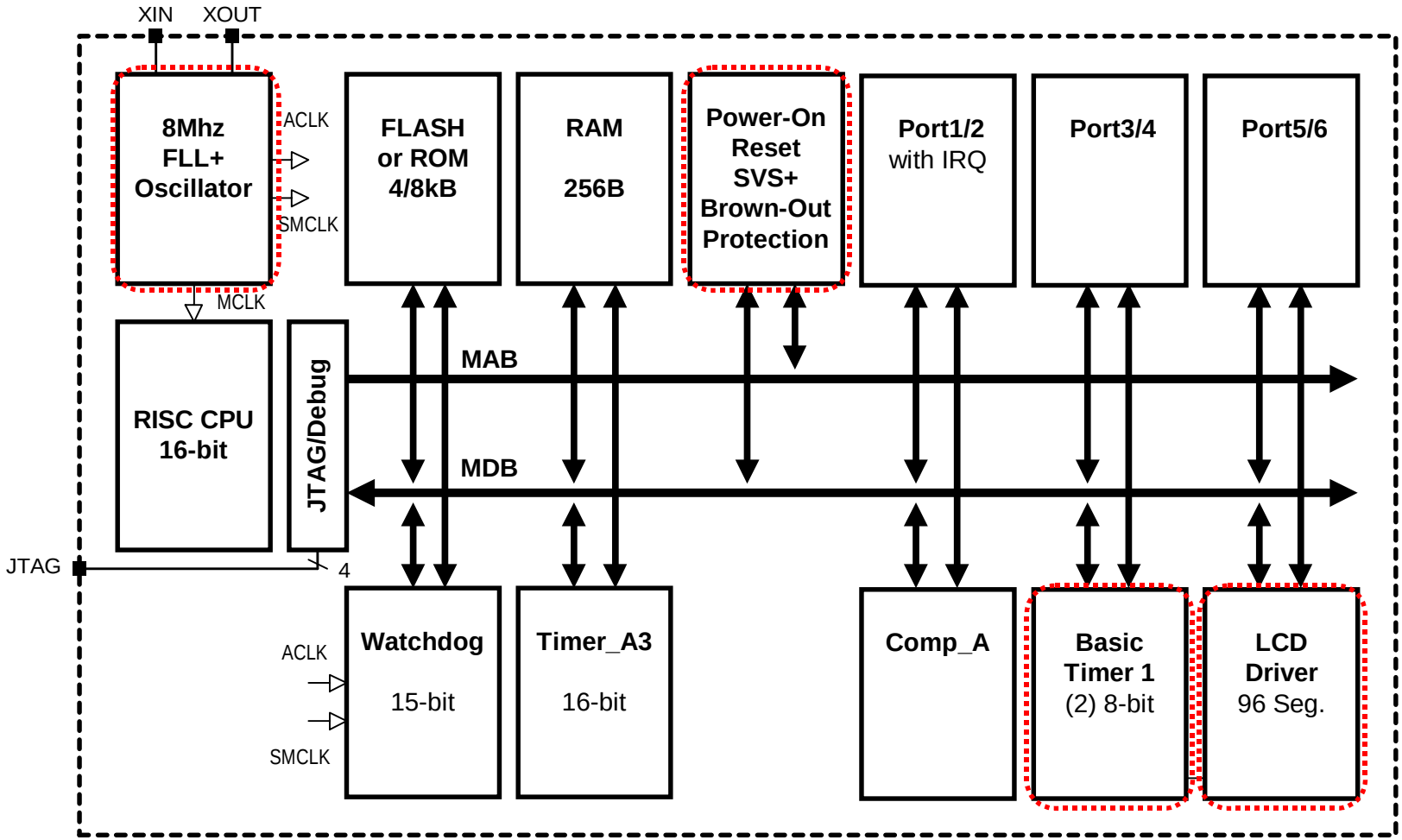
MSP430F16x

Preview



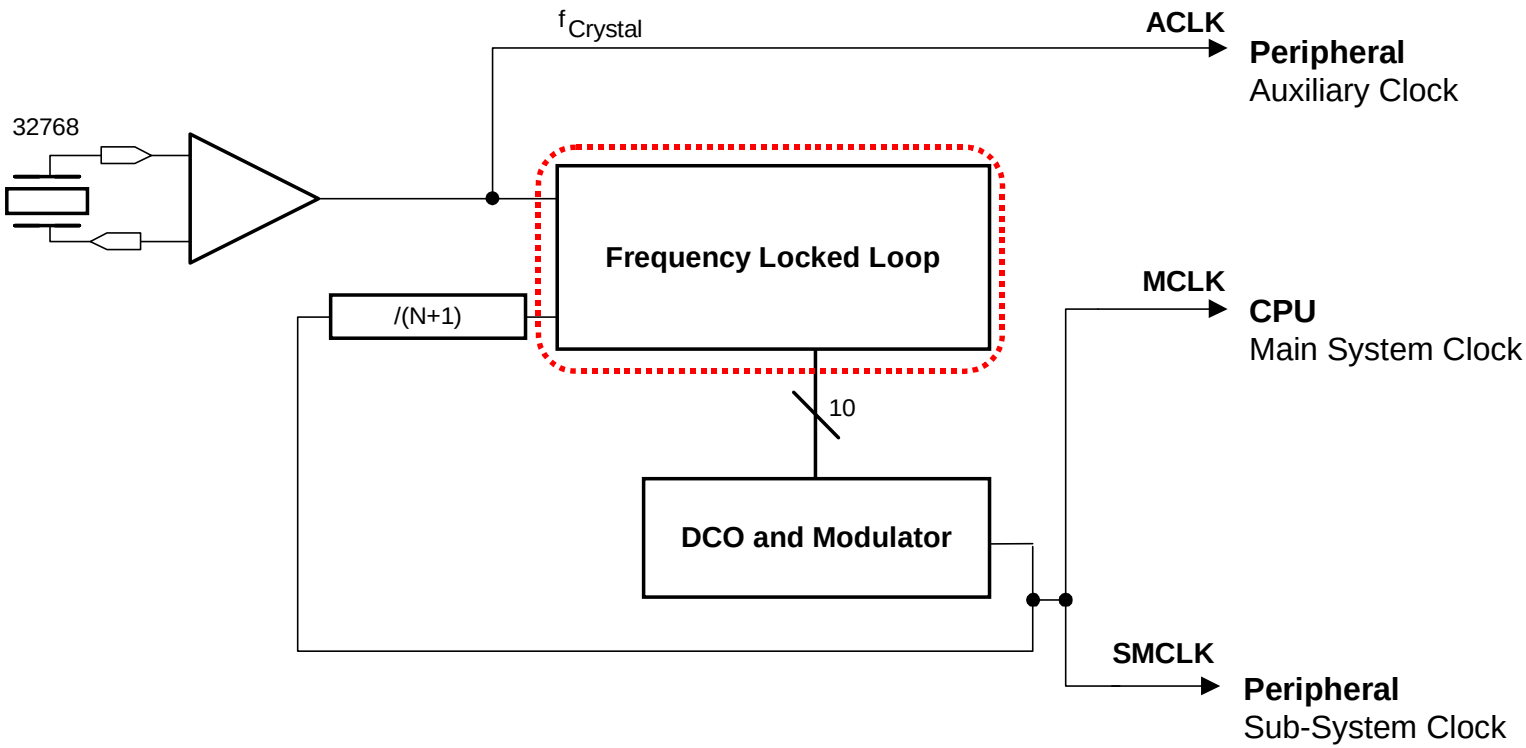
64 TQFP

MSP430x41x



64 TQFP

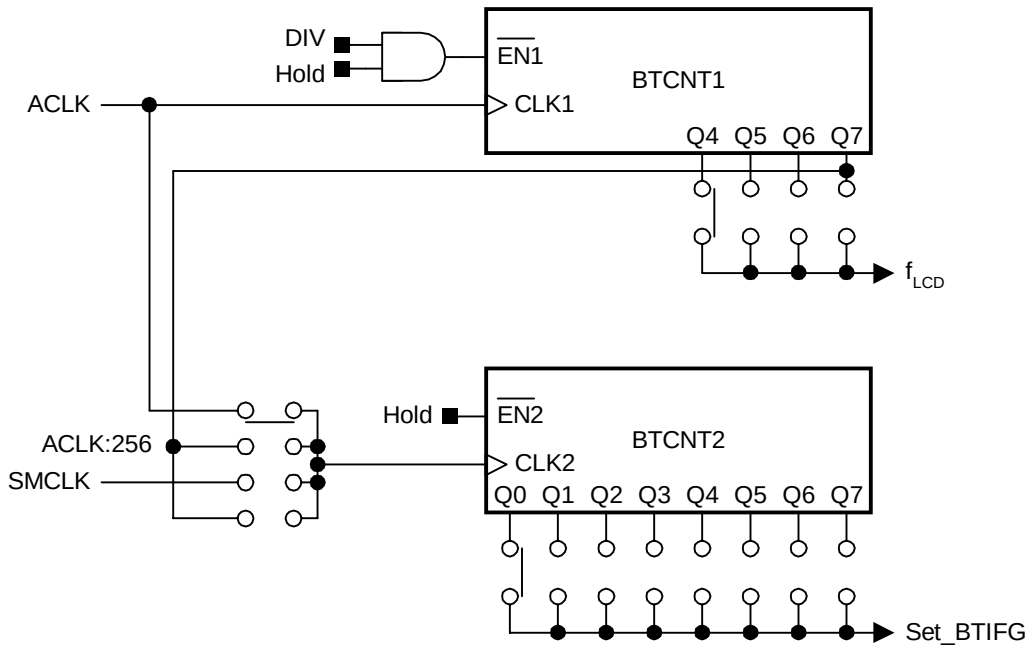
MSP430x41x/42x FLL Concept



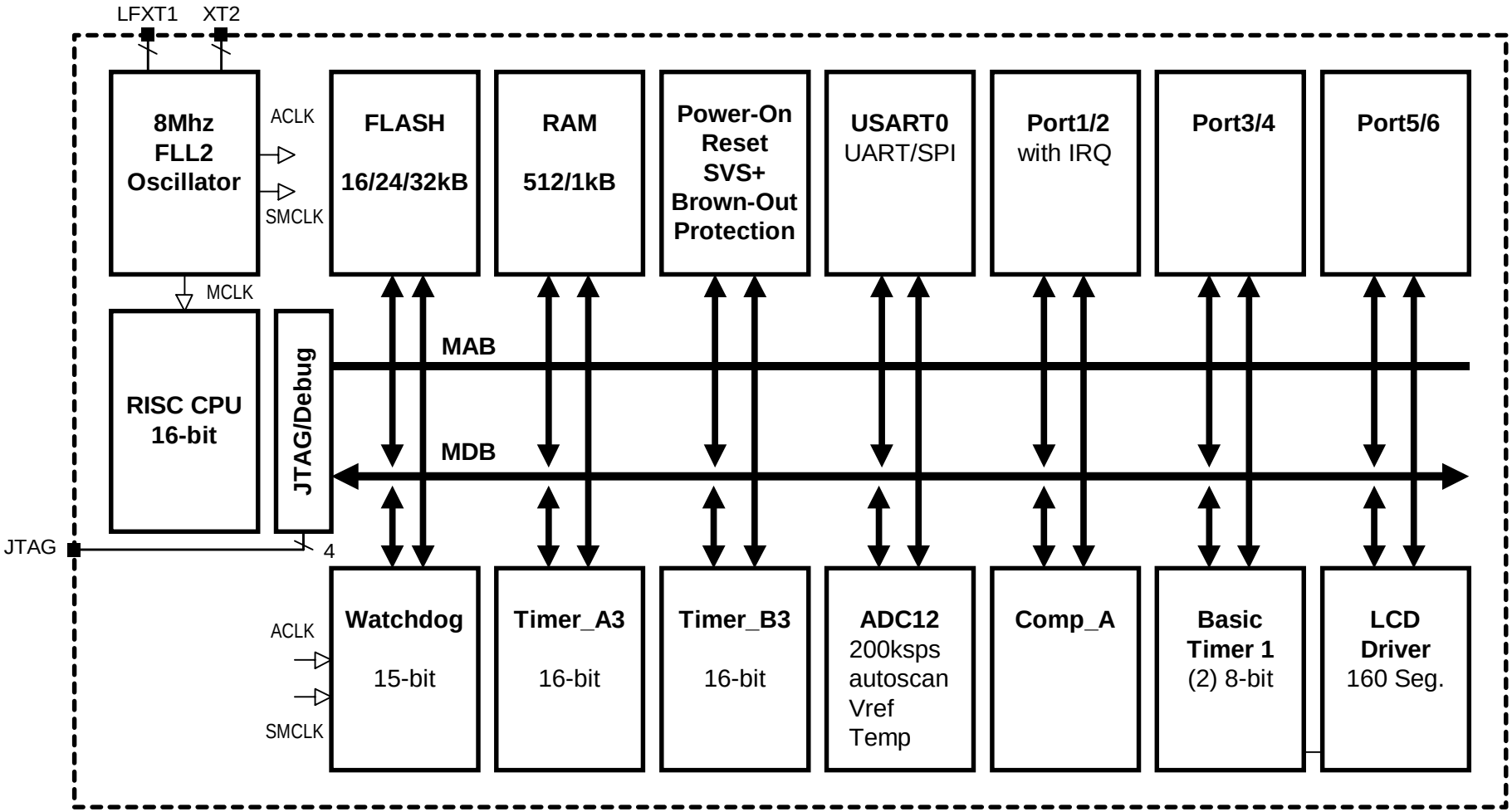
DCO generated clock is automatically enabled and stable from low-power modes in less than 6uS - US Patent # 5,877,641

MSP430x4xx Basic Timer Introduction

- One 16-bit or two 8-bit timers
- Support for real-time clock interrupt
- LCD Drive frequency
- One interrupt vector with enable and flag in SFR registers

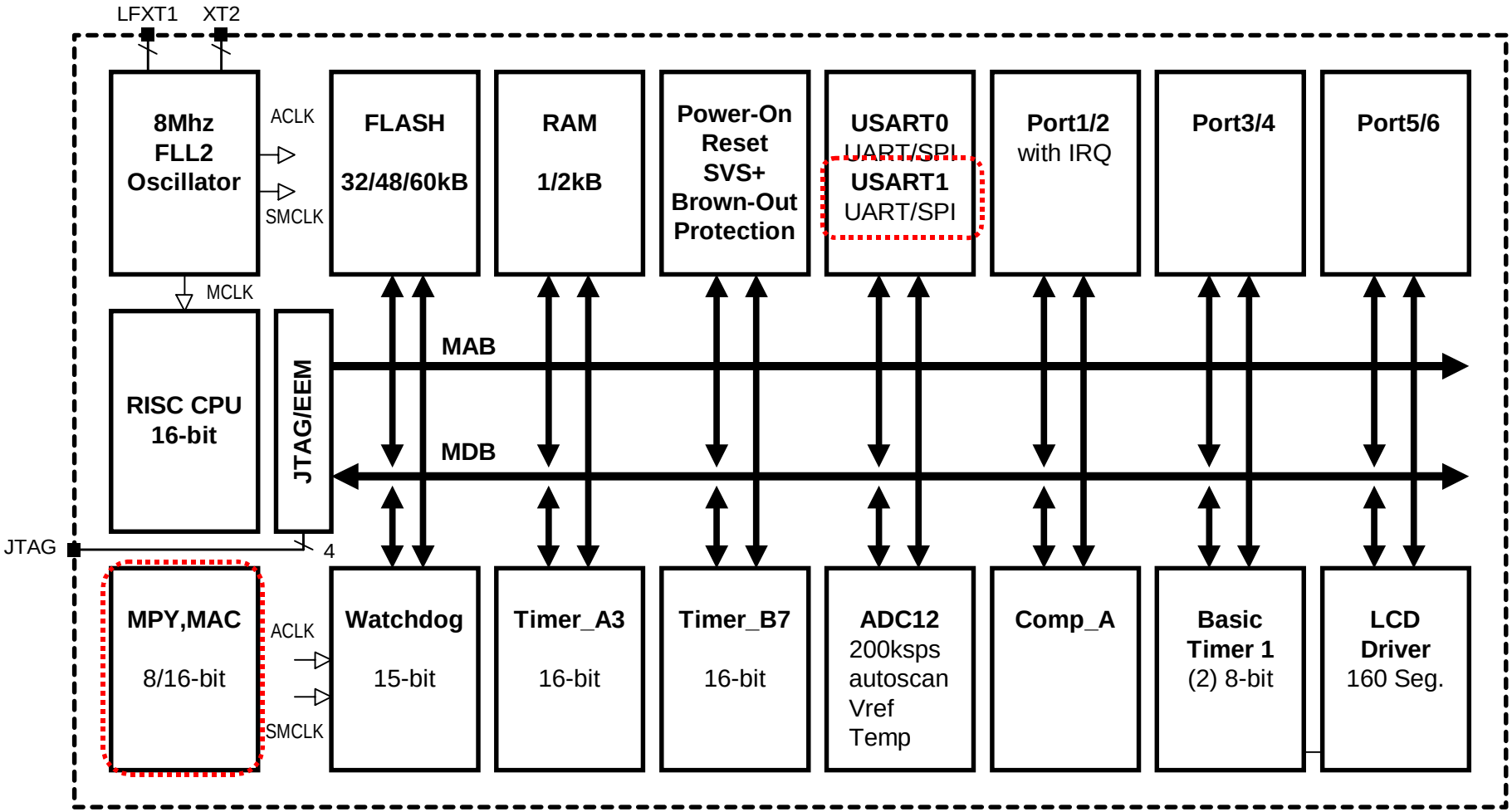


MSP430F43x



80/100 TQFP

MSP430F44x



100 TQFP

Using Standard Bit Definitions

- Which code line turns off the watchdog timer???

```
WDTCTL = 0x5A80;

WDTCTL = 0xA580;

WDTCTL = 0xA540;

WDTCTL = WDTPW + WDTHOLD     ; // Stop watchdog timer
```

- Using standard bit definitions simplifies coding and debugging

Efficient MSP430 C Coding tips

- Local variables are allocated in registers - code efficient and lower power
- Use globals sparingly!

```
// Example 1
unsigned int i;              // assigned in RAM
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD; // Stop watchdog
  P1DIR |= 0x01;            // P1.0 to output
  for (;;)
  {
    P1OUT ^= 0x01;          // Toggle P1.0
    i = 50000;              // Delay
800E  B24050C30002  mov.w  #0c350h,&i
    do (i--);
8014  ?0004:
8014  B2530002      add.w  #0ffffh,&i
    while (i != 0);
  }
}
8018  82930002     tst.w  &i
801C  FE23         jne    ?0004
```

```
// Example 2
void main(void)
{
  unsigned int i;           // Assigned to R12
  WDTCTL = WDTPW + WDTHOLD; // Stop watchdog
  P1DIR |= 0x01;            // P1.0 to output
  for (;;)
  {
    P1OUT ^= 0x01;          // Toggle P1.0
    i = 50000;              // Delay
800E  3C4050C3  mov.w  #0c350h,R12
    do (i--);
8012  ?0004:
8012  3C53      add.w  #0ffffh,R12
    while (i != 0);
  }
}
8014  FE23     jne  ?0004
```

Allocated in RAM

Allocated in Register

Efficient MSP430 C Coding tips

- The first two function parameters are passed in registers
- Use globals sparingly!

```
// Example 3
void delay(void);
unsigned short value;
void main(void)
{
 WDTCTL = WDTPW + WDTHOLD;//Stop watchdog
 P1DIR |= 0x01;           // P1.0 output
 for (;;)
 {
 P1OUT ^= 0x01;           // Toggle P1.0
 value = 50000 ;
 800E   B24050C30002
 delay ();                // Delay
 8014   B0121A80   call  #delay
  }
 }
 void delay(void)
{
do (value--);
 801A   delay:
 801A   B2530002   add.w #0ffffh,&value
 while (value != 0);
 801E   82930002   tst.w &value
 8022   FB23       jne   delay }
 8024   3041       ret
```

RAM access

```
// Example 4
void delay(unsigned short value);
void main(void)
{
 WDTCTL = WDTPW + WDTHOLD;// Stop watchdog
 P1DIR |= 0x01;           // P1.0 output
 for (;;)
 {
 P1OUT ^= 0x01;           // Toggle P1.0
 delay (50000);           // Delay
 800E   3C4050C3   mov.w 0x0c350h,R12
 8012   B0121880   call  #delay
  }
 }
 void delay(unsigned short value)
 {
do (value--);
 8018   delay:
 8018   3C53       add.w #0ffffh,R12
 while (value != 0);
 801A   FE23       jne   delay
  }
 801C   3041       ret
```

Efficient Register Usage

Some More C Coding Tips To Consider

- Use bit mask instead of bitfields for **unsigned int** and **unsigned char** - bit masks execute faster than bitfields
- Use unsigned data types as much as possible - execute more efficiently
- Use ANSI prototypes - ANSI functions often more efficient than K&R.
- Use pointers to access structures and unions - passing structures and unions is costly!
- Use “static const” class to avoid run-time copying of structures, unions, and arrays.

Development Tool Summary

TI IDE	Product Family
MSP-FET430X110	MSP430x11x
MSP-FET430P120	MSP430x11x/12x(2)
MSP-FET430P140	MSP430x13x/14x/16x
MSP-FET430P410	MSP430x41x/42x
MSP-FET430P440	MSP430x43x/44x
MSP-EVK430S320	MSP430x31x/32x
MSP-EVK430S330	MSP430x33x
MSP-PRGS430	MSP430x - universal device programmer

Third Party	Tool
ANSI-C Compiler	www.iar.com
ANSI-C Compiler	**www.quadrovox.com**
Gang programmers	BP, Data I/O

Unlimited ANSI-C compiler with source level debug for only $395 compatible with TI JTAG FET

All TI IDE's include IAR debugger, assembler/linker and 4kB C-compiler

Support www.ti.com/sc/msp430

Documentation

SLAC001	MSP430 CD-ROM
SLAUxxx	Family User's Guide's
SLASxxx	Device Datasheets
SLASxxx	45+ Application reports
web	200+ Downloadable code examples
SLAB034	Product Bulletin
kickstart	FET User's Guide

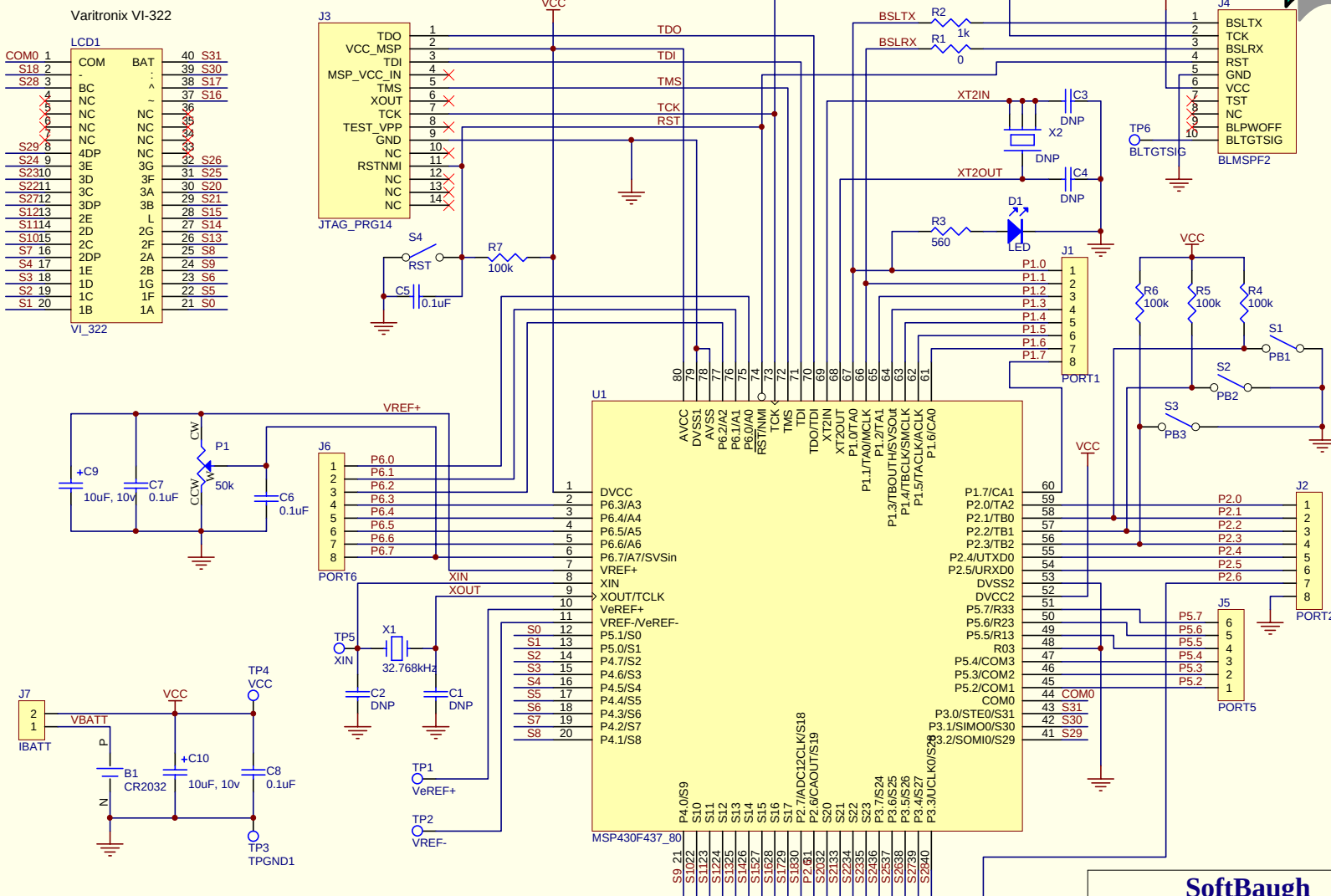
FAQ

www.ti.com/sc/knowledgebase

Product Information

Americas	+1(972) 644-5580
France	+33 (0) 1 30 70 11 64
Germany	+49 (o) 8161 80 3311
Italy	800 79 11 37
UK	+44 (0) 1604 66 33 99
Japan	+81-3-3344-5317
Asia	+886-2-23786800

D437V - MSP430F437 LCD Demo



SoftBaugh

www.softbaugh.com 800 794-5756

Sheet: MSP430F437 LCD Demo		
Project: D437V		
Date: 3-Nov-2002	1 of 1	v. 1

“Flashing the LED” Example Software

LAB0

```
//*******************************************************************************
//  D437_1 Demo - Software Toggle P1.0 by xor'ing inside of a software loop
//
//  M.Buccini
//  Texas Instruments, Inc
//  November 2002
//*******************************************************************************

#include  <msp430x43x.h>

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop watchdog timer
  FLL_CTL0 |= XCAP18PF;                     // Configure load caps
  P1DIR |= 0x01;                            // Set P1.0 to output direction

  for (;;)
  {
    unsigned int i;

    P1OUT ^= 0x01;                          // Toggle P1.0 using exclusive-OR

    i = 50000;                              // Delay
    do (i--);
    while (i != 0);
  }
}
```

“Flashing the LED” Questions

LAB0

- Did the LED Flash?
  - Yes
  - No
- Where you able to set a breakpoint?
  - Yes
  - No
- Change SCFQCTL from 0x1F to 0x7F - what happens to the LED Flash Rate?
  - Faster
  - Slower
- What is the average current consumption of your D437?
  - _____ uA

Notes

Agenda - Dallas, TX - November 2002

Flash MCU ULP Architecture

16-bit RISC CPU:

- von Neumann unified memory architecture
- CPU register file
- Instruction set and addressing modes

Flash Memory:

- Basic facts including data retention, programming voltage and algorithms
- In-system programming (ISP)
- Using Flash as virtual EEPROM

Clock System Analysis:

- Importance of a multi-clock system in low-power applications
- MSP430x1xx Basic Clock System (BCS) implementation
- Ultra-low power 32kHz oscillator characteristics
- Detailed discussion of the digitally controlled oscillator (DCO) and modulator
- Stabilizing the BCS DCO with a software frequency locked loop (FLL) and Rosc
- MSP430x4xx FLL hardware implementation
- Using high-speed crystals

LAB Session 1

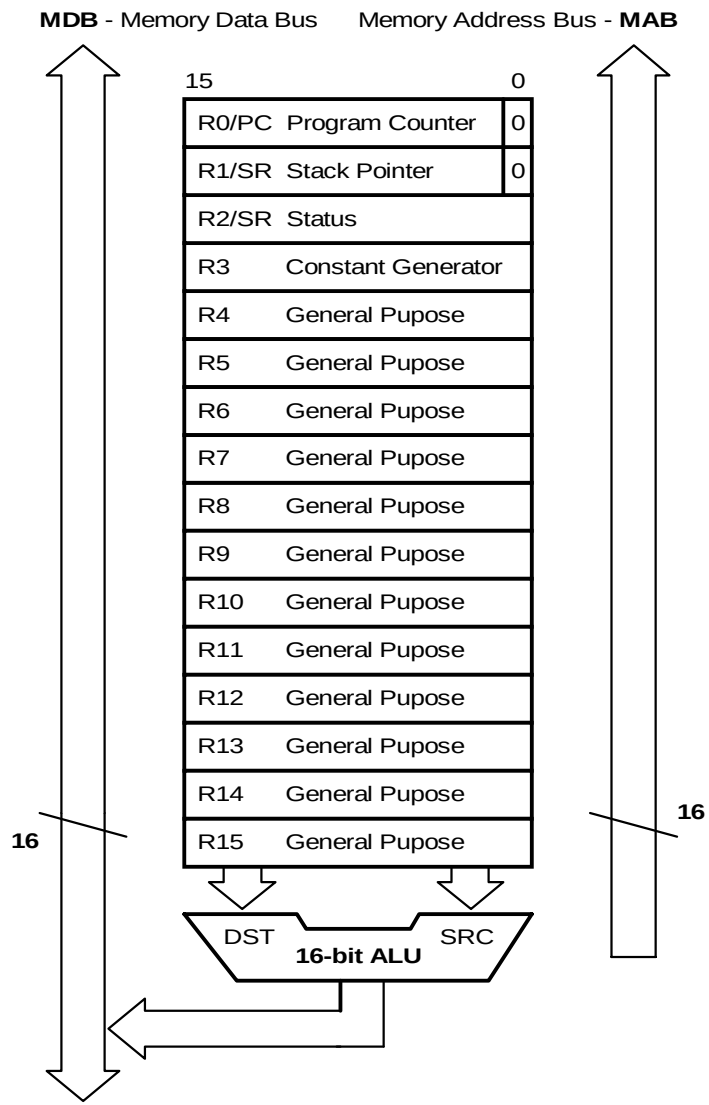
Lab: UART application to demonstrate BCS and FLL DCO stabilization
Lab: Calibrating the ADC12 internal voltage reference using ISP

What Is An *Ultra-low Power* Architecture?

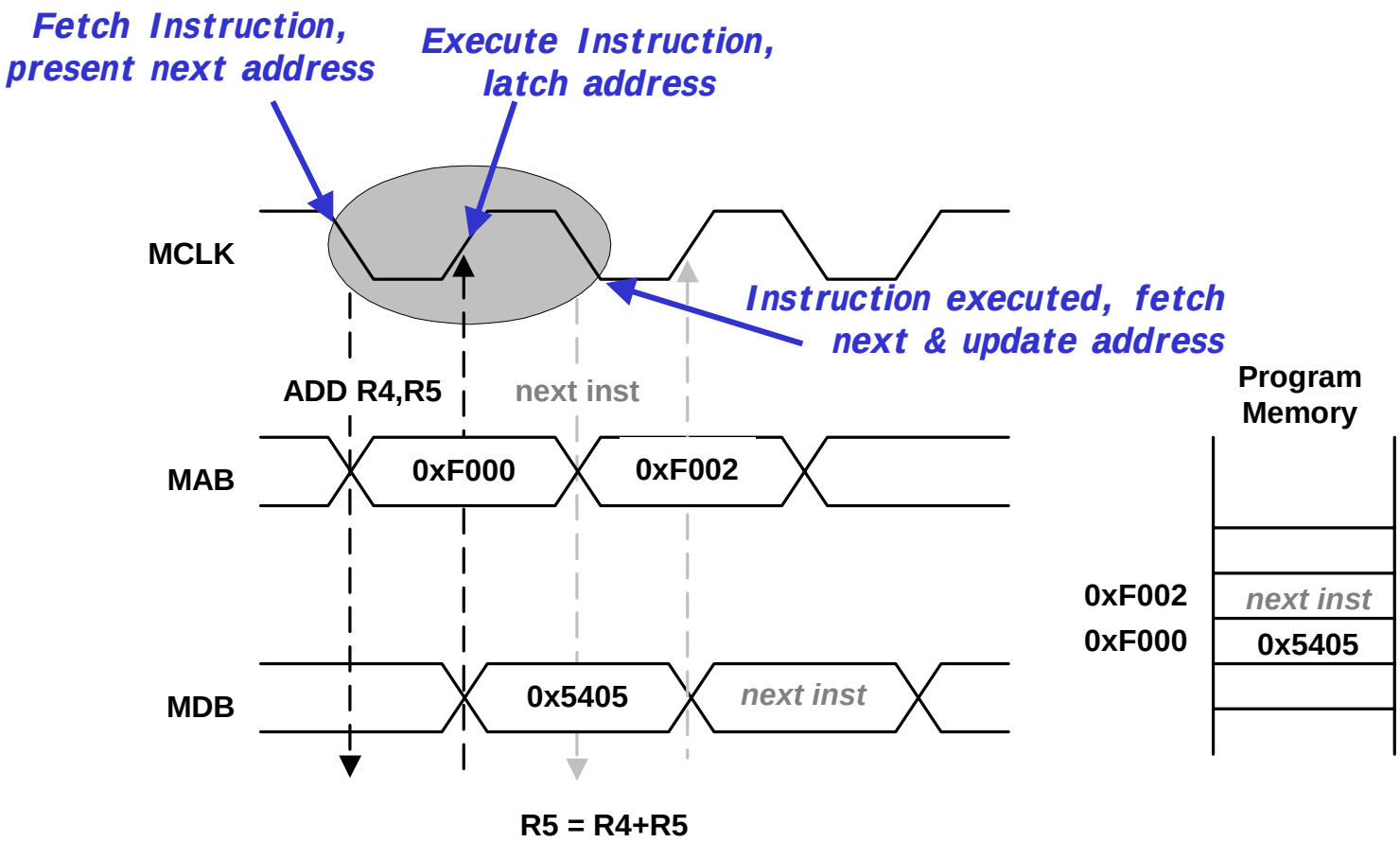
- **Instruction set that minimizes CPU cycles per task**
  - Large register file keeps information in the CPU, minimizes fetches to memory
  - Indirect addressing for fast table look-up
  - Indirect-indirect addressing allows memory to memory movement when needed
  - 16-bit address path allows page-free branching
  - 16-bit data path allows direct movement and operation on high resolution data
- **Ability to use alternate hardware that provides low-power options for tasks**
  - Asynchronous timers with capture/compare and latching
  - Direct transfer controller, DMA and ADC auto-scan
- **Fast vectored interrupts from all peripherals**
- **Flexible asynchronous clocking with fast start-up and stabilization**
- **Low leakage input pins < 50nA**

MSP430 Orthogonal 16-bit RISC CPU

- Large 16-bit register file eliminates single accumulator bottleneck
- High-bandwidth 16-bit data and address bus with no paging
- RISC architecture with 27 instructions and 7 addressing modes
- Single-cycle register operations with full-access
- Direct memory-memory transfer designed for modern programming
- Compact silicon 30% smaller than an '8051, saves power and cost

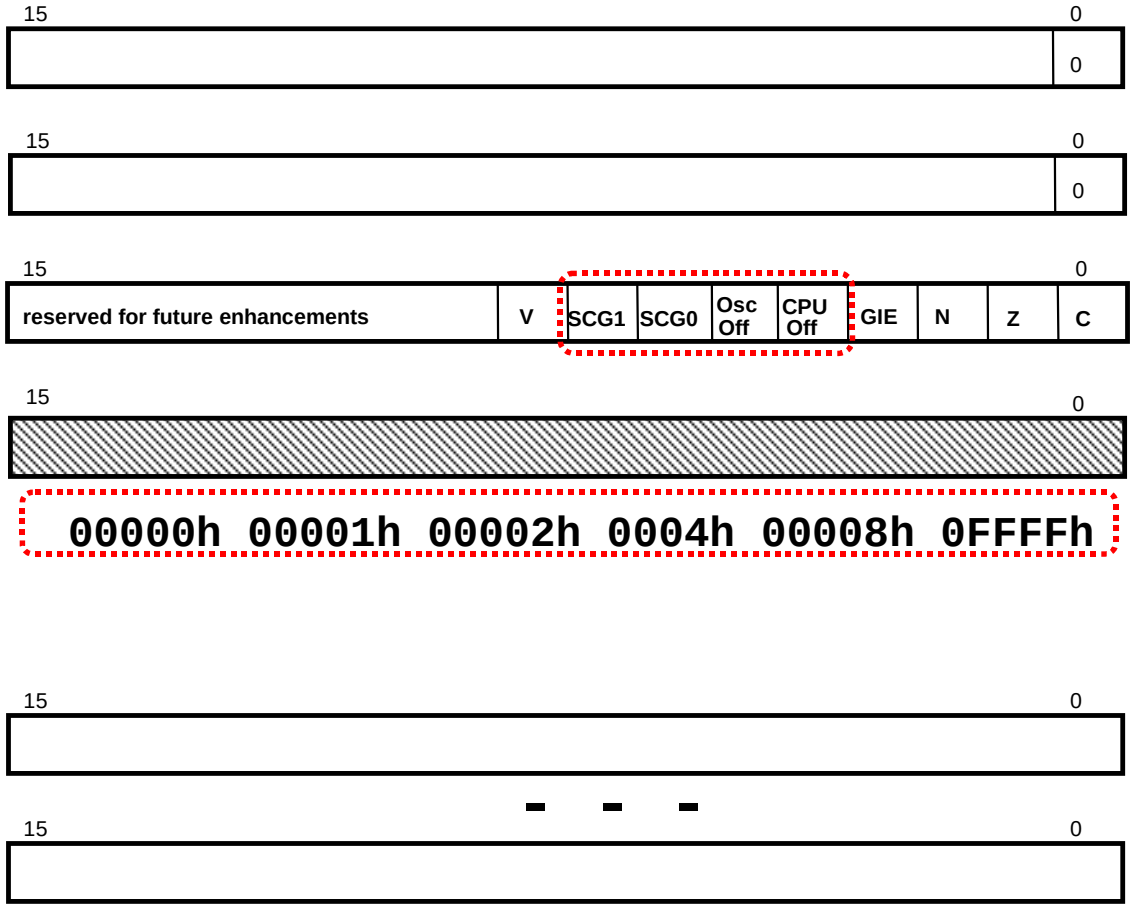


Double Data Fetch Technology - DDFT



DDFT uses double memory access on both rising and falling processor clock edges allowing one instruction to be executed for every clock

CPU Registers



R0 - PC Program Counter
16-bit = no paging

R1 - SP Stack Pointer
Addressable = great "C" code

R2 - SR Status Register
Define LPMx

R3/R2 - CG Constant Generator
automatic generation of common used values reduces code size 30%

R4 - General Purpose

R15 - General Purpose

R4 through R15 are single-cycle, general purpose and identical in all respects - used for math, storage, and addressing modes.

What is a Constant Generator?

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

The assembler uses the R2 / R3 addressing modes automatically if one of the above six constants is used.

RISC / CISC Programming Architecture

- **Three instruction formats**
  - Source, destination
  - Destination
  - Jumping
- **Fifty-one instructions available in assembler**
  - 27 basic instructions ⇨ ***RISC***
  - 24 emulated instructions ⇨ ***CISC***
- **Seven addressing modes for source, four for destination**

	source	destination
Register Mode	✓	✓
Indexed Mode	✓	✓
Symbolic Mode	✓	✓
Absolute Mode	✓	✓
Indirect Mode	✓	
Indirect-autoincrement Mode	✓	
Immediate Mode	✓	

- **Bit, byte and word processing**

27 Core RISC Instructions

Format I Source,Destination	Format II Single Operand	Format III +/- 9bit Offset
add(.b)	call	jmp
addc(.b)	swpb	jc
and(.b)	sxt	jnc
bic(.b)	push(.b)	jeq
bis(.b)	reti	jne
bit(.b)	rra(.b)	jge
cmp(.b)	rrc(.b)	jl
dadd(.b)		jn
mov(.b)		
sub(.b)		
subc(.b)		
xor(.b)		

Emulated Instructions

- Simply easier to understand with no code size or speed penalty
- Replaced by assembler with core instructions using CG, PC and SP

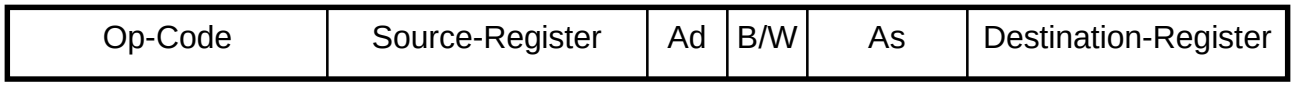
```
clrc                    ; Clear carry (emulated)
bic.w    #01h,SR        ; Core instruction

dec.w    R4             ; Decrement (emulated)
sub.w    #01,R4         ; Core instruction

ret                     ; Return (emulated)
mov.w    @SP+,PC        ; Core instruction
```

51 Total Instructions

Format I Source,Destination	Format II Single Operand	Format III +/- 9bit Offset	Support
add(.b)	**br**	jmp	**clrc**
addc(.b)	call	jc	**setc**
and(.b)	swpb	jnc	**clrz**
bic(.b)	sxt	jeq	**setz**
bis(.b)	push(.b)	jne	**clrn**
bit(.b)	**pop.(b)**	jge	**setn**
cmp(.b)	rra(.b)	jl	**dint**
dadd(.b)	rrc(.b)	jn	**eint**
mov(.b)	**inv(.b)**		**nop**
sub(.b)	**inc(.b)**		**ret**
subc(.b)	**incd(.b)**		reti
xor(.b)	**dec(.b)**		
	decd(.b)		
	adc(b)		
	sbc(.b)		
	clr(.b)		
	dadc(.b)		
	rla(.b)		
	rlc(.b)		
	tst(.b)		

Three Instruction Formats

; Format I Source and Destination




```
5405          add.w   R4,R5                 ; R4+R5=R5 xxxx
5445          add.b   R4,R5                 ; R4+R5=R5 00xx
```

; Format II Destination Only


```
6404          rlc.w   R4                    ;
6444          rlc.b   R4                    ;
```

; Format III There are 8(Un)conditional Jumps


```
3c28           jmp     Loop_1               ; Goto Loop_1
```

Register Addressing Mode

Op-Code	Source-Register	Ad	B/W	As	Destination-Register
0100	**0100**	**0**	**0**	**00**	**0101**

```
4405            mov.w   R4,R5                   ;

4445            mov.b   R4,R5                   ;
```

Valid for Source and destination As=00, Ad=0
The operand is contained in one of the CPU registers R0 to R15.
This is the fastest addressing mode and needs the least memory .

Register Indexed Addressing Mode

Op-Code	Source-Register	Ad	B/W	As	Destination-Register
0100	**0100**	**1**	**0**	**01**	**0101**

```
4495 01000200   mov.w   100h(R4),200h(R5) ;

44150100        mov.w   100h(R4),R5       ;
```

Valid for Source and destination As=01, Ad=1
The address of the operand is the sum of the index and the contents of the register.

Symbolic Addressing Mode

Op-Code	Source-Register	Ad	B/W	As	Destination-Register
0100	0000	1	0	01	0000

```
4090ffa80006   mov.w   EDE,TONI          ;

4015ffac       mov.w   EDE,R5            ;
```

Source and destination As=01, Ad=1
The content of the addresses EDE / TONI are used for the operation. The source or destination address is computed as a difference from the PC and uses the PC in indexed addressing mode. Any address in the 64k memory space is addressable.

Absolute Addressing Mode

Op-Code	Source-Register	Ad	B/W	As	Destination-Register
0100	**0010**	**1**	**0**	**01**	**0010**

```
4292 01720174   mov.w   &CCR0,&CCR1        ;

42150172        mov.w   &CCR0,R5           ;
```

Source and destination As=01, Ad=1
The contents of the fixed addresses are used for the operation.
The SR is used in the indexed mode to create an absolute 0. Use for hardware peripherals located at an absolute address that can never be relocated.

Register Indirect Addressing Mode

Op-Code	Source-Register	Ad	B/W	As	Destination-Register
0100	0100	0	0	10	0101

```
4425            mov.w   @R4,R5              ;

4465            mov.b   @R4,R5              ;
```

Source only As=10, Ad=n/a

The registers are used as a pointer to the operand.

The indexed mode with zero index may be used for “indirect register addressing” of the destination operand.

```
44a50000        mov.w   @R4,0(R5)           ;
```

Register Indirect Autoincrement Addressing Mode

Op-Code	Source-Register	Ad	B/W	As	Destination-Register
0100	0100	0	0	11	0101

```
4435            mov.w   @R4+,R5                 ;

4475            mov.b   @R4+,R5                 ;
```

Source only As=11, Ad=n/a
The registers are used as a pointer to the operand. The registers are incremented afterwards - by 1 in byte mode, by 2 in word mode.

Immediate Addressing Mode

Op-Code	Source-Register	Ad	B/W	As	Destination-Register
0100	**0000**	**0**	**0**	**11**	**0101**

```
4035 1234      mov.w   #1234h,R5          ; Any 16-bit value
```

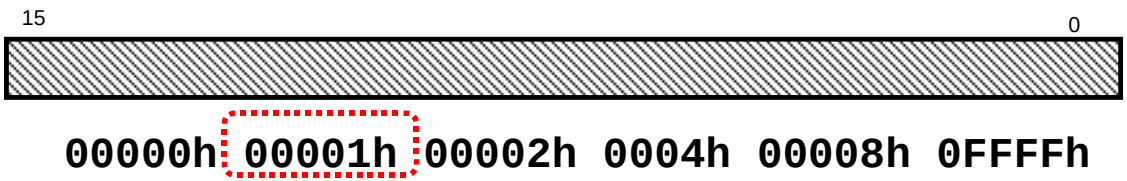
Source only As=11, Ad=n/a

Any immediate 8 or 16 bit constant can be used with the instruction. The PC is used in autoincrement mode to emulate this addressing mode.

Code Reduction Effect of Constant Generator

Op-Code	Source-Register	Ad	B/W	As	Destination-Register
0100	0011	0	0	01	0100

```
4314        mov.w   #0001h,R4           ;
```



R3/R2 - CG Constant Generator

automatic generation of commonly used values reduces code size 30%

Machine Cycles for Format I Instructions

Address Mode As	Address Mode Ad	#-of-Cycles	Length of Instruction [words]	Examples
00, Rn	0, Rm	1	1	MOV R5,R8
	0, PC	2	1	BR R9
00, Rn	1, x(Rm)	4	2	ADD R5,2(R6)
	1, EDE			XOR R8,EDE
	1,&EDE			MOV R5,&EDE
01,x(Rn)	0, Rm	3	2	MOV 2(R5),R7
01, EDE				AND EDE,R6
01,x(Rn)	1,x(Rm)	6	3	ADD 4(R4),6(R9)
01,EDE	1,TONI			CMP EDE,TONI
01,&EDE	1,&EDE			MOV R5,&EDE
10,@Rn	0, Rm	2	1	AND @R4,R5
10,@Rn	1,x(Rm)	5	2	XOR @R5,8(R6)
	1, EDE			MOV @R5,EDE
	1,&EDE			XOR @R5,&EDE
11,@Rn+	0,Rm	2	1	ADD @R5+,R6
	0, PC	3		BR @R5+
11,#N	0,Rm	2	2	MOV #20,R9
	0,PC	3		BR #2AEh
11,@Rn+	1,x(Rm)	5	2	MOV @R9+,2(R4)
11,#N	1,EDE		3	ADD #33,EDE
11,@Rn+	1,&EDE		2	MOV @R9+,&EDE
11,#N			3	ADD #33,&EDE

Machine Cycles for Format II Instructions

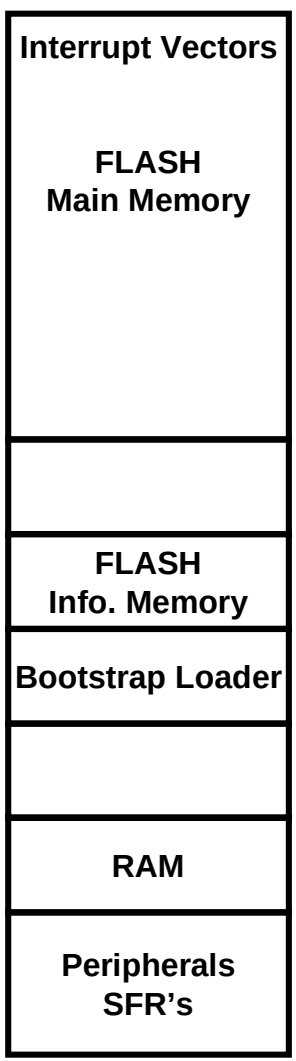
Address Mode As/Ad	#-of-Cycles RRA RRC SWPB SXT	#-of-Cycles PUSH/ CALL	Length of Instruction [words]	Examples
00, Rn	1	3/4	1	SWPB R5
01, x(Rn) 01, EDE	4	5/5	2	CALL Table(R7) PUSH EDE
10, @Rn	3	4/4	1	RRC @R9
11, @Rn+ 11, #N	3	4/5	1 2	SWPB @R10+ CALL 2(R7)

Machine Cycles for Format III Instructions

All Jxx - instructions need the same #-of-cycles independent of executing a Jump

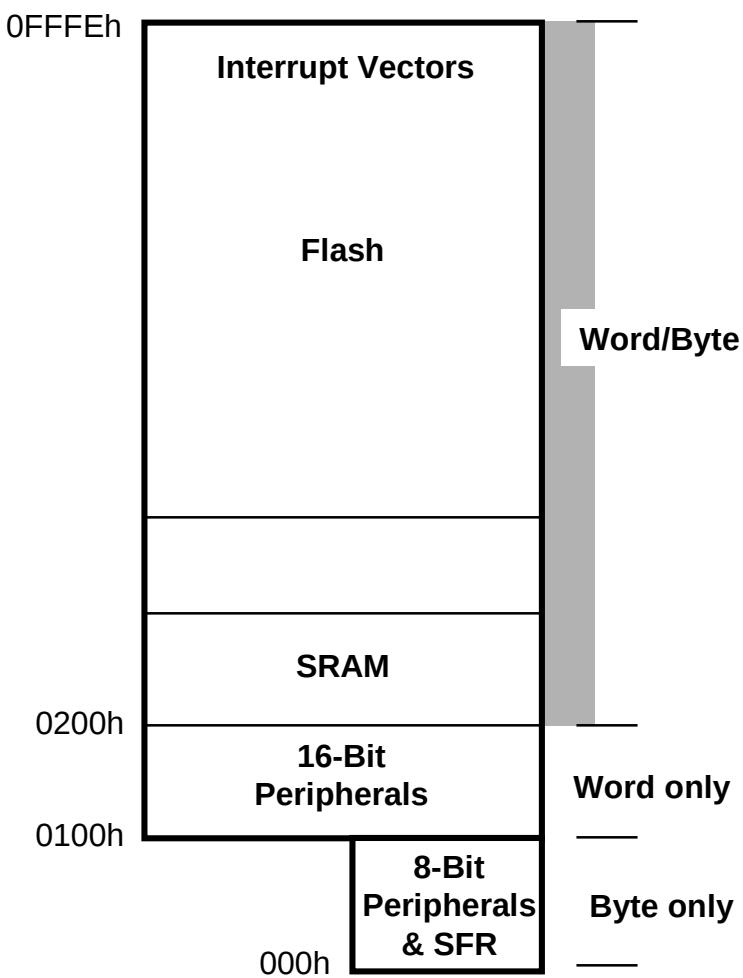
- Clock Cycles: 2
- Length of Instruction: 1 word

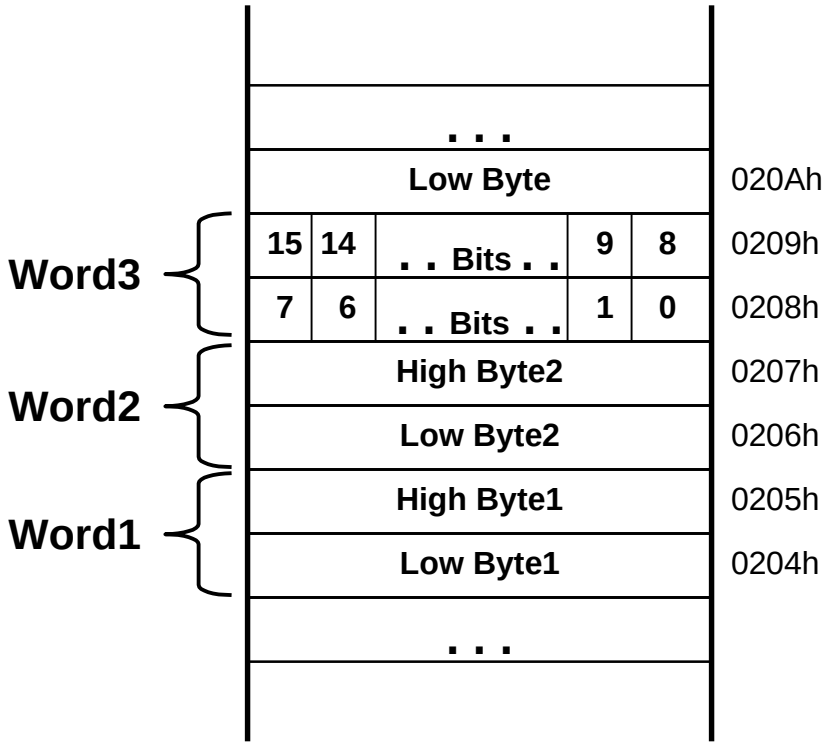
MSP430 Memory Model



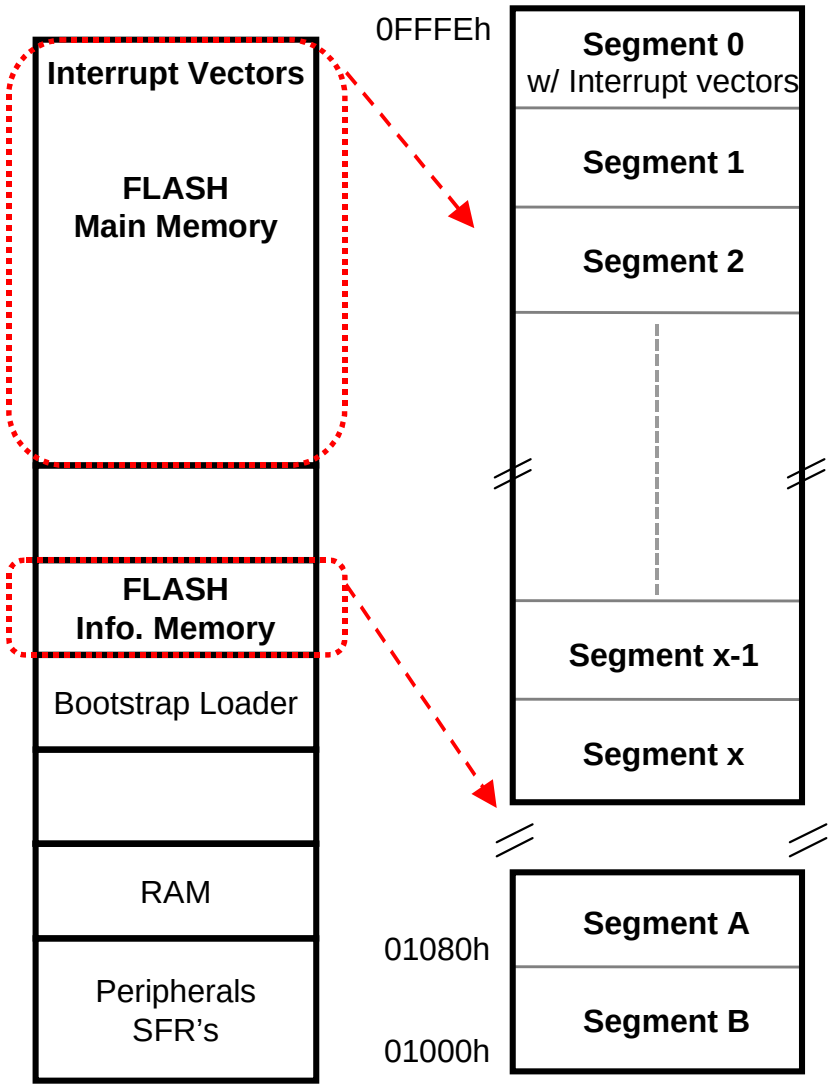
- Unified 64kB continuous memory map
- Same instructions for data and peripherals
- Program and data in Flash or RAM with no restrictions
- Easy to understand with no paging
- Designed for modern programming techniques such as pointers and fast look-up tables

Memory Byte/Word Organization



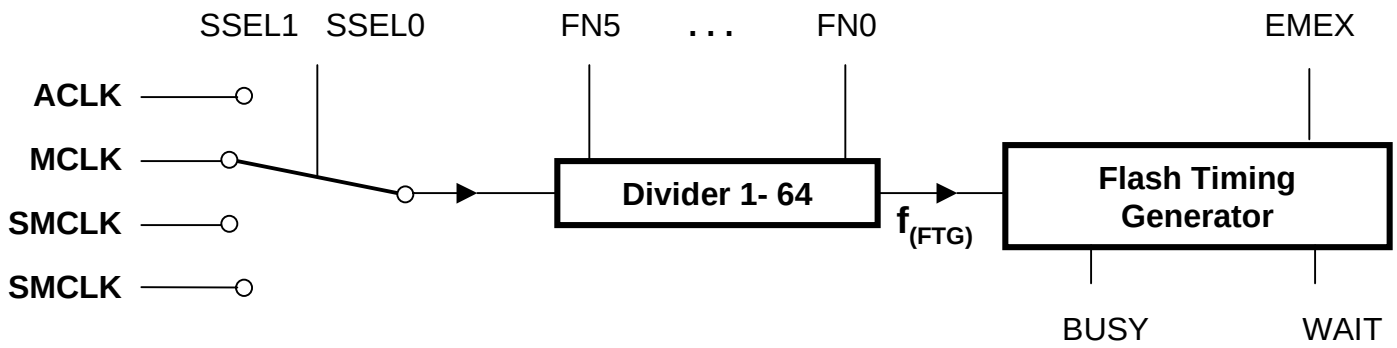


MSP430 Flash Memory Facts and Figures



- Main: (x) 512B seg.
  Info: (2) 128B seg.
- 1.8 - 3.6v operation
  2.7 - 3.6v programming
- Program and data in main or info with no restrictions
- Access password protected
- Program a bit, byte or word with JTAG, BSL or ISP
- 100k erase/program cycles typical
- 100-year data retention typical
- Less than 2s for 60kB programming time possible

Ultra-low Power Flash Controller



	15							8	0
FCTL1	PASSWORD	SEGWRT	WRT	- - -	- - -	- - -	MEras	Erase	- - -
FCTL2	PASSWORD	SSEL1	SSEL0	FN5	FN4	FN3	FN2	FN1	FN0
FCTL3	PASSWORD	- - -	- - -	EMEX	Lock	WAIT	ACCVIFG	KEYV	BUSY

Single Word or Segment Write

Single Segment or Mass Erase

EMergency **Ex**it Stops the active operation immediately and shuts down all internal parts of the flash controller

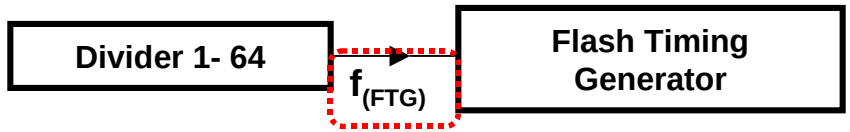
Program and erase disabled

Segment-write flag

Violation flags

Flag shows if Flash module is assessable.

Ultra-low Power Flash Programming Time



Flash timing generator frequency, $f_{(FTG)}$		MSP430F11x1	257	476	kHz

$I_{(DD-PGM)}$	Current during program cycle (see Note 26)	V_{CC} = 2.7 V/3.6 V, MSP430F11x1		3 5	mA
$I_{(DD-ERASE)}$	Current during erase cycle (see Note 26)	V_{CC} = 2.7 V/3.6 V, MSP430F11x1		3 5	mA
$t_{(retention)}$	Write/erase cycles	MSP430F11x1	10^4	10^5	
	Data retention T_J = 25°C	MSP430F11x1	100		Year

NOTES: 23. The power source to blow the fuse is applied to TEST pin.

24. Once the JTAG fuse is blown, no further access to the MSP430 JTAG/test feature is possible. The JTAG block is switched to bypass mode.

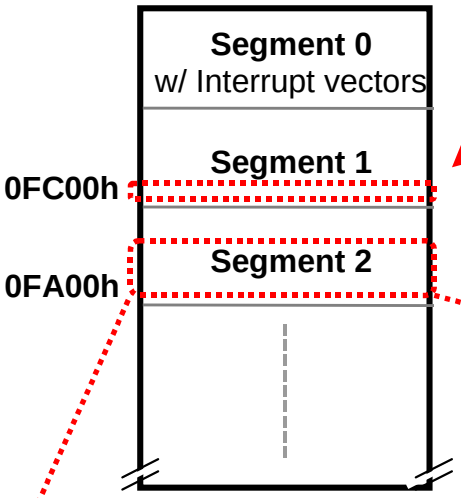
25. $f_{(TCK)}$ may be restricted to meet the timing requirements of the module selected.

26. Duration of the program/erase cycle is determined by $f_{(FTG)}$ applied to the flash timing controller. It can be calculated as follows:

$t_{(word\ write)} = 35 \times 1/f_{(FTG)}$
$t_{(block\ write,\ byte\ 0)} = 30 \times 1/f_{(FTG)}$
$t_{(block\ write,\ byte\ 1-63)} = 20 \times 1/f_{(FTG)}$
$t_{(mass\ erase)} = 5297 \times 1/f_{(FTG)}$
$t_{(page\ erase)} = 4819 \times 1/f_{(FTG)}$

How long does it take to program 1 byte or word? = 35/476000 = 73us

Ultra-low Power Flash In-System Programming

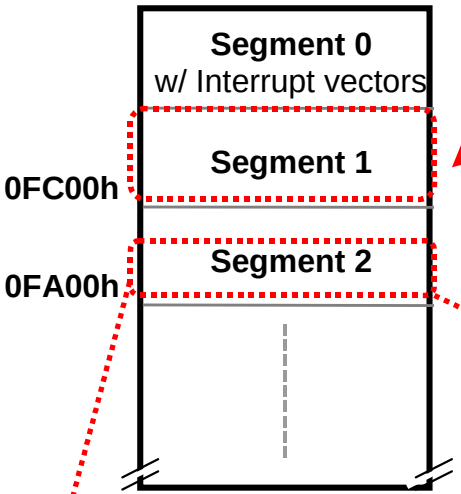


Example: Program 01234h @0FC00h

- Main program writes directly to Flash
- Password protection
- PC is held while BUSY flag is set - 35/f(FTG)

```
*address =   &0xFC00
// Flash_write
          FCTL3 = FWKEY;                  // Unlock the Flash
          FCTL1 = FWKEY | WRT;            // Enable Flash write
          *address = 0x1234;              // Write data to the Flash
          FCTL3 = FWKEY | LOCK;           // Lock the Flash

// Flash_Erase
          FCTL3 = FWKEY;                  // Unlock the Flash
          FCTL1 = FWKEY | ERASE;          // Enable Flash write
          *address = 0;                   // Dummy write to the Flash segment
          FCTL3 = FWKEY | LOCK;           // Lock the Flash
```

Ultra-low Power Flash In-System Erase



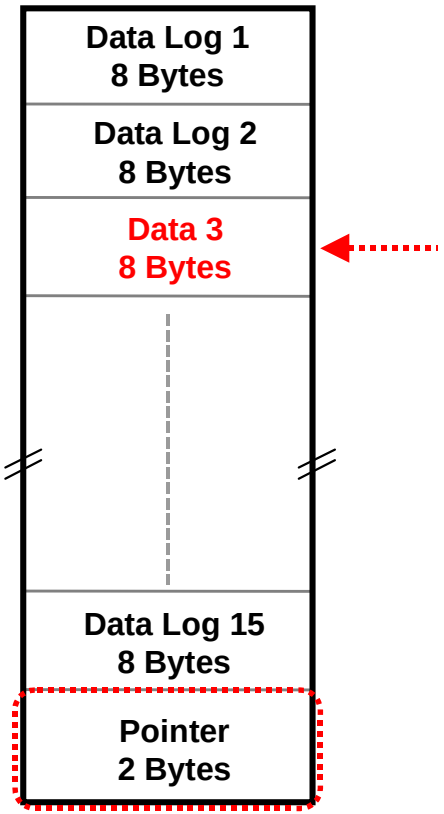
Example: Erase Segment 1

- Main program writes directly to Flash
- Password protection
- PC is held while BUSY flag is set - 4819/f(FTG)

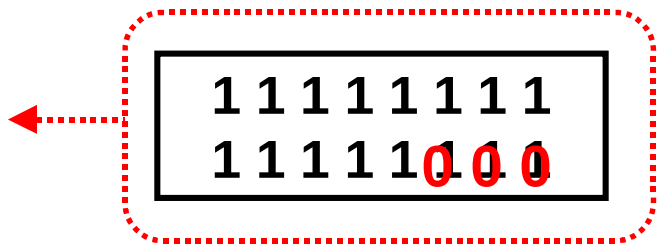
```
*address =   &0xFC00
// Flash_write
        FCTL3 = FWKEY;                  // Unlock the Flash
        FCTL1 = FWKEY | WRT;            // Enable Flash write
        *address = data;                // Write data to the Flash
        FCTL3 = FWKEY | LOCK;           // Lock the Flash
```

```
// Flash_Erase
        FCTL3 = FWKEY;                  // Unlock the Flash
        FCTL1 = FWKEY | ERASE;          // Enable Flash write
        *address = 0;                   // Dummy write to the Flash segment
        FCTL3 = FWKEY | LOCK;           // Lock the Flash
```

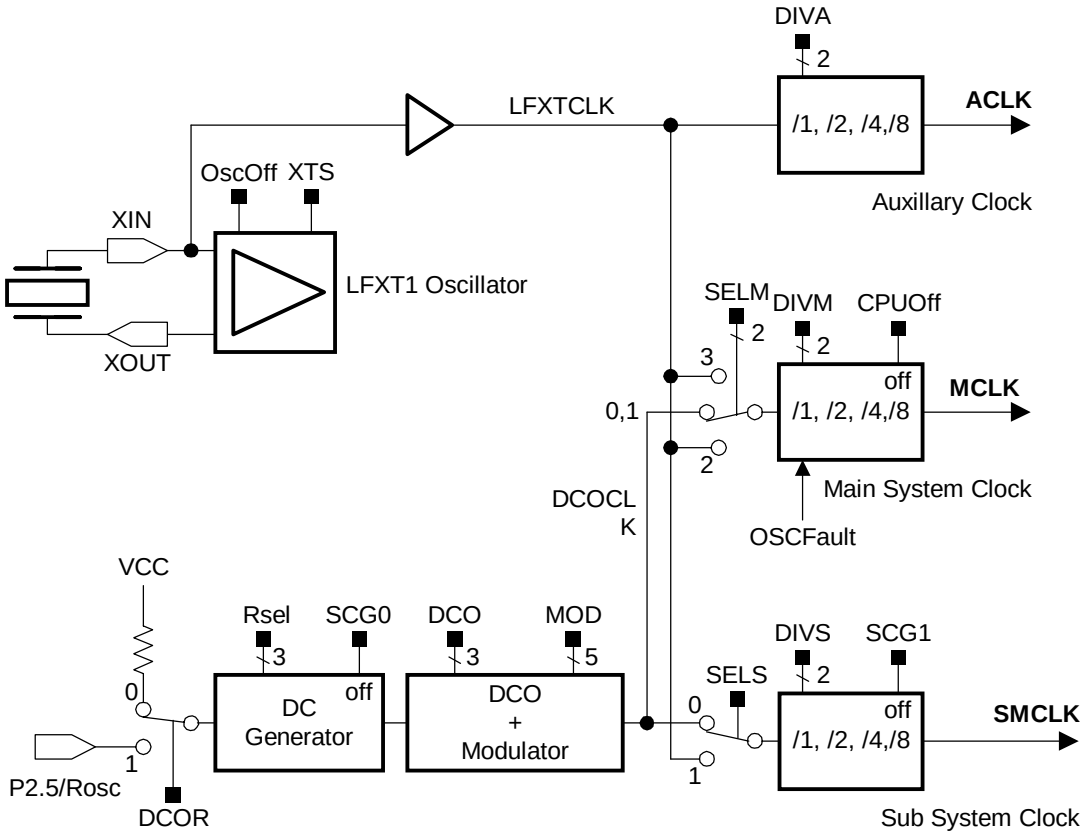
Ultra-low Power Virtual EEPROM Implementation



- In this example 8 bytes of data per logging is assumed - 15 data log points = 120 bytes
- One 2 byte pointer is used
- Tracking is done by clearing each bit starting from the LSB for every data log point.
- This approach extends the life of the Flash by reducing the number of erase cycles
- Any Flash segment can be used
- **Write/erase cycles = erase cycles**



MSP430x11x/12x Basic Clock



BCSCTL2 058h

SELM1	SELM0	DIVM1	DIVM0	SELS	DIVS1	DIVS0	DCOR
rw-1	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

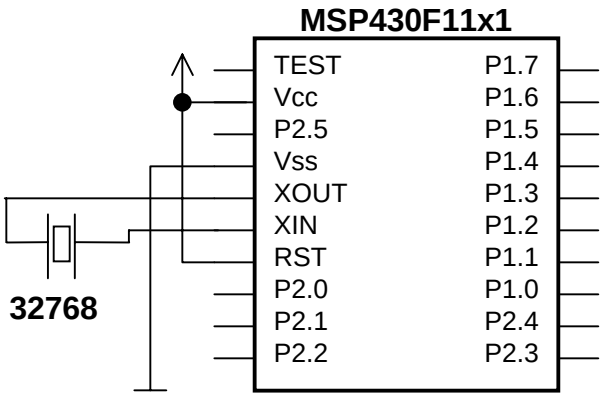
BCSCTL1 057h

XT2OFF	XTS	DIVA1	DIVA0	XT5V	RSEL2	RSEL1	RSEL0
rw-(1)	rw-(0)	rw-(0)	rw-(0)	rw-0	rw-1	rw-0	rw-0

DCOCTL 056h

DCO2	DCO1	DC01	MOD4	MOD3	MOD2	MOD1	MOD0
rw-0	rw-1	rw-1	rw-0	rw-0	rw-0	rw-0	rw-0

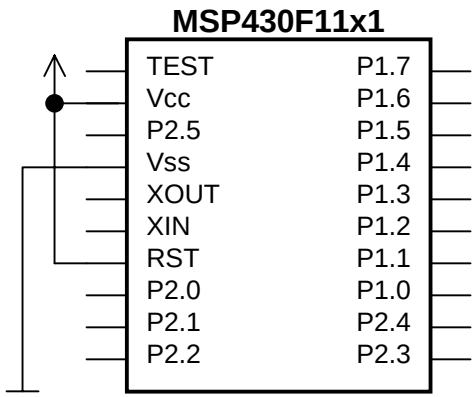
MSP430x11x/12x Basic Clock Set-up



ACLK = 32768 (or divisor)

MCLK ~ 1MHz DCOCLK (or divisor)

SMCLK ~ 1MHz DCOCLK (or divisor)



ACLK = 0

MCLK ~ 1MHz DCOCLK (or divisor)

SMCLK ~ 1MHz DCOCLK (or divisor)

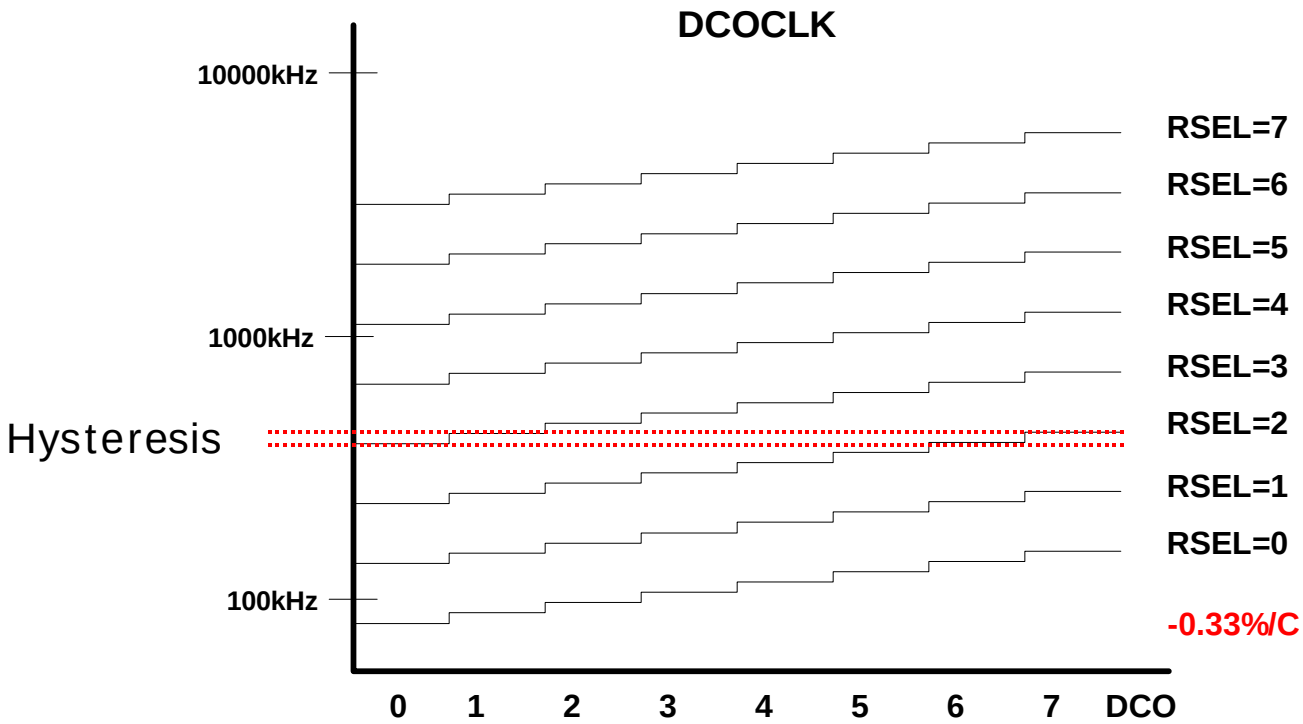
ACLK defaults to LFXT 32kHz-watch crystal. No external components required for LFXT. MCLK and SMCLK default to DCOCLK

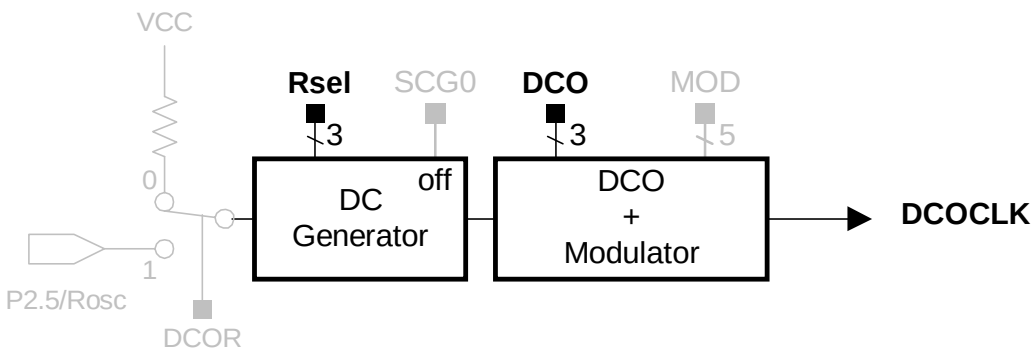
MSP430x1xx DCO Parameters

DCO

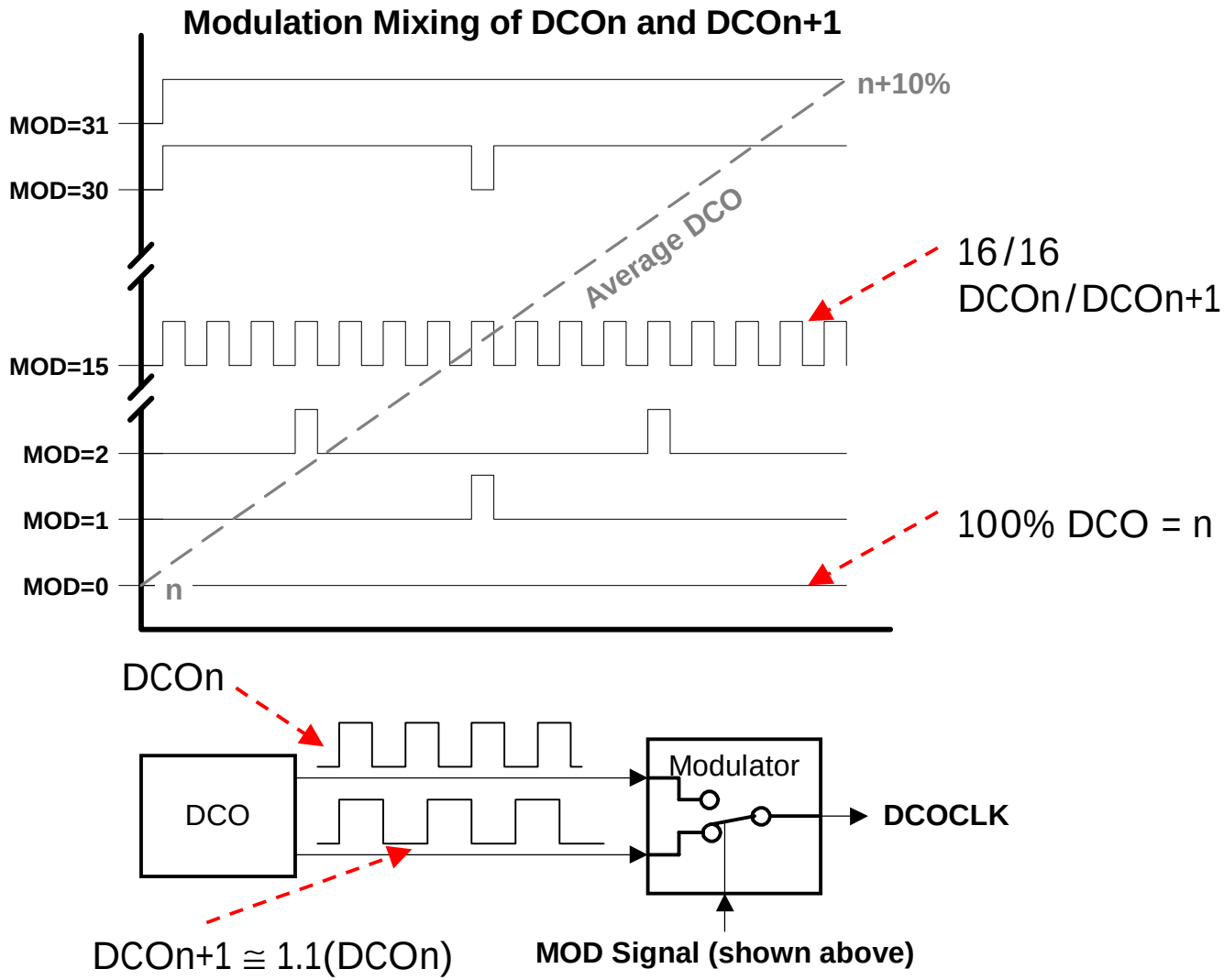
PARAMETER	TEST CONDITIONS		MIN	TYP	MAX	UNIT
$f_{(DCO03)}$	R_{sel} = 0, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	0.08	0.12	0.15	MHz
		V_{CC} = 3 V	0.08	0.13	0.16	
$f_{(DCO13)}$	R_{sel} = 1, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	0.14	0.19	0.23	MHz
		V_{CC} = 3 V	0.14	0.18	0.22	
$f_{(DCO23)}$	R_{sel} = 2, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	0.22	0.30	0.36	MHz
		V_{CC} = 3 V	0.22	0.28	0.34	
$f_{(DCO33)}$	R_{sel} = 3, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	0.37	0.49	0.59	MHz
		V_{CC} = 3 V	0.37	0.47	0.56	
$f_{(DCO43)}$	R_{sel} = 4, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	0.61	0.77	0.93	MHz
		V_{CC} = 3 V	0.61	0.75	0.9	
$f_{(DCO53)}$	R_{sel} = 5, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	1	1.2	1.5	MHz
		V_{CC} = 3 V	1	1.3	1.5	
$f_{(DCO63)}$	R_{sel} = 6, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	1.6	1.9	2.2	MHz
		V_{CC} = 3 V	1.69	2	2.29	
$f_{(DCO73)}$	R_{sel} = 7, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	2.4	2.9	3.4	MHz
		V_{CC} = 3 V	2.7	3.2	3.65	
$f_{(DCO77)}$	R_{sel} = 7, DCO = 7, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	4	4.5	4.9	MHz
		V_{CC} = 3 V	4.4	4.9	5.4	
$f_{(DCO47)}$	R_{sel} = 4, DCO = 7, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V/3 V	F_{DCO40} x1.7	F_{DCO40} x2.1	F_{DCO40} x2.5	MHz
$S_{(Rsel)}$	$S_R = f_{Rsel+1}/f_{Rsel}$	V_{CC} = 2.2 V/3 V	1.35	1.65	2	ratio
$S_{(DCO)}$	$S_{DCO} = f_{DCO+1}/f_{DCO}$	V_{CC} = 2.2 V/3 V	1.07	1.12	1.16	
D_t	Temperature drift, R_{sel} = 4, DCO = 3, MOD = 0 (see Note 21)	V_{CC} = 2.2 V	–0.31	–0.36	–0.40	%/°C
		V_{CC} = 3 V	–0.33	–0.38	–0.43	
D_V	Drift with V_{CC} variation, R_{sel} = 4, DCO = 3, MOD = 0 (see Note 21)	V_{CC} = 2.2 V/3 V	0	5	10	%/V

MSP430x1xx Programmable DCO



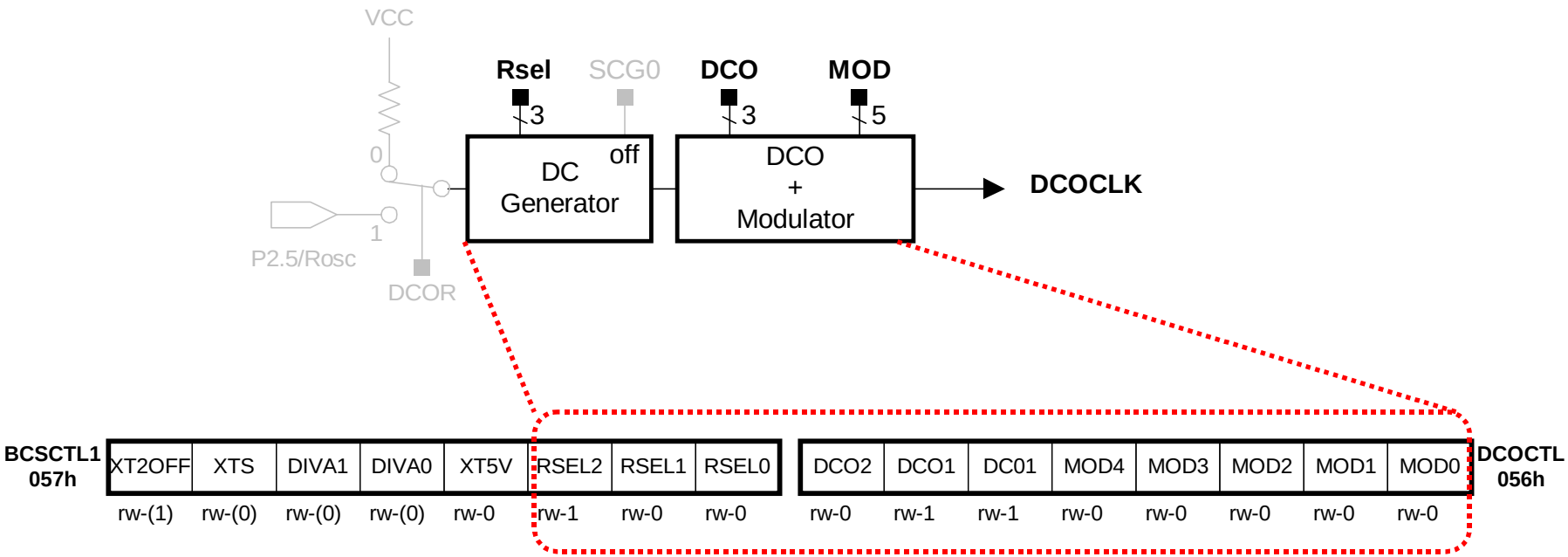


MSP430x1xx DCO Modulation



MSP430x1xx Basic Clock Frequency Setting

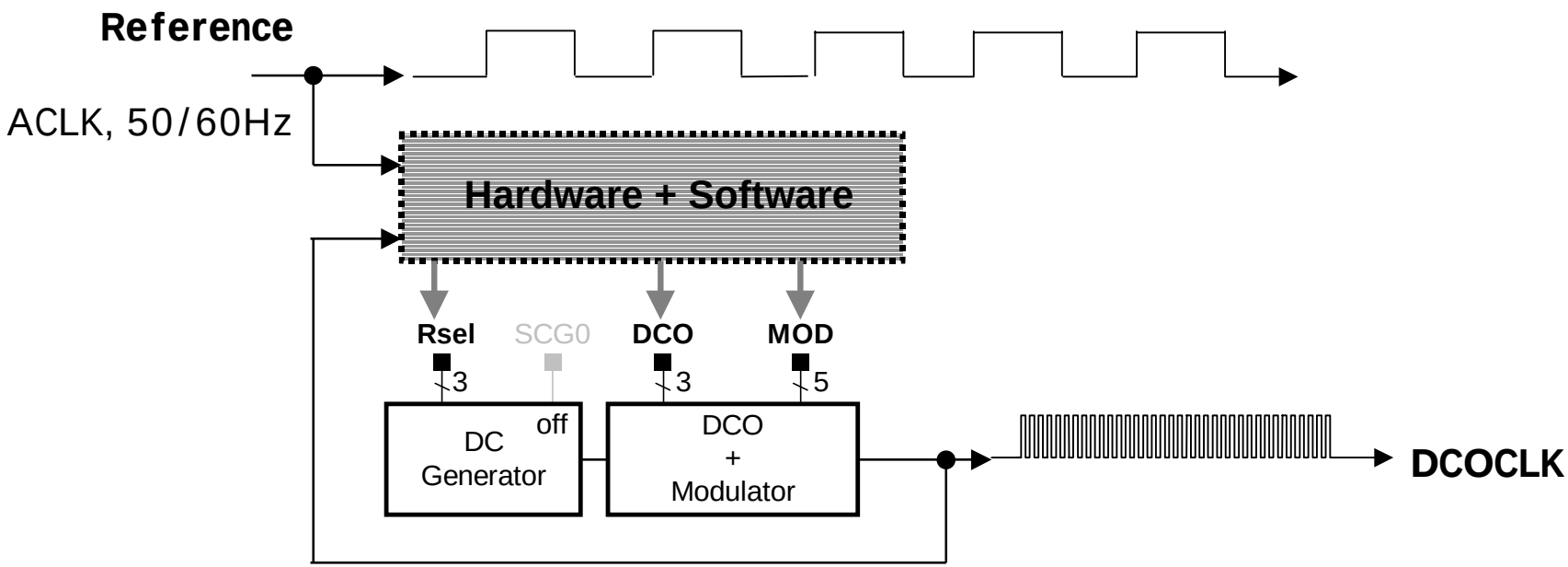
```
BCSCTL1 |= RSEL2 + RSEL1;      // DCOCLK~2MHz

DCOCTL++;                      // DCO Faster

DCOCTL--;                      // DCO Slower
```



Think of as an 11-bit register

MSP430x1xx Basic Clock SW FLL Concept

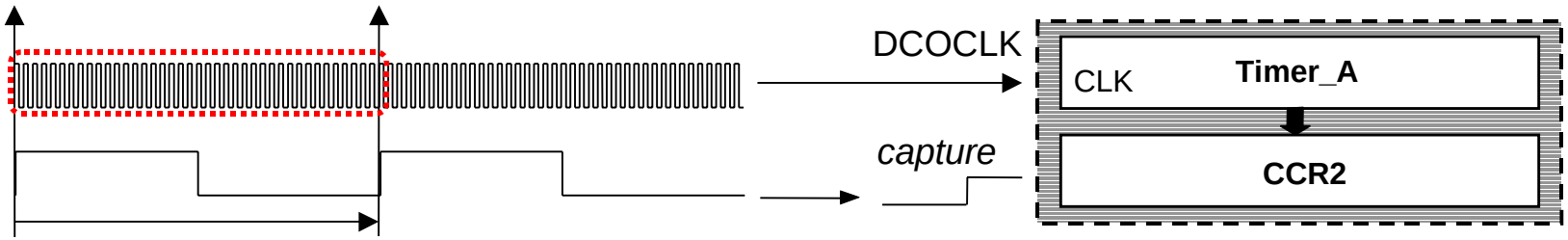
- Periodic loop adjusts DCOCLK by integrating over reference frequency
- DCOCLK is digitally programmable with software 100kHz ~ 5Mhz



MSP430x1xx Basic Clock SW FLL Example

Set DCOCLK= 1,000,000 with ACLK= 4096

- DCOCLK = 1,000,000 is clock source for timer_A
- ACLK = 4096 triggers 244us capture of DCOCLK on CCI2B
- CCI2B capture = 1,000,000/4096 ~244

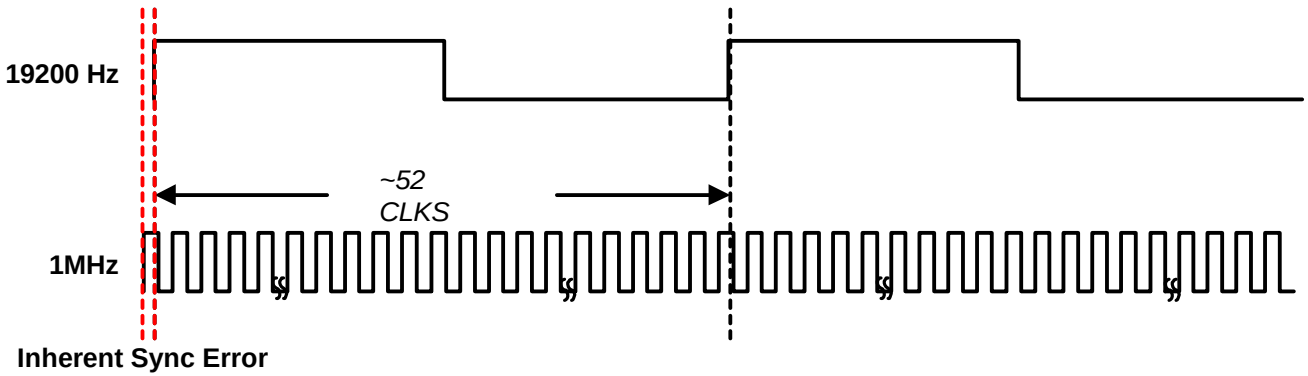



```
// Partial SW FLL Code
    break;                                  // If equal, leave

    else if (244 <  Compare  )              // DCO is too fast, slow it down
    DCOCTL--;

    else                                    // DCO is too fast, slow it down
    DCOCTL++;
```

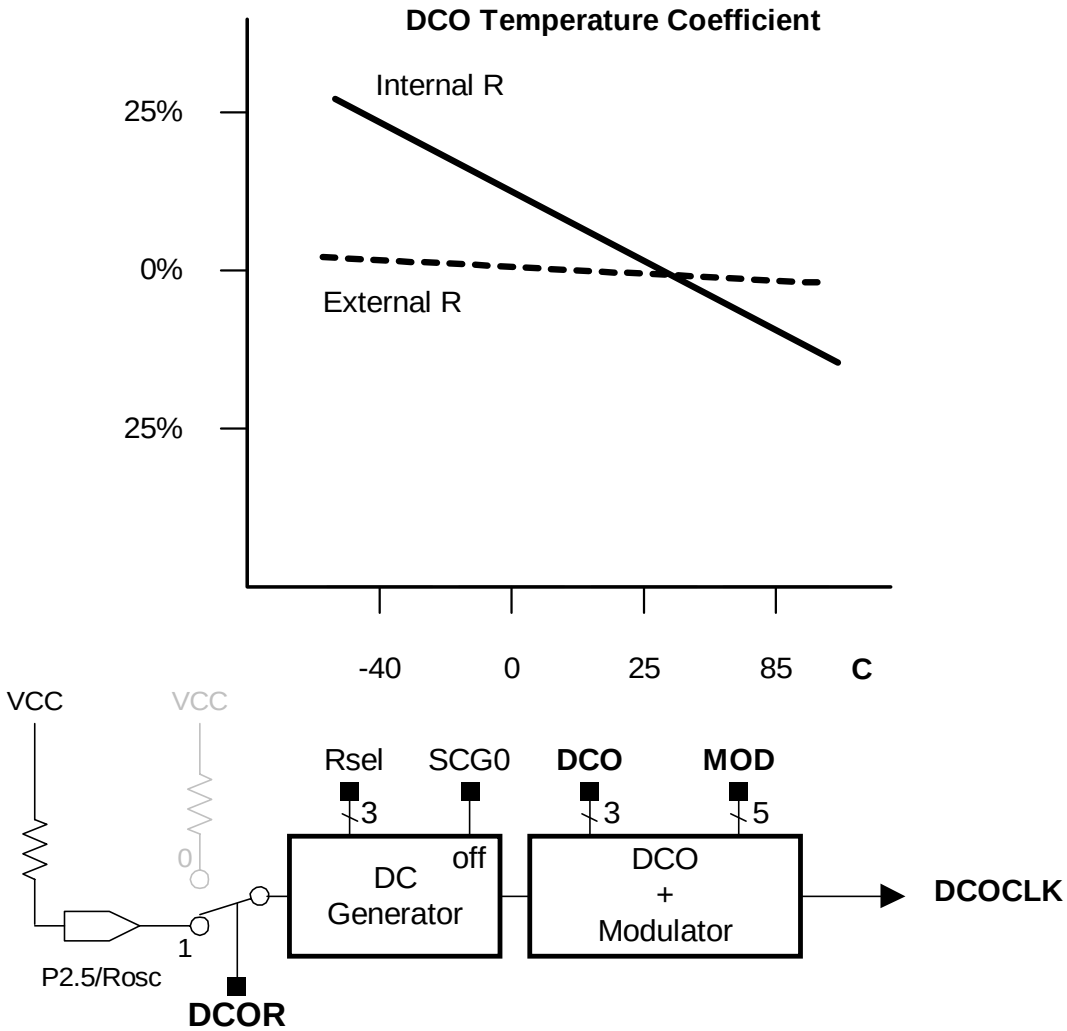
Does Modulation Jitter Effect An Application?

Example shows 19200 UART application using 1MHz baud rate clock

- Inherent error with perfect 1MHz is a 1us baud clock - or 1%
- Additional DCOCLK modulation error is only 50ns - or 0.1%

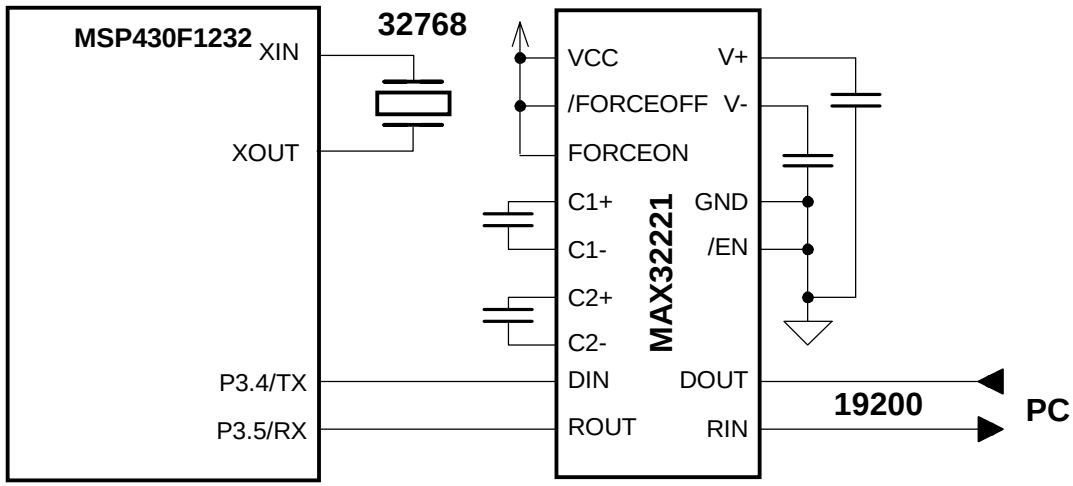


DCO Stability Improvement With External Rosc



BCS DCO Stability In A UART Application

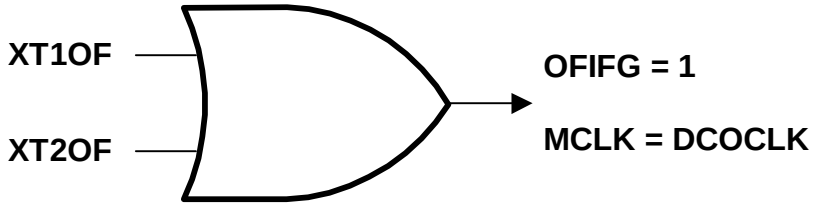
Group Demo

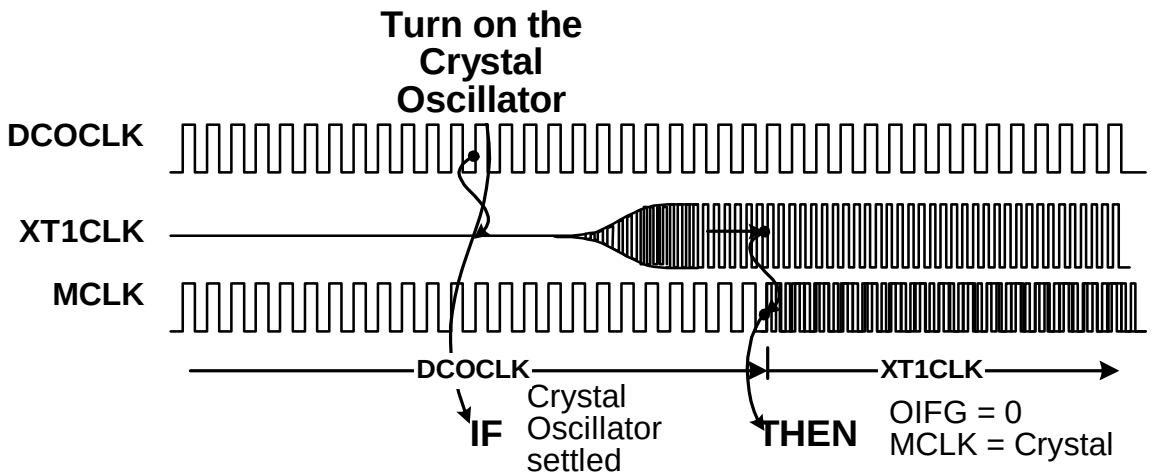


Software FLL sets the DCOCLK that is also used for UART baud rate generation.

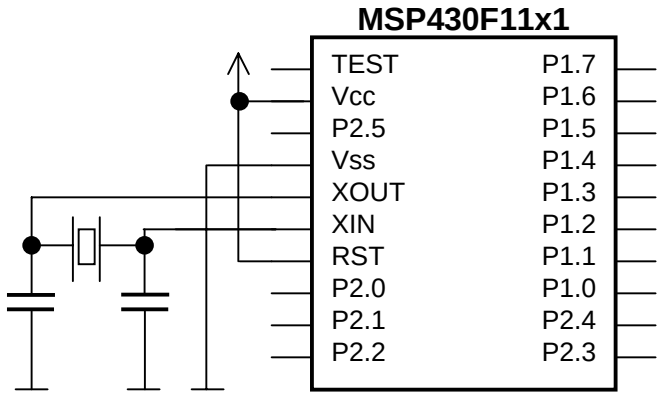
- Does the UART operate at 19200 using the DCO? What happens to the UART connection over temperature?
- What happens to the UART connection when the software FLL remains on always and the 430 is heated up /cooled down?

BCS Oscillator Fault





MSP430x11x/12x Basic Clock HF XTAL MCLK



ACLK = LFXT1 (or divisor)

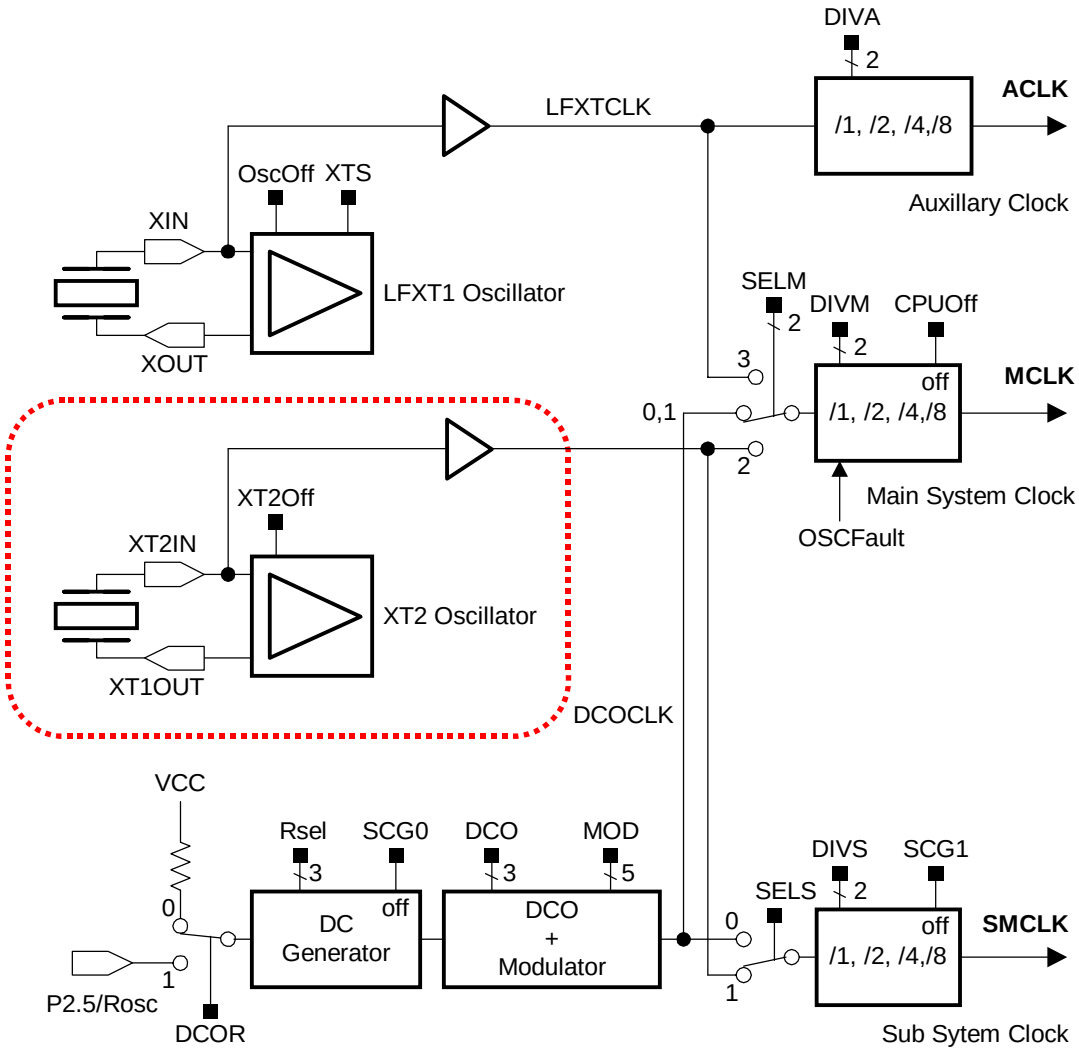
MCLK = DCOCLK or LFXT1

SMCLK = DCOCLK or LFXT1 (or divisor of)

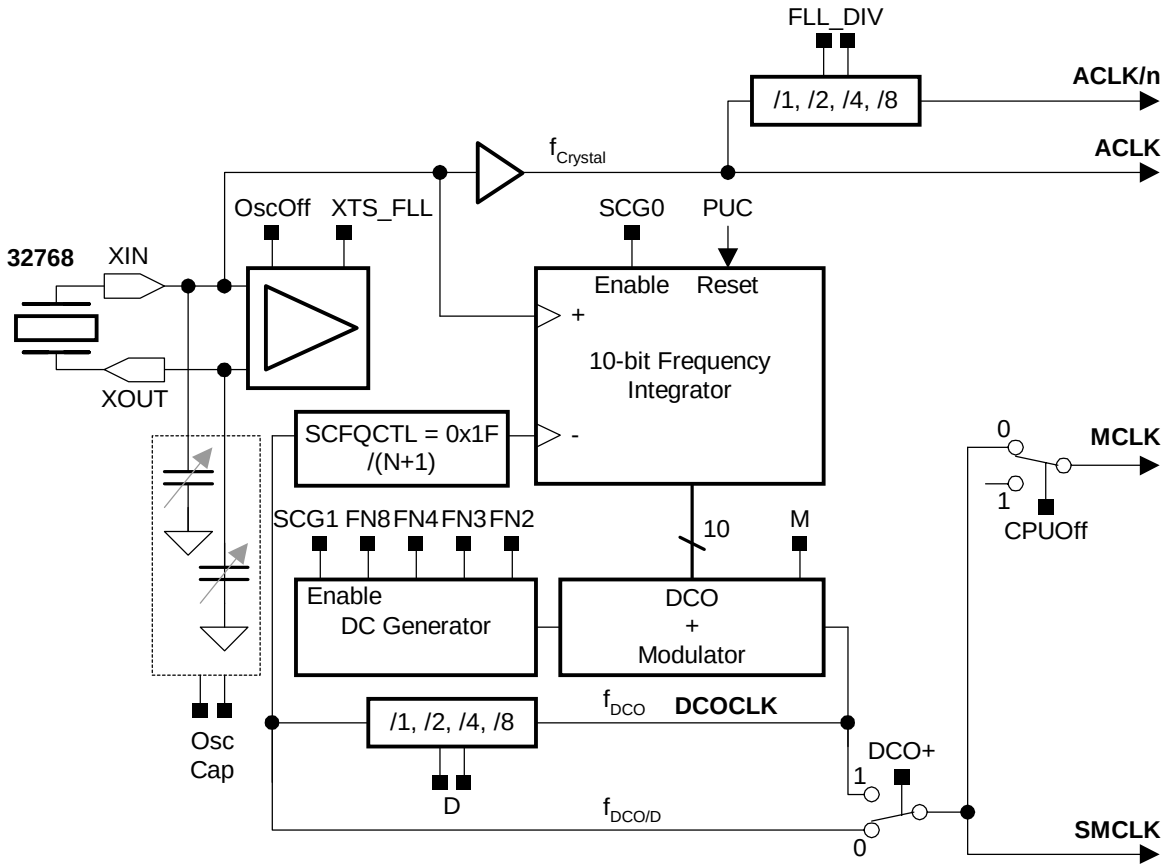
```
unsigned int i;
BCSCTL1 |= XTS;                          // ACLK = LFXT1 = HF XTAL

 do
 {
 IFG1 &= ~OFIFG;                         // Clear OSCFault flag
 for (i = 0xFF; i > 0; i--);             // Time for flag to set
 }
 while ((IFG1 & OFIFG) != 0);            // OSCFault flag still set?

 BCSCTL2 |= SELM1+SELM0;                 // MCLK = LFXT1 (safe)
```

External high frequency XTAL - capacitors required per crystal specification

MSP430x13x/14x/15x/16x Basic Clock

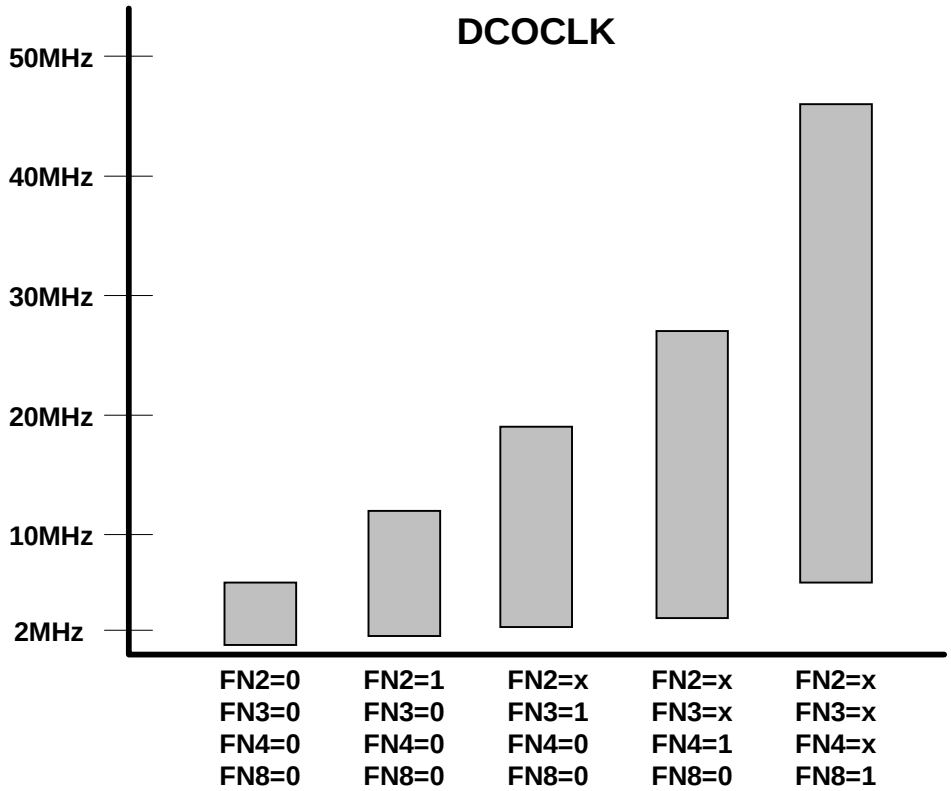


MSP430x41x/42x FLL+

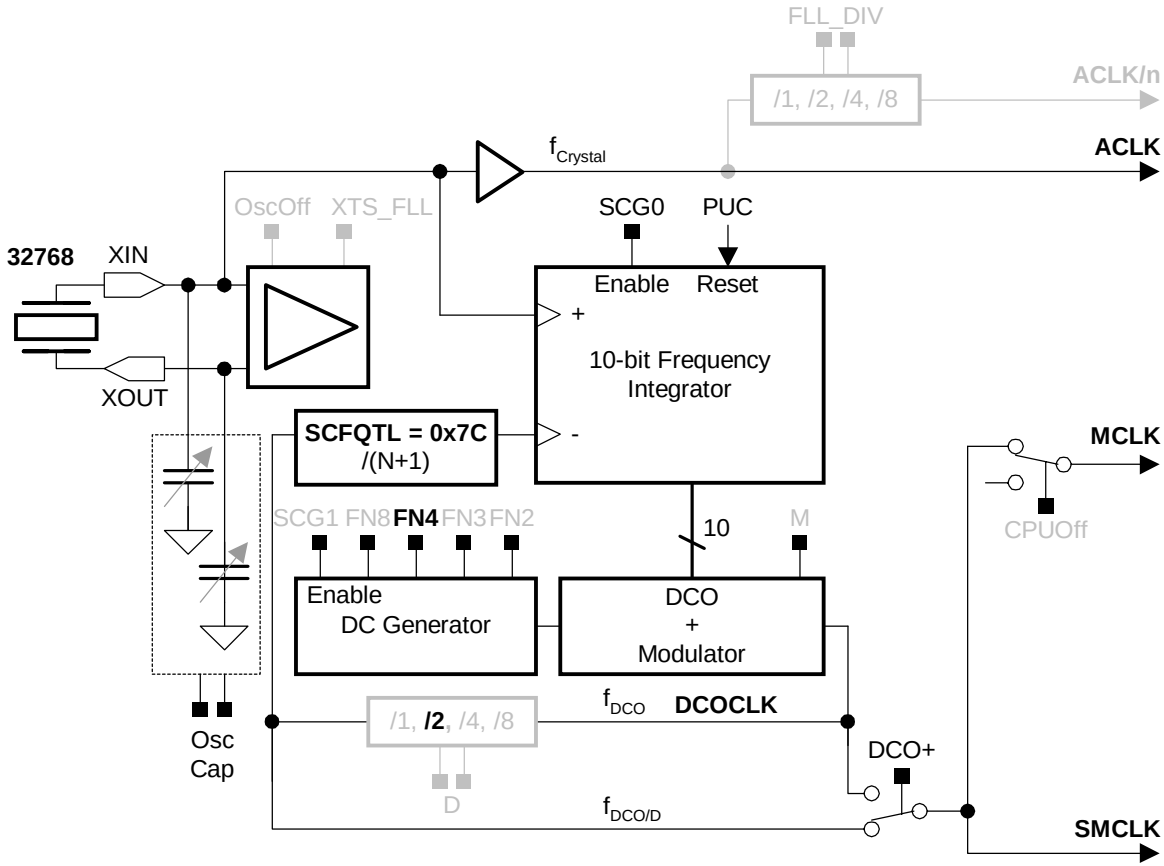


Caution: MSP430x4xx OSC Caps default =0pf
- make sure to configure in software

MSP430x4xx Programmable FLL+ and DCO

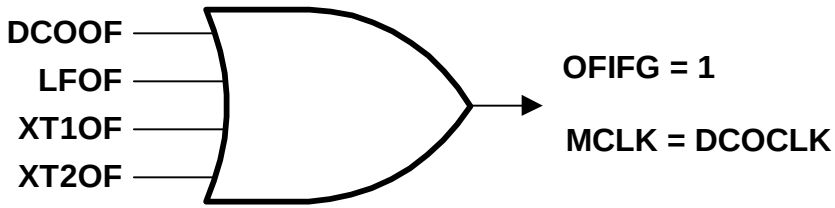


MSP430x4xx Example of Setting FLL+ to 4.096MHz



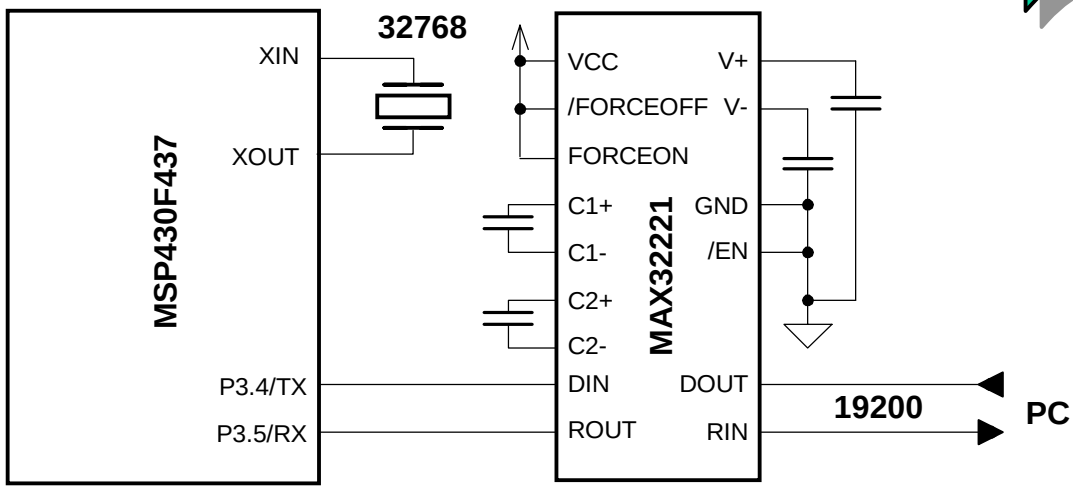

```
FLL_CTL0 | = DCOPLUS + XCAP18PF;        // Load Caps
SCFI0 |= FN_4;                          // 84MHz nominal
SCFQCTL = (125-1);                      // 125 x 32768 = 4.096 Mhz
```

FLL+ Oscillator Fault



FLL+ Stability In a UART Application

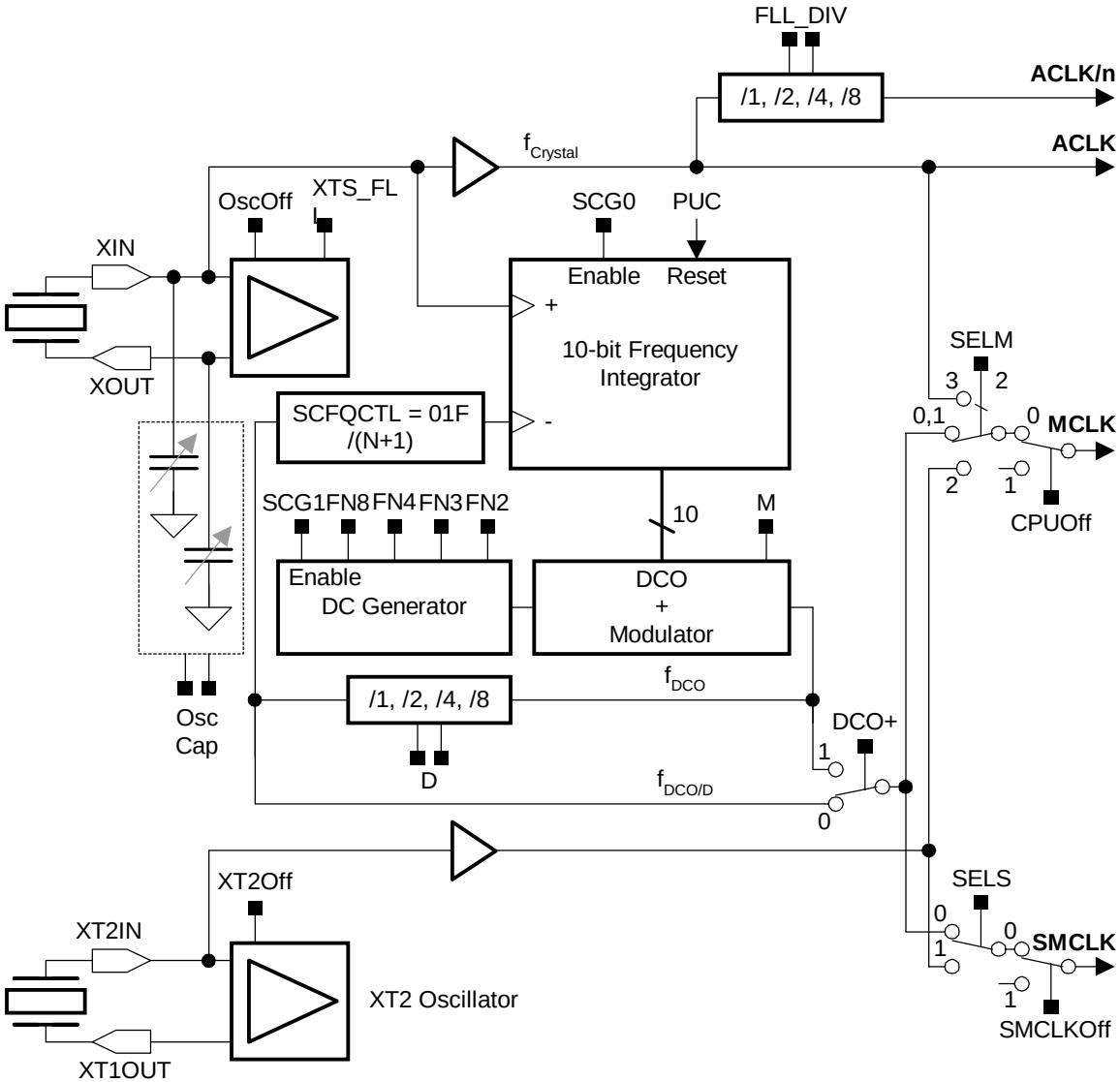
Instructor Demo



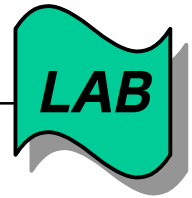
Hardware FLL automatically calibrates the DCOCLK with no CPU resources. The DCOCLK is used for UART baud rate generation.

- What happens to the UART connection over temperature?

MSP430x43x/44x FLL+



D437_3.c ADC12 Vref Cal. Theory



The MSP430 ADC12 specification states the following for VREF+ :

- VREF = 2.5 V tolerance is +/- 0.1 V
- VREF = 1.5 V tolerance is +/- 0.06V

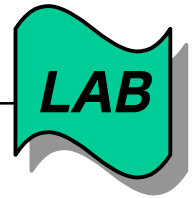
From the following formula,

Measured value in mV = (Counts / 4096) * VREF in mV

To achieve absolute accuracy in the “Measured value”, VREF has to be accurate. The in-system programmable (ISP) flash of the MSP430 simplifies the calibration of the VREF deviation by software as follows:

1. To begin calibration the ideal value for VREF is assumed and stored in flash.
2. The Measured value using ADC12 is then compared with an accurate millivoltmeter and VREF value is incremented or decremented by firmware when the user presses the corresponding push buttons on the D437 .
3. When both values match, cal mode is exited with the new VREF value programmed in flash INFO memory - firmware will now use the calibrated VREF.

D437_3.c ADC12 Vref Cal. Software Using ISP




```
// Global "variables" (in flash-INFO)
typedef unsigned int word;

#pragma memory=dataseg(INFO)
static word Refcal_flash;                    // ADC12 Ref calibration in Flash INFO memory
#pragma memory=default


void check_cal(void)
{ if (Refcal_flash == 0xffff)                // Check if Flash INFO Refcal erased?
  flash_write(&Refcal_flash, 1500);          // Write initial assumed value 1500mV
 }

void Ref_cal(void)
{ P2IE = 0;                                  // Disable Push button interrupts
  if ((P2IN&PB_TIME)==0)                     // If Time button pressed decrement cal
  Refcal_ram--;
  if ((P2IN&PB_TEMP)==0)                     // If Temp button pressed increment cal
  Refcal_ram++;
  if ((P2IN&PB_VOLT)==0)                     // If Volt button pressed store cal in Flash
  { flash_erase(&Refcal_flash);              // Erase Flash INFO segment of Refcal
    flash_write(&Refcal_flash,Refcal_ram);   // Write cal data to Refcal
    calmode |= PB_VOLT;                      // Clear cal mode
    P2IFG = 0;                               // Clear any pending push button int flags
    P2IE = (PB_TIME | PB_TEMP | PB_VOLT);    // Enable push button interrupts
   }
 }
```

Notes

Agenda - Dallas,TX - November 2002

Optimal Low Power Mode Utilization

Low Power Modes:

- Power source selection
- Power managing external components
- Importance of reducing system overhead
- Clock system design considerations
- Switching between low power modes

Optimizing Low Power Design with Peripheral Features:

- Using ADC12 autoscan to reduce data handling overhead
- Using ADC10 DTC to reduce data handling overhead

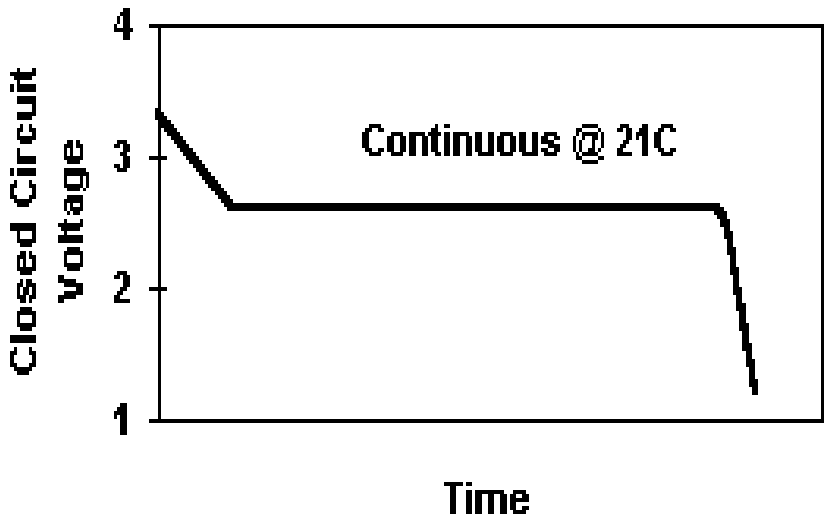
Interrupt Architecture:

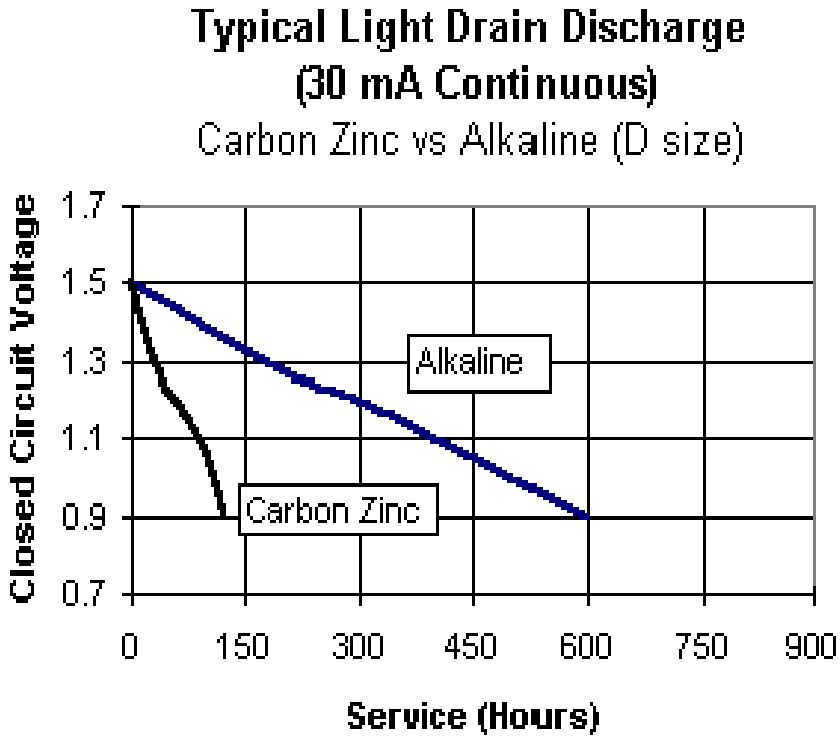
- Interrupt vectors and priorities
- Enabling external and internal interrupt sources
- Adding interrupt support software in assembler and C
- Code solutions for managing single and multiple source interrupts

LAB Session 2

Lab: Reducing power consumption using LPM3
Lab: Demonstrate ADC12 autoscan sampling speed improvement

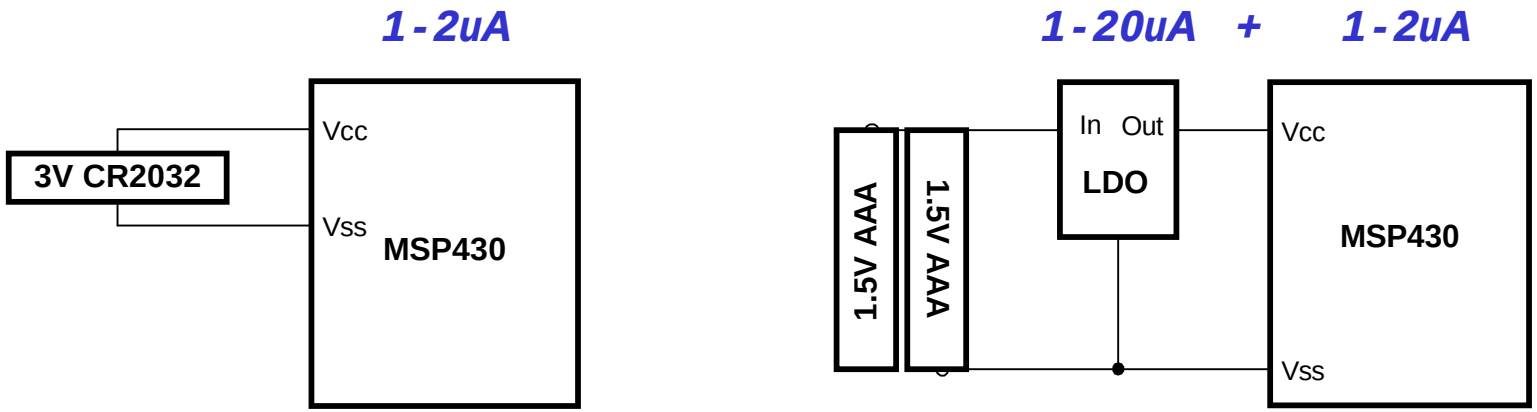
Battery Choice Can Eliminate Power Management





Flat discharge of lithium is ideal for direct-power

Direct-Supply Extends Battery Life



Assumed minimum VCC = 2.7V

~ 2uA average current

@2.7V CR2032 discharged 90%

10-year battery-life

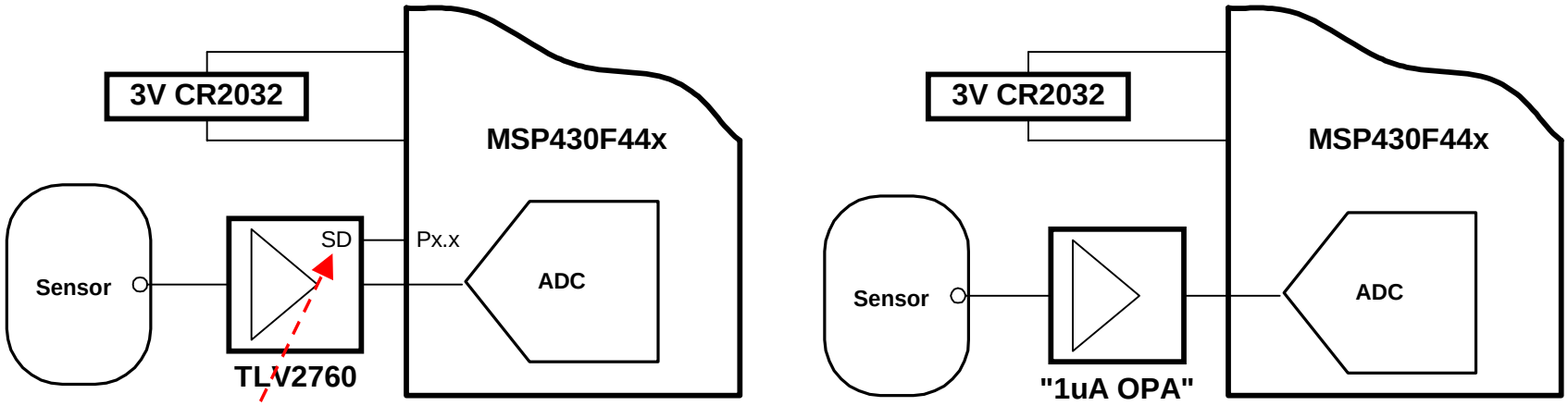
~ 5uA average current

@2.7V "2xAA" discharged 12%

2-year battery-life

Flat discharge of lithium battery allows deep discharge

Power Manage Peripherals



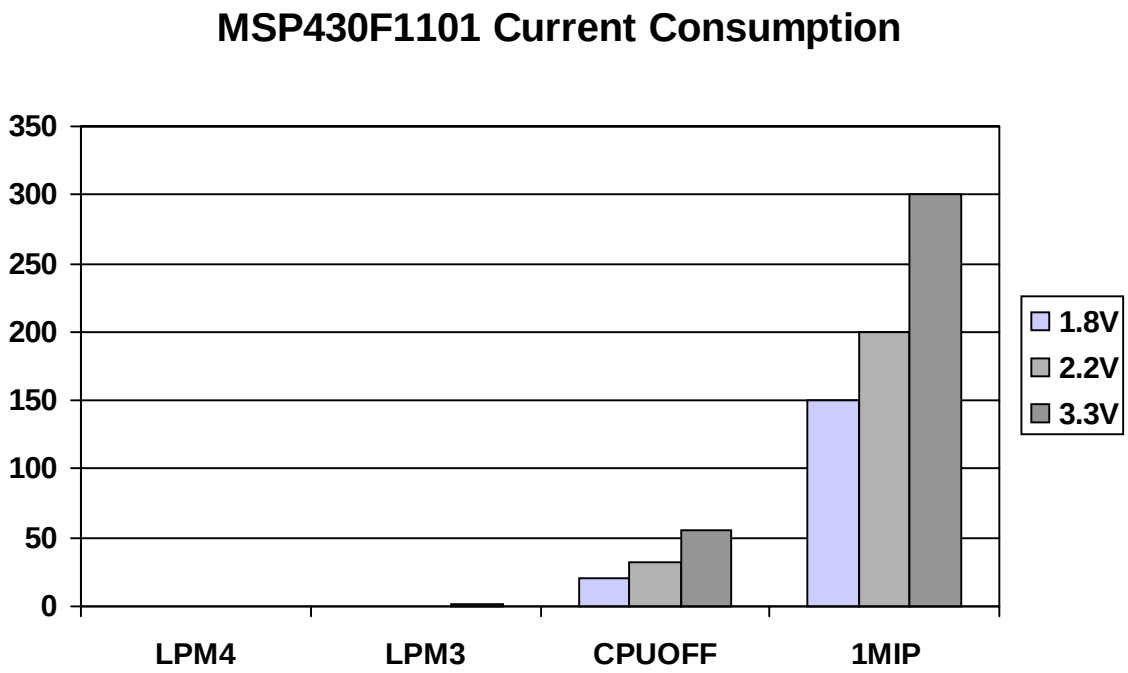
Instrument active 4 x 1-minute interval / day

OPA with shutdown
0.01uA = Shutdown current
20uA = Active current
0.06uA = Average current

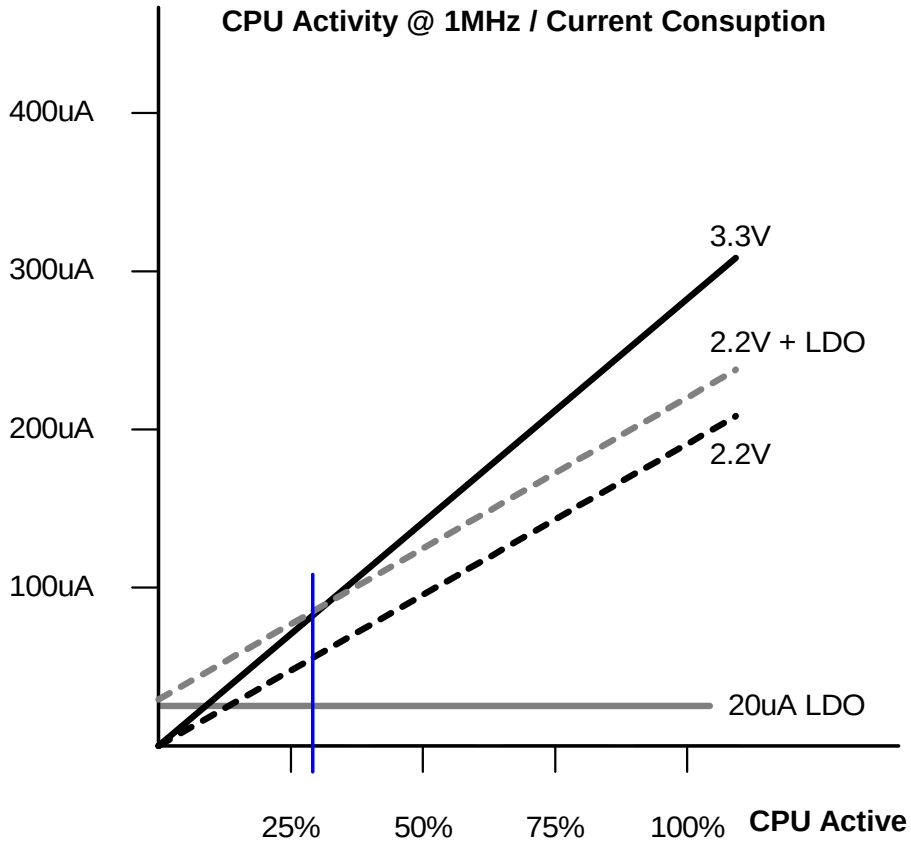
OPA without shutdown
1uA = Quiescent current
1uA = Active current
1uA = Average current

OPA with shutdown is 20x lower total power in portable measurement applications

Reducing Operating Voltage Reduces Power Consumption

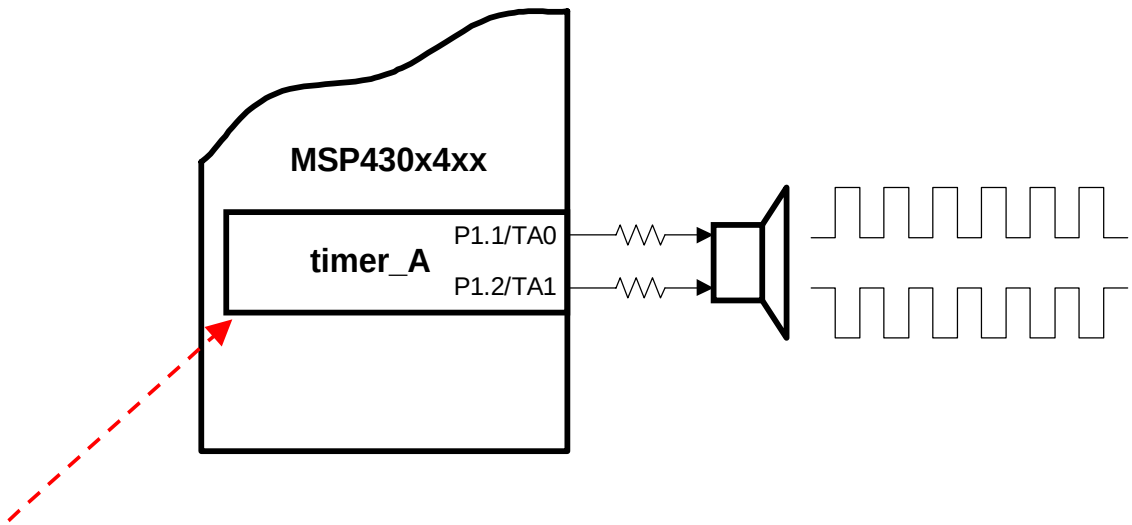


Activity Profile Directs Power Management Decisions



Applications that must have high CPU activity can benefit from reduced Vcc

Correct Use of Internal Peripherals

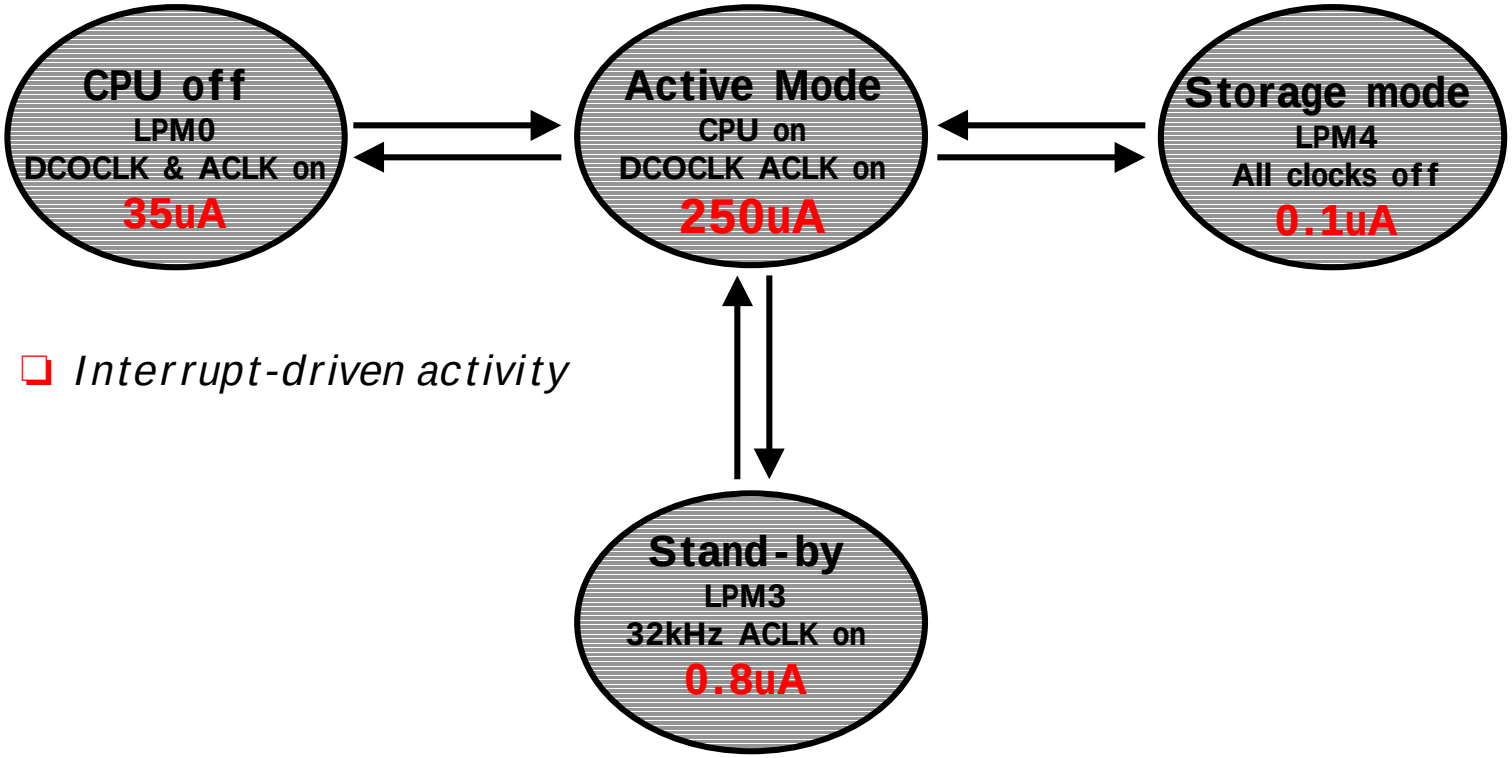


Buzzer is driven using timer_A PWM outputs requiring no CPU resources

```
;
;A bad way to toggle P1.1/2 - all software
Mainloop    xor.b   #003h,&P1OUT            ; Toggle
            mov.w   #165,R4                 ; Delay
L1          dec.w   R4                      ;
            jnz     L1                      ;
            jmp     Toggle                  ;
```

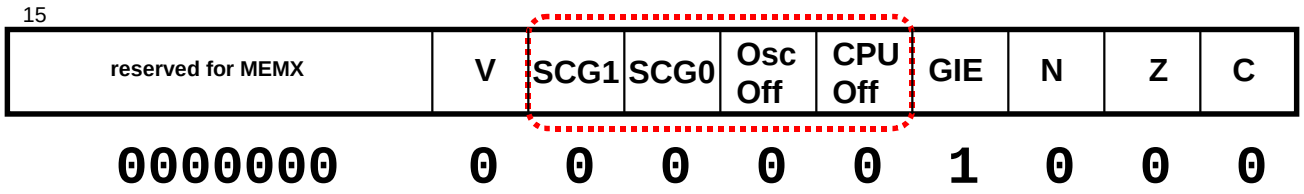
Ultra-low Power Event-Driven Activity

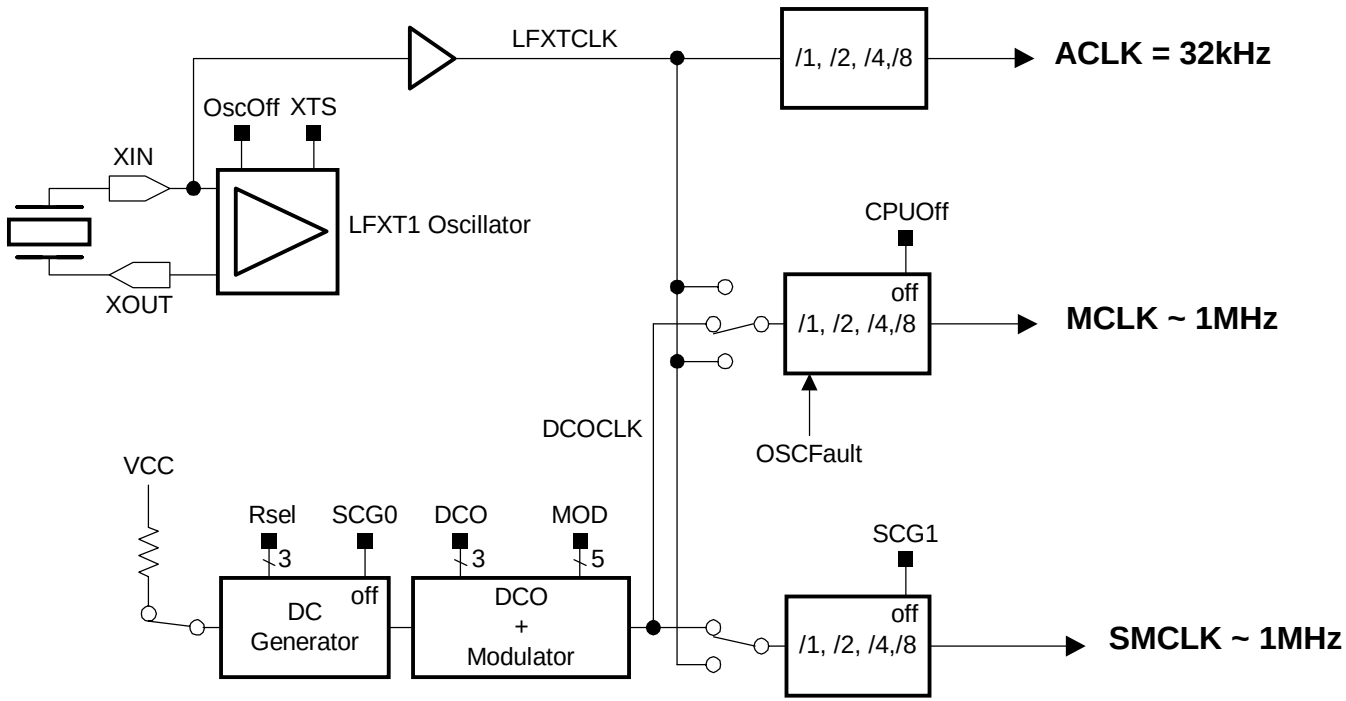
- *On-demand 16-bit burst performance*



- *Interrupt-driven activity*
- *Normal mode LPM3*

Ultra-low Power Active Mode

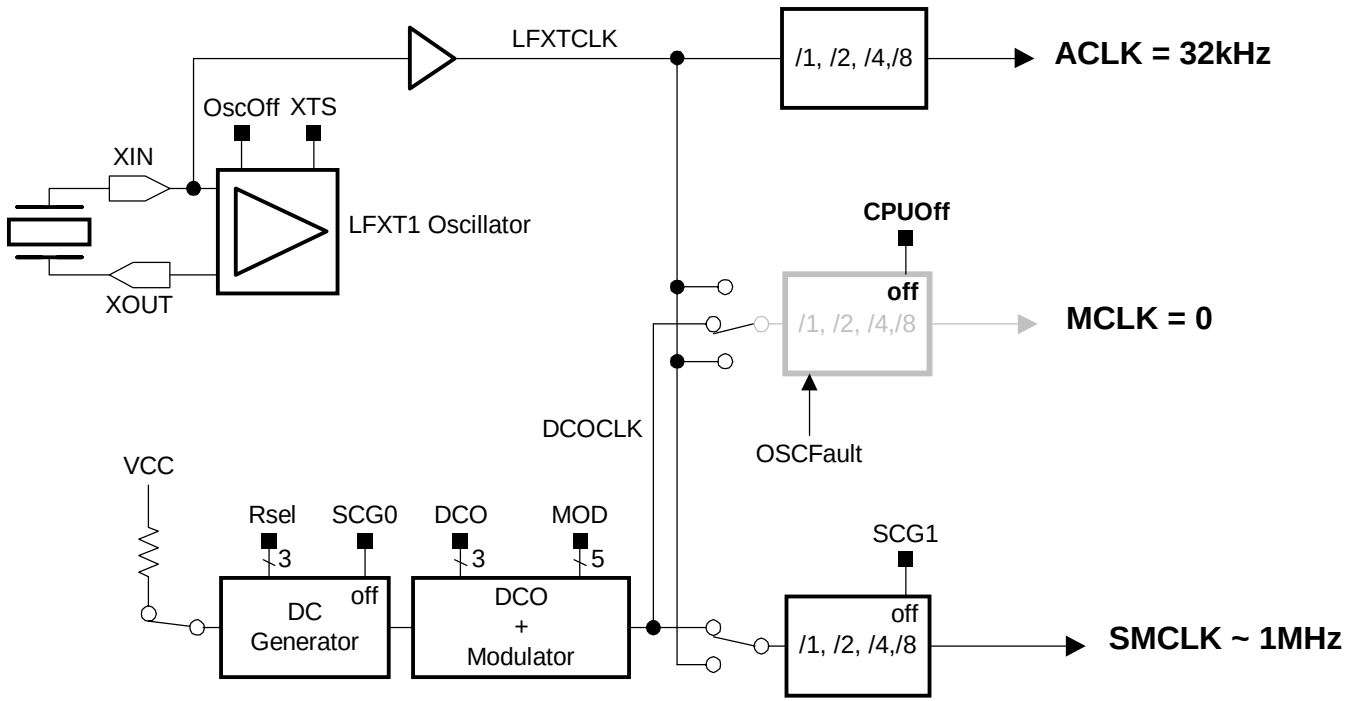




Optimal Solution Uses 32kHz XTAL and DCO ~ 250uA

Ultra-low Power Mode LPM0 - MCLK Off

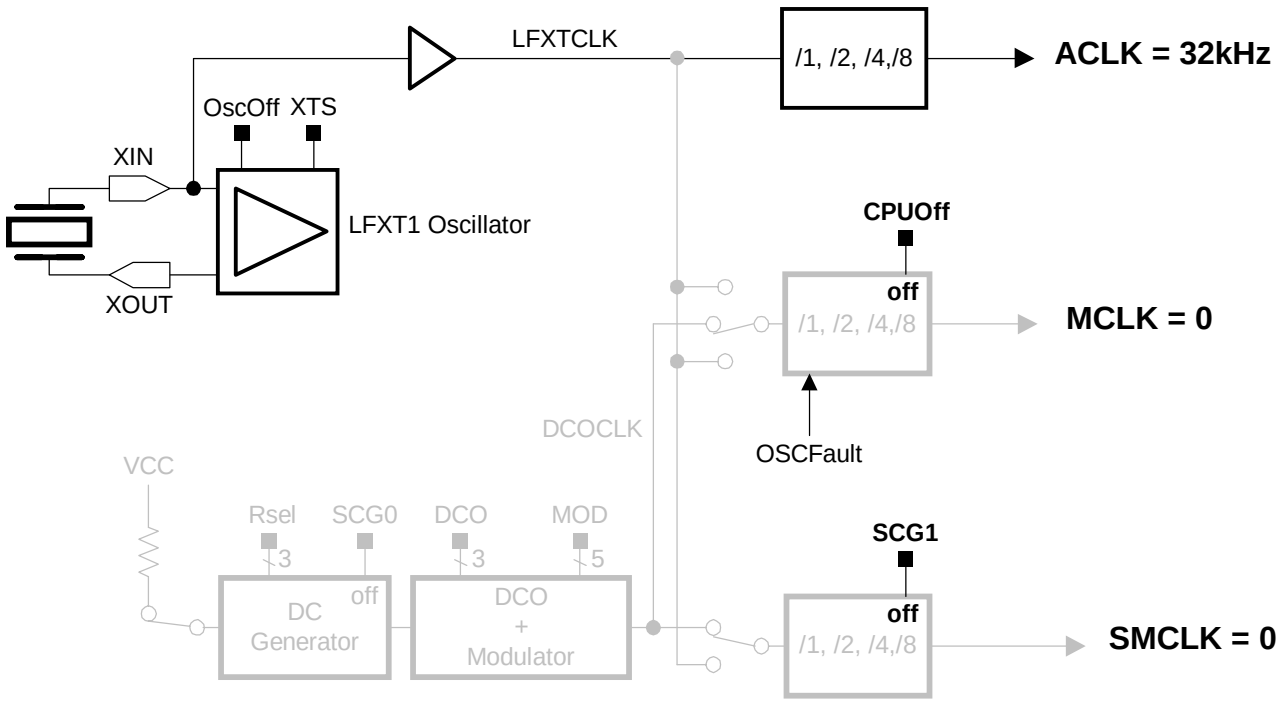
15										
reserved for MEMX	V	SCG1	SCG0	Osc Off	CPU Off	GIE	N	Z	C	SR Status Register
0000000	0	0	0	0	1	1	0	0	0	



Current Consumption ~ 30uA

Ultra-low Power Mode LPM3 - MCLK and DCOCLK Off

15										
reserved for MEMX	V	SCG1	SCG0	Osc Off	CPU Off	GIE	N	Z	C	Status Register
0000000	0	1	1	0	1	1	0	0	0	

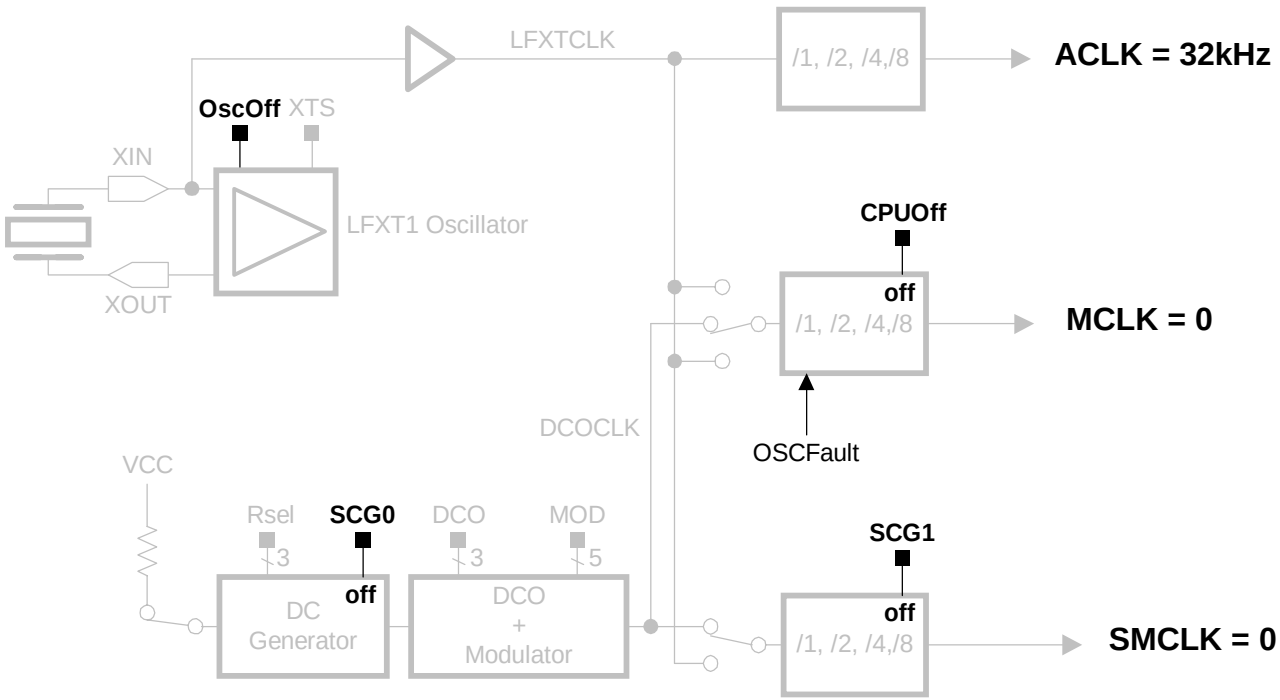


Stand-by RTC - Mode Current Consumption ~ 1uA

Ultra-low Power Mode LPM4 - All Clocks Off

reserved for MEMX	V	SCG1	SCG0	Osc Off	CPU Off	GIE	N	Z	C
0000000	0	1	1	1	1	1	0	0	0

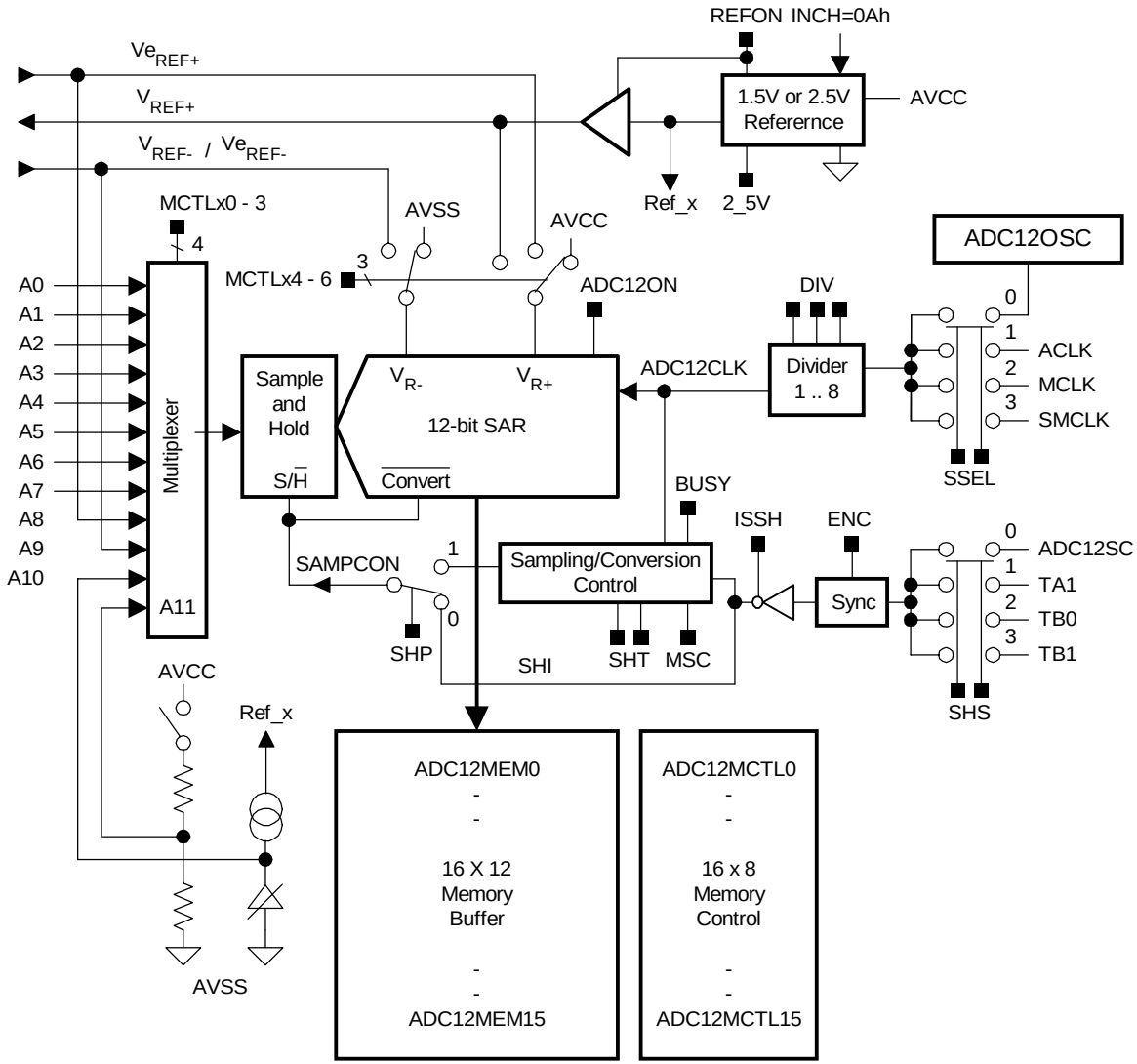
Status Register



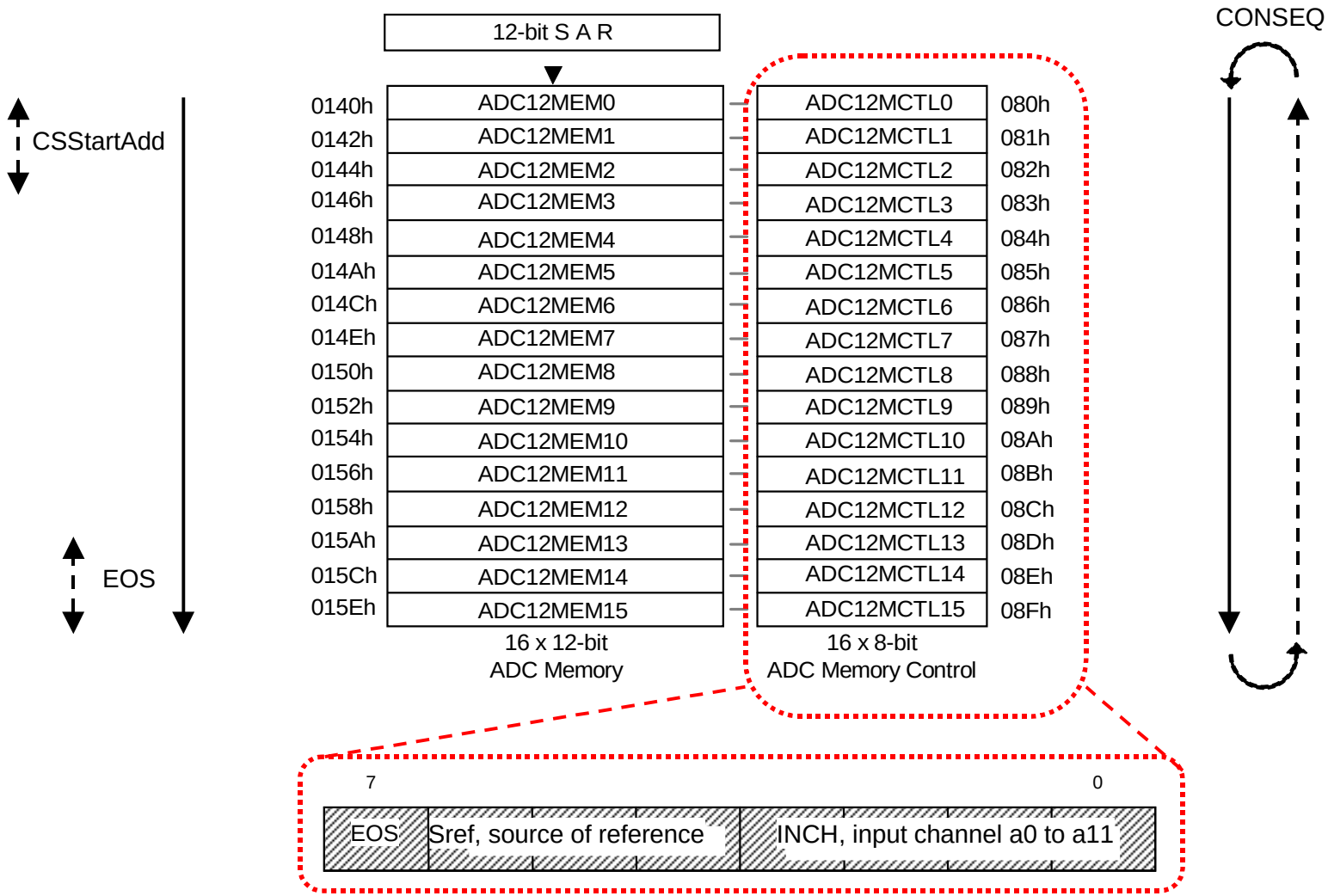
Storage Mode - Current Consumption ~ 0.1uA

ADC12

- 200ksps+
- Programmable sample & hold
- Band-gap Vref
- Temp. sensor
- Low-battery detect
- Auto-scan
- 16 word conversion buffer
- One interrupt vector generator with enable and and flags in module register



ADC12 Autoscan Feature



ADC12 Performance With ADC12OSC

Group Demo

Facts on ADC12

- Internal ADC12OSC = 4MHz typical
- Conversion time = 13/ADC12CLK, @4MHz Tconv = 3.25uS
- Sampling time = nX4/ADC12CLK, here n = 4 (min required)

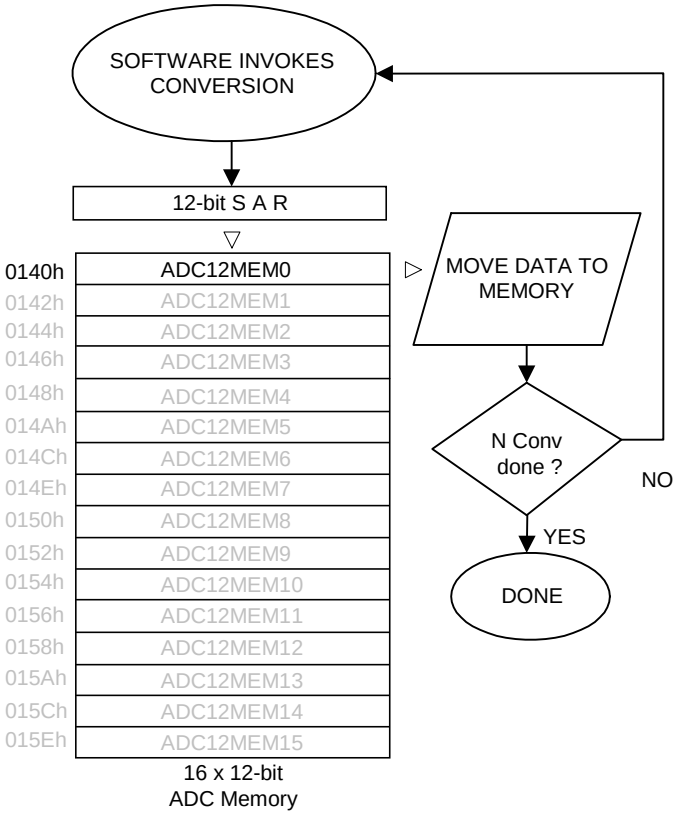
  Tsample = 4us for ADC12CLK @ 4MHz and n = 4
- Time to complete 1 conversion =Tconv + Tsample = 7.25uS
- Theoretical Ft(max) for ADC12CLK @ 4MHz, is 7.25us

  Fs(max) = 138 KSPS

Note: Increasing the ADC12CLK (6MHz max) frequency will increase the ADC12 throughput.

Software Scan of ADC12 Example D437_4.c

Group Demo



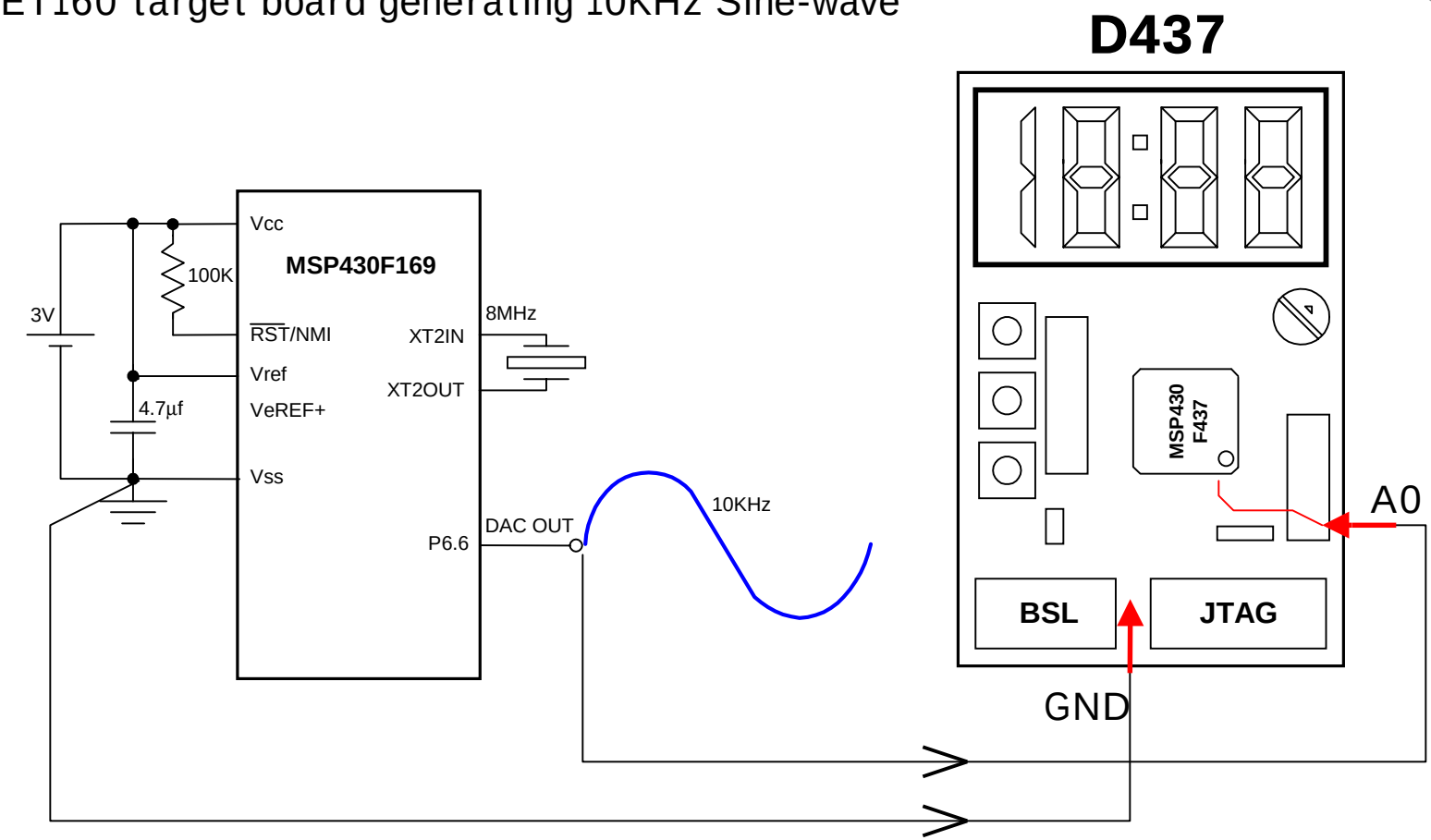
- Very inefficient
- Does not take advantage of Timer invoked sampling and Auto-scan hardware
- Sampling frequency controlled by software
- Loads CPU almost 100% of the time
- Fs(max) limited ~ 30ksps

```
for (i=0; i<16; i++)            // To collect 16x
  {
    ADC12CTL0 |= ADC12SC;       // Start conversion
    LPM0;                       // Enter LPM0
  }
  ADC12CTL0 &= ~ENC;            // Clear ENC bit

  interrupt[ADC_VECTOR] void ADC12ISR (void)
  {
  ADCData[i] = ADC12MEM0;       // Move results
  LPM0_EXIT;                    // Exit LPM0
  }
```

ADC12 Demo

Group Demo

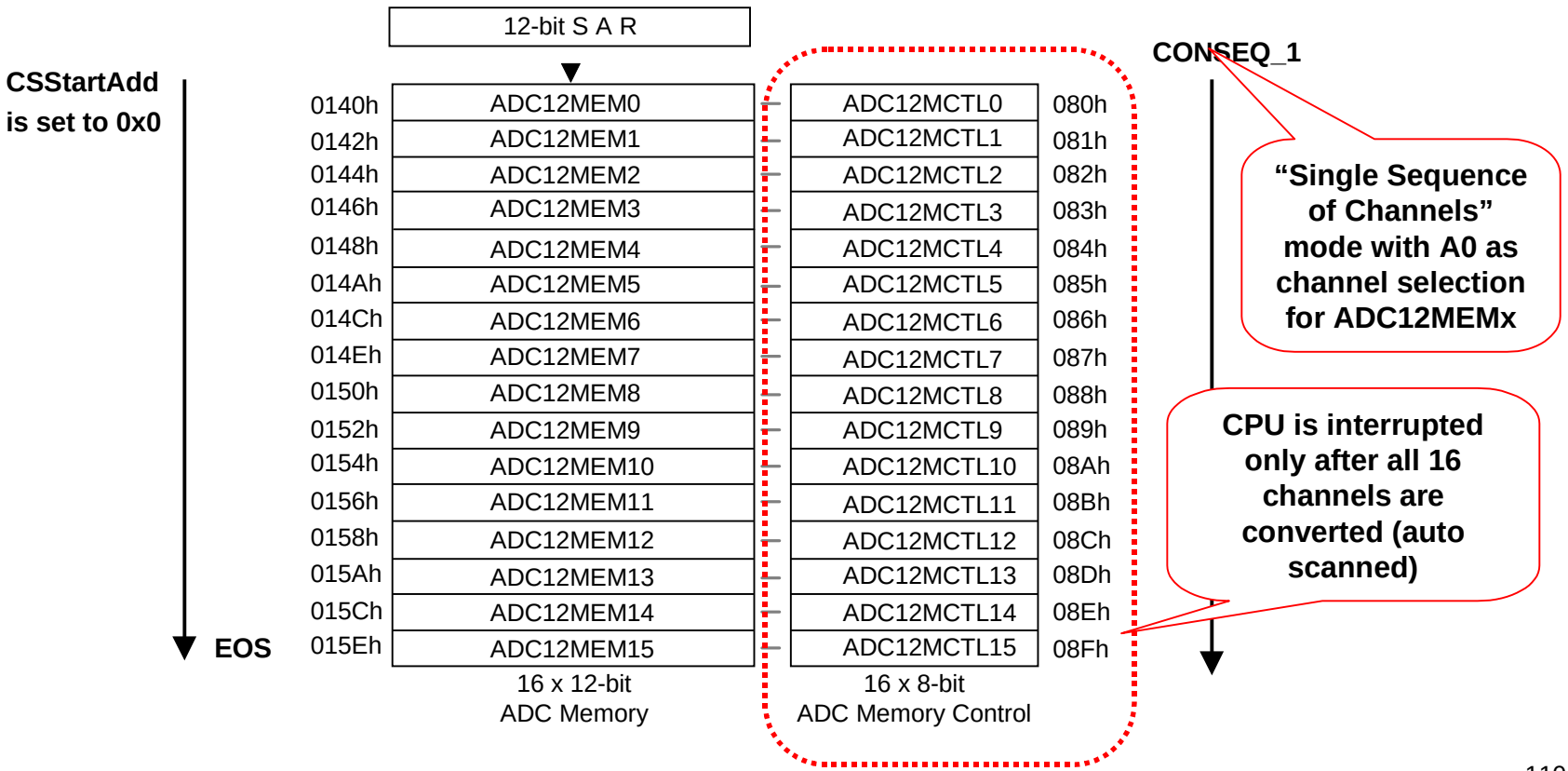
- D437 demo board running D437_4.c and D437_5.c
- FET160 target board generating 10KHz Sine-wave



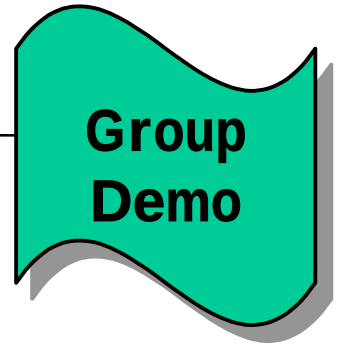
Auto-scan of ADC12

Group Demo

- Auto-scan flow showing A0 input scanned 16 times and stored automatically and sequentially in the 16 ADC12MEMx locations - ADC12MCTL15 configured to end sequence



Auto-scan of ADC12 Example D437_5.c




```
unsigned int ADCData[16],i;
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;       // Stop watchdog timer
  FLL_CTL0 |= XCAP18PF;           // Turn ON internal osc caps
  P6SEL |= 0x01;                  // Enable A/D channel A0
  TACTL = TASSEL_2+MC_1;          // Timer_A setup SMCLK UP Mode
  TACCR0 = 8;                     // Set sampling period 116.5KHz
  TACCTL1 = OUTMOD_3;             // CCR1 setup
  TACCR1 = 7;                     // For triggering ADC12
  ADC12MCTL15 |= EOS ;            // Assign end of sequence to ADCMEM15
  ADC12CTL0 = ADC12ON+SHT0_2;     // Turn on ADC12, set sample/hold time
  ADC12CTL1 = SHS_1+SHP+CONSEQ_1; // TA1 triggers sample timer,single sequence
  ADC12IE = 0x8000;               // Enable ADC12IFG.15
  ADC12CTL0 |= ENC;               // Enable conversions
  ADC12CTL0 |= ADC12SC;           // Start conversions
  _EINT();                        // Enable interrupts
  LPM0;                           // Enter LPM0

  for ( i=0; i<16; i++)
  ADCData[i] = ADC12MEM[i];      // Move ADCMEMx to RAM

  _DINT();
  ADC12CTL0 &= ~ENC;              // Clear ENC to disable conversions
  }                               // SET BREAKPOINT HERE TO READ DATA
   interrupt[ADC_VECTOR] void ADC12ISR (void)
  {
   ADC12IFG &= ~0x8000;           // Clear ADC12IFG.15
   LPM0_EXIT;                     // Exit LPM0 on return from interrupt
  }
```

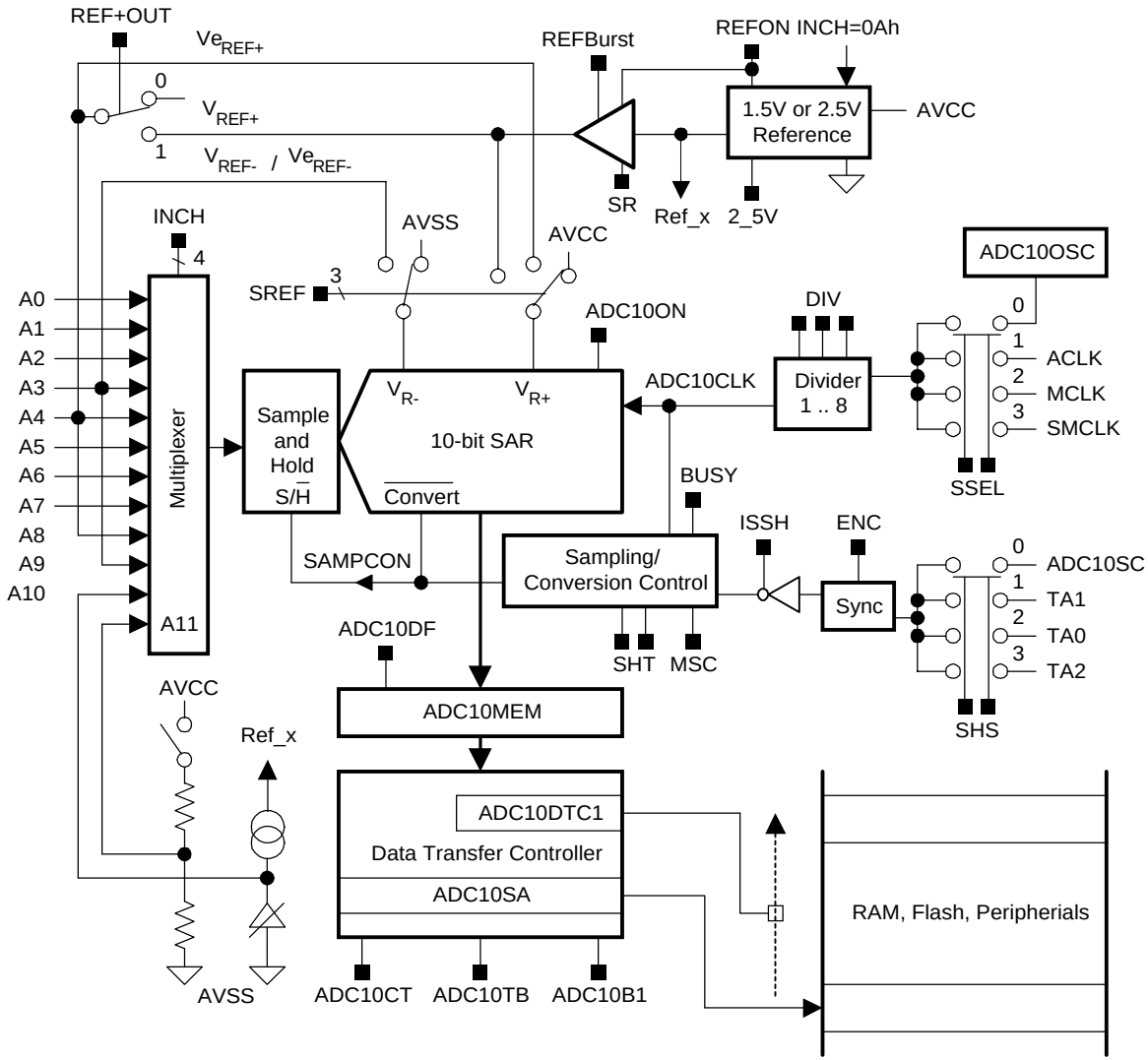
Results of Example D437_5.c

Group Demo

Software Scan	Auto-Scan
Number of Samples per cycle =	Number of Samples per cycle =
How much time does the CPU have for running useful code ?	How much time does the CPU have for running useful code ?
Is this the best way of using the ADC12 in the MSP430 ?	Is this the better way of using the ADC12 in the MSP430 ?

ADC10

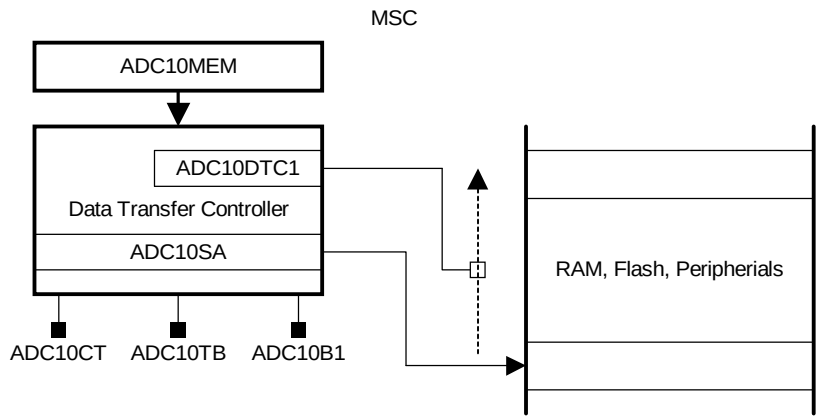
- 200ksps+
- Programmable sample & hold
- Band-gap Vref
- Temp. sensor
- Low-battery detect
- Signed/unsigned justification
- Autoscan
- Data Transfer Controller
- One interrupt vector with enable and flags in module register



MSP430 ADC10 Using DTC Example

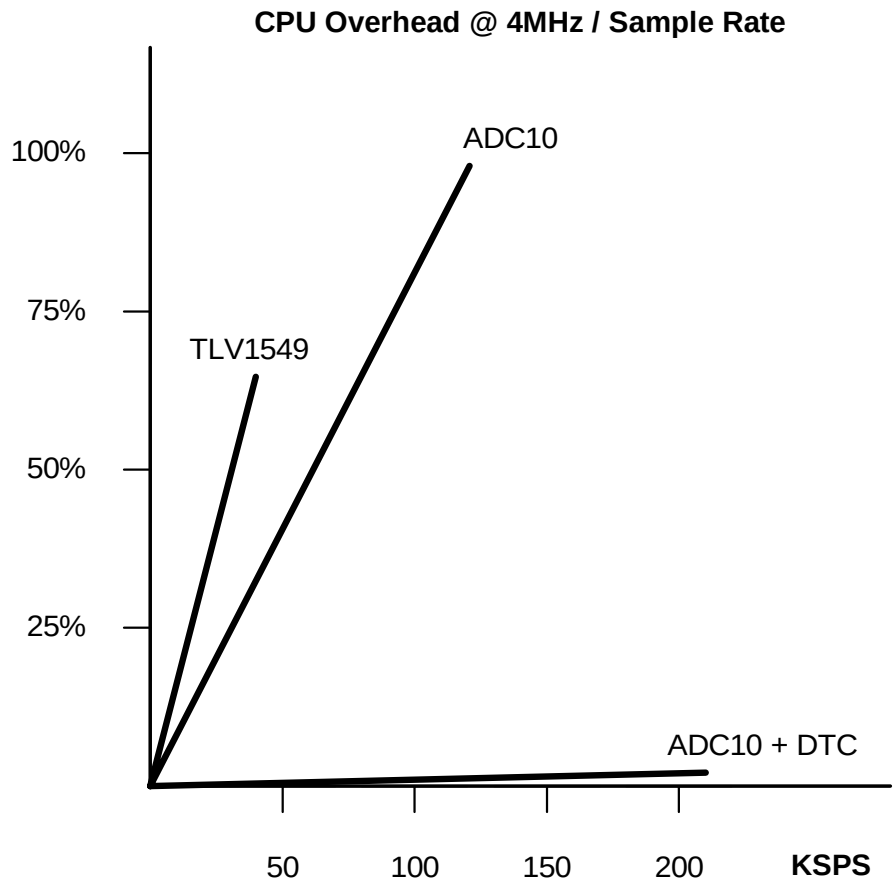
```
;ADC10 Sample A0 32x Using DTC Example - 1 MCLK per conversion
ADC_Samp32x bis.w   #ADC10SC+ENC,&ADC10CTL0 ; Enable and Start sampling
            bis.w   #CPUOFF,SR              ; CPU not required
```

DTC reduces CPU loading >50x

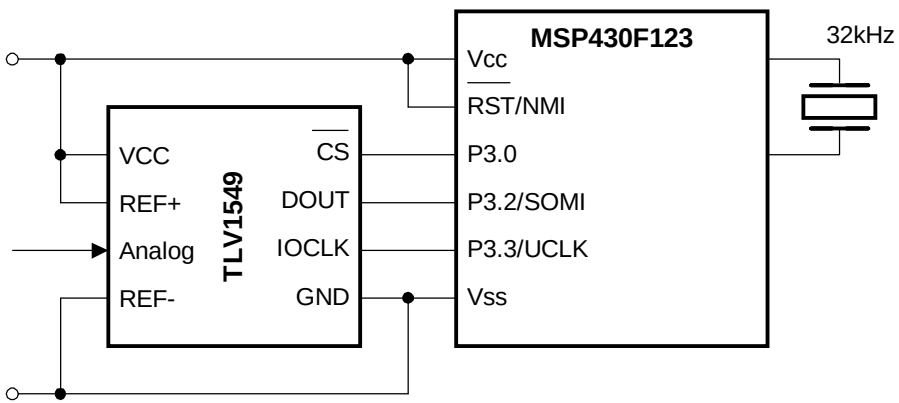


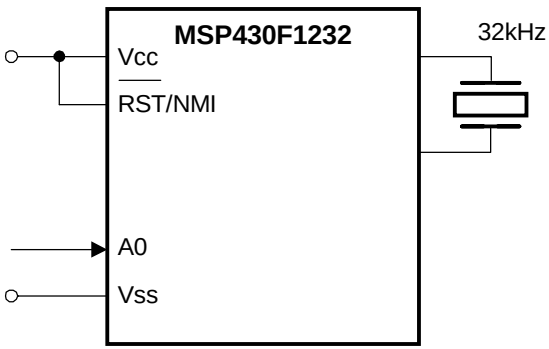

```
;ADC10 Sample A0 32x Using Software Example - >52 MCLK per conversion
            clr.w   R5                      ; Clear pointer
ADC_Loop    bis.w   #ADC10SC+ENC,&ADC10CTL0 ; (6) Enable and Start sampling
L1          bit.w   #ADC10BUSY,&ADC10CTL1   ; <--\
            jnz     L1                      ;   (41)
            mov.w   &ADC10MEM,0200h(R5)     ; <--/
            incd.w  R5                      ; (1)
            cmp.w   #020h,R5                ; (2)
            jne     ADC_Loop                ; (2)
```

ADC10+DTC

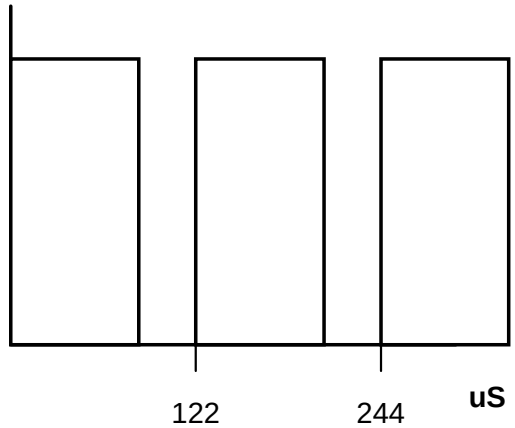


ADC10+DTC 8192SPS Data Acquisition Example

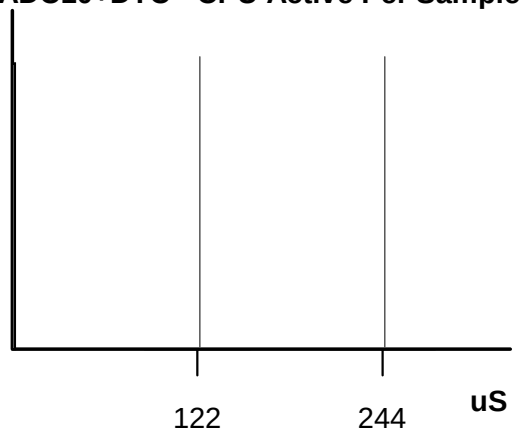




External 10-bit ADC - CPU Active Per Sample



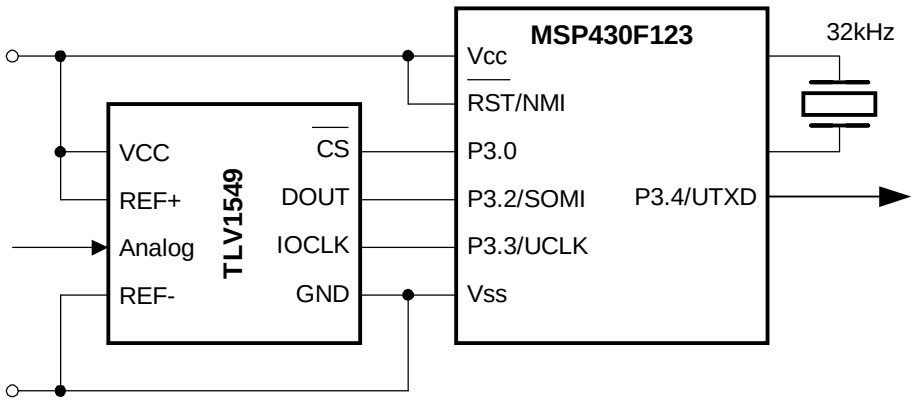
ADC10+DTC - CPU Active Per Sample



DTC relieves the CPU from data handling and servicing serial communication

ADC10+DTC

Group Demo

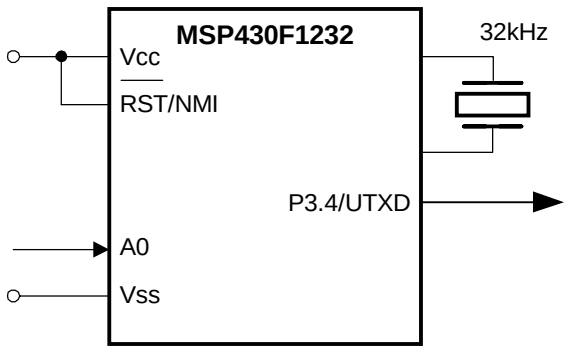


1 MIPS

2 MIPS

. . .

- Maintain 8192ksps and TX to PC every 1MIP

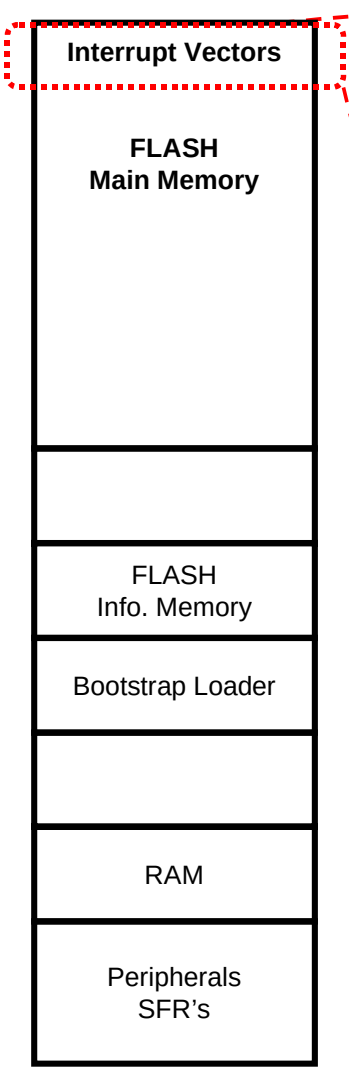


1 MIPS

2 MIPS

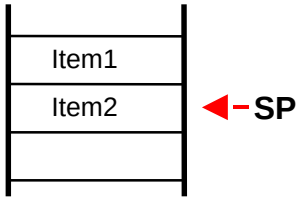
. . .

MSP430 Memory Mapped Interrupt Vectors - F11x1

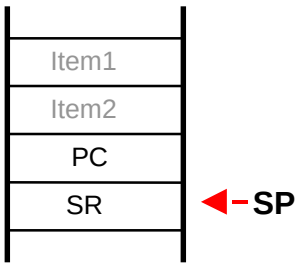


INTERRUPT	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-up	WDTIFG	Reset	0FFFEh	15, highest
NMI	NMIIFG (Note 1 &3)	(non)-maskable	0FFFCh	14
			0FFFAh	13
			0FFF8h	12
Comparator_A	CAIFG	maskable	0FFF6h	11
Watchdog timer	WDTIFG	maskable	0FFF4h	10
Timer_A	CCIFG0	maskable	0FFF2h	9
Timer_A	CCIFG1,	maskable	0FFF0h	8
			0FFEEh	7
			0FFECh	6
			0FFEAh	5
			0FFE8h	4
I/O Port P2	P2IFG.0	maskable	0FFE6h	3
I/O Port P1	P1IFG.0	maskable	0FFE4h	2
			0FFE2h	1
			0FFE0h	0, lowest

Interrupt Processing

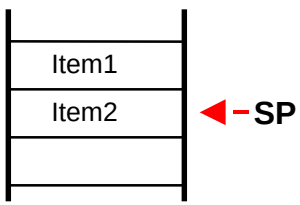


Prior to ISR



ISR hardware - automatically

- PC pushed
- SR pushed
- Interrupt vector moved to PC
- GIE, CPUOFF, OscOFF and SCG1 cleared
- IFG flag cleared on single source flags

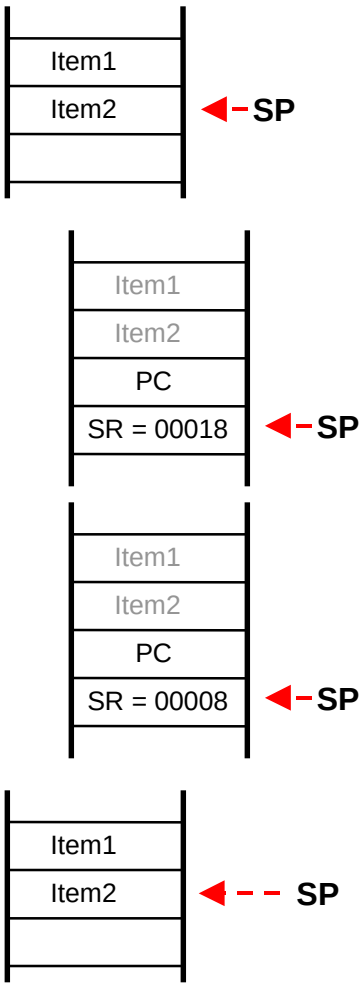


`reti` - automatically

- SR popped - *original*
- PC popped

SR and PC content saved automatically to TOS - ISR requires 11 cycles overhead

Ultra-low Power Modes In Assembler




```
#include  "msp430x11x1.h"

              ORG     0F000h                  ;
RESET         mov.w   #300h,SP                ;
SetupWDT      mov.w   #WDT_MDLY_32,&WDTCTL    ; 32ms interrupt
              bis.b   #WDTIE,&IE1             ; WDT interrupt
Mainloop      bis.w   #CPUOFF+GIE,SR          ; LPM0 with eint
              nop                             ;
              jmp     Mainloop                ;

WDT_ISR       bic.w   #CPUOFF,0(SP)           ; Clear CPUOFF bit TOS
              reti                            ;

              ORG     0FFFEh                  ;
              DW      RESET                   ;
              ORG     0FFF4h                  ;
              DW      WDT_ISR                 ;
```

ISR software modifies TOS - exit LPMx at end of ISR

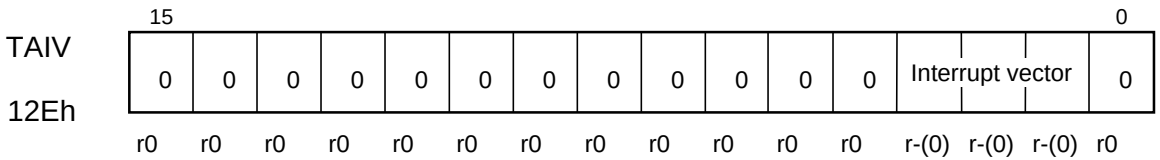
Ultra-low Power Modes In C

```
void main(void)
{
  WDTCTL = WDT_MDLY_32;                    // 32ms interrupt
  IE1 |= WDTIE;                            // WDT interrupt

  for (;;)
  {
    _BIS_SR(CPUOFF+GIE);                   // LPM0 with interrupt
    _NOP
  }
}

interrupt[WDT_VECTOR] void watchdog_timer (void)
{
    _BIC_SR_IRQ(CPUOFF);                   // Clear CPUOFF bit from TOS
}
```

ISR software modifies TOS - exit LPM3 at end of ISR

Timer_A3 Interrupt Vector Generator TAIV

- One dedicated interrupt vector for CCR0
- One shared interrupt vector for CCR1, CCR2 and TAIFG



Interrupt Priority	Interrupt Source	Short form	Vector Address	Vector Register TAIV Contents
Highest	Capture/Compare 0	CCIFG0	X	N.A.
	Capture/Compare 1	CCIFG1	Y	2
	Capture/Compare 2	CCIFG2	Y	4
	Reserved	n/a	Y	6
	Reserved	n/a	Y	8
Lowest	Timer Overflow	TAIFG	Y	10
	Reserved	n/a	Y	12
	Reserved	n/a	Y	14
	No interrupt	n/a	Y	0

Vector generator allows expansion of vectored-interrupt capability

Timer_A3 Interrupt Handling in Assembler

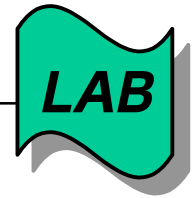
```
;-----------------------------------------------------------------
CCR0_ISR; CCIFG0 is reset automatically                          6
;-----------------------------------------------------------------
          xor.b     #01h,&P1OUT
          reti                                                  5
;
;-----------------------------------------------------------------
CCRX_ISR;                                                       6
;-----------------------------------------------------------------
          add       &TAIV,PC  ; Add offset to Jump table        3
          reti                ; Vector 0: No interrupt          5
          jmp       CCR1_ISR  ;                                 2
          jmp       CCR2_ISR  ;                                 2
          reti                ; no source
          reti                ; no source
;
TIMOVH    xor.b     #02h,&P1OUT
          reti                                                  5
;
CCR1_ISR  xor.b     #04h,&P1OUT
          reti                ;                               5
;
CCR2_ISR  xor.b     #08h,&P1OUT
          reti                ;                               5
```

Timer_A3 TAIV Handling with C

```
// Timer A0 interrupt service routine
interrupt[TIMERA0_VECTOR] void Timer_A (void)
{
  P1OUT ^= 0x01;                    // CCR0
}

// Timer_A3 Interrupt Vector (TAIV) handler
interrupt [TIMERA1_VECTOR] void Timer_A(void)
{
 switch( TAIV )
 {
   case  2: P1OUT ^= 0x02;          // CCR1
            break;
   case  4: P1OUT ^= 0x04;          // CCR2
            break;
   case 10: P1OUT ^= 0x08;          // overflow
            break;
 }
}
```

TAIV used as case indicator

D437_3.c Reducing Power Consumption Using LPM3



- Modify the main function of D437_3.c to LPM3 properly

```
void main(void)
{
  init_sys();
  check_cal();
  Refcal_ram = Refcal_flash;
  calmode = (P2IN & PB_VOLT);
  while (1)
  {
   // THE FOLLOWING LINES SHOULD BE UNCOMMENTED FOR HIGH POWER DEMO
   IE2 = 0x00; // Disable Basic Timer Interrupt
   while (!(IFG2&BTIFG)); // Poll for Basic Timer Interrupt
   IE2|= BTIFG; // Enable Basic Timer Interrupt
   // THE FOLLOWING LINE SHOULD BE UNCOMMENTED FOR ULTRA LOW POWER DEMO
   // LPM3; // The main loop is synchronous to the 1 second interrupt.
    switch (mode)
     {
       case TIME: time(); break;
       case TEMPERATURE: temperature(temp_mode); break;
       case VOLTAGE: voltage(); break;
       default: break;
     }
  } // while (1)
} // main()
```

comment

uncomment

Notes

Agenda - Dallas, TX - November 2002

Digital Communication

JTAG and Embedded Emulation:

- JTAG description and functionality
- Designing for embedded emulation
- In-system debugging with on-chip emulation logic

GPIO:

- Port initialization, interrupts and alternative function selection
- Understanding current source and sink capability

USART:

- Peripheral module initialization sequence
- Effective USART baud-rate generation
- Using the USART in low power applications
- SPI master and slave mode selection

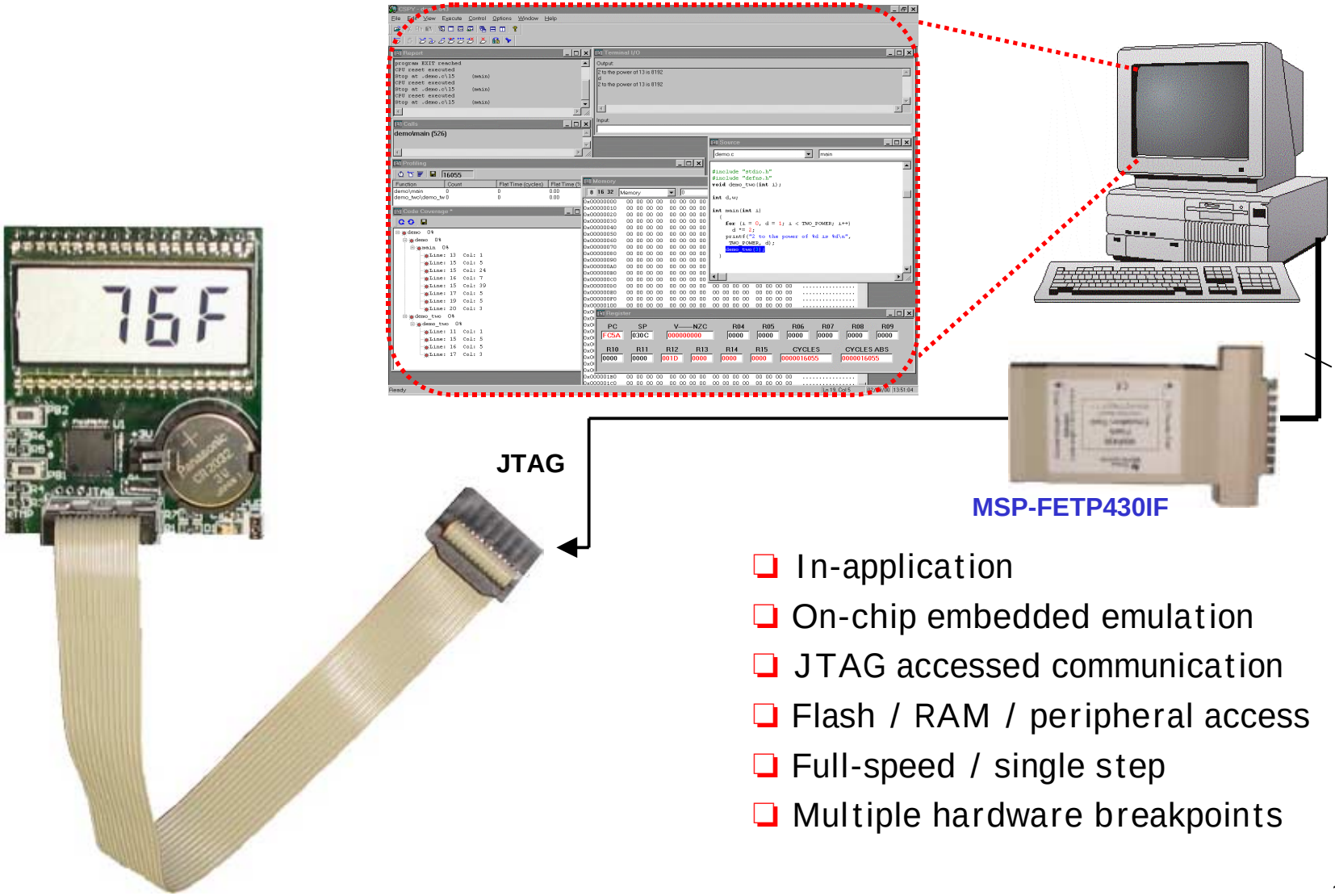
LCD:

- Display memory organization
- Contrast control and temperature compensation
- LCD design process and efficient coding examples

LAB Session 3

Lab: USART capabilities from ultra-low LPM3
Lab: LCD contrast control demonstration

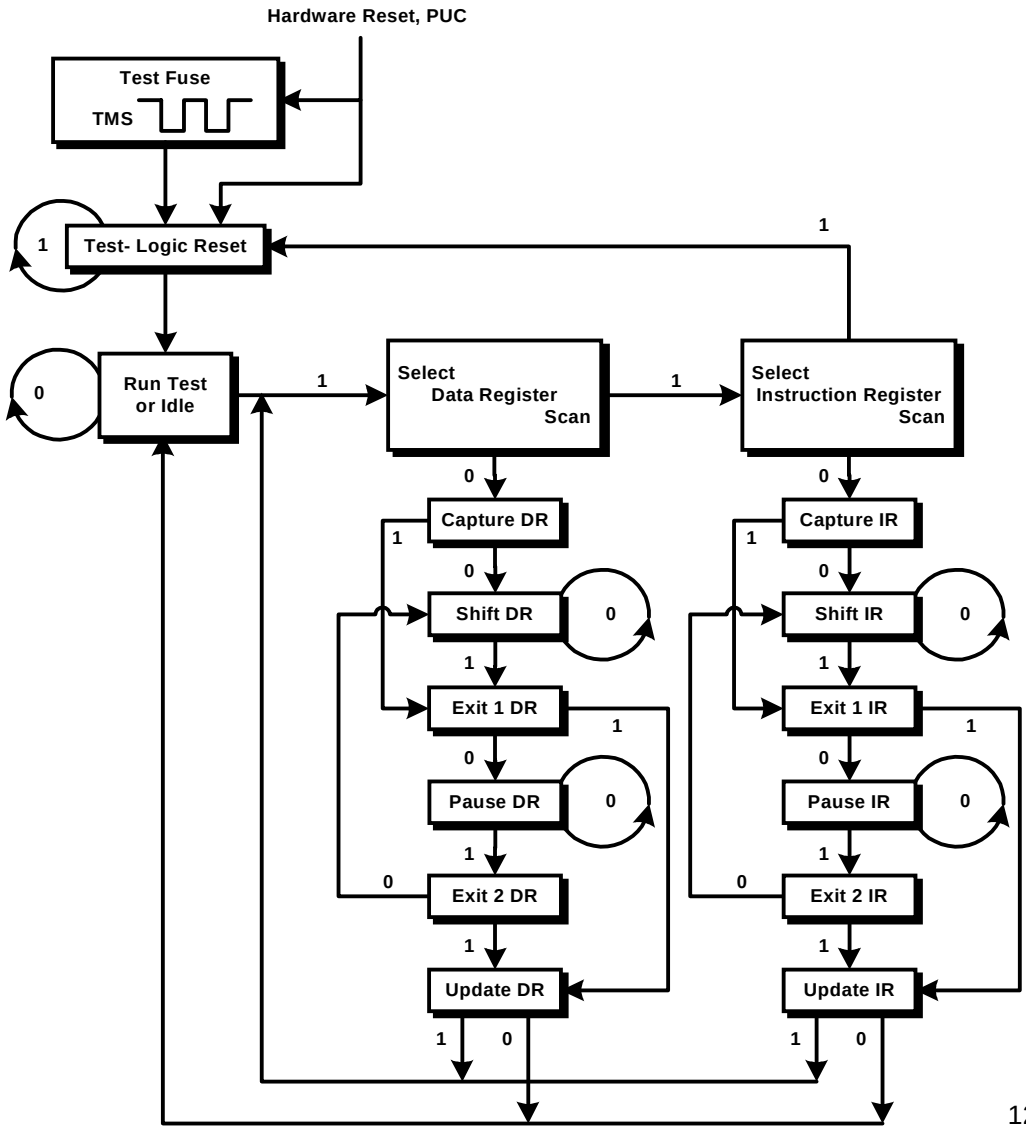
Embedded Emulation = Real-time System Debug



- In-application
- On-chip embedded emulation
- JTAG accessed communication
- Flash / RAM / peripheral access
- Full-speed / single step
- Multiple hardware breakpoints

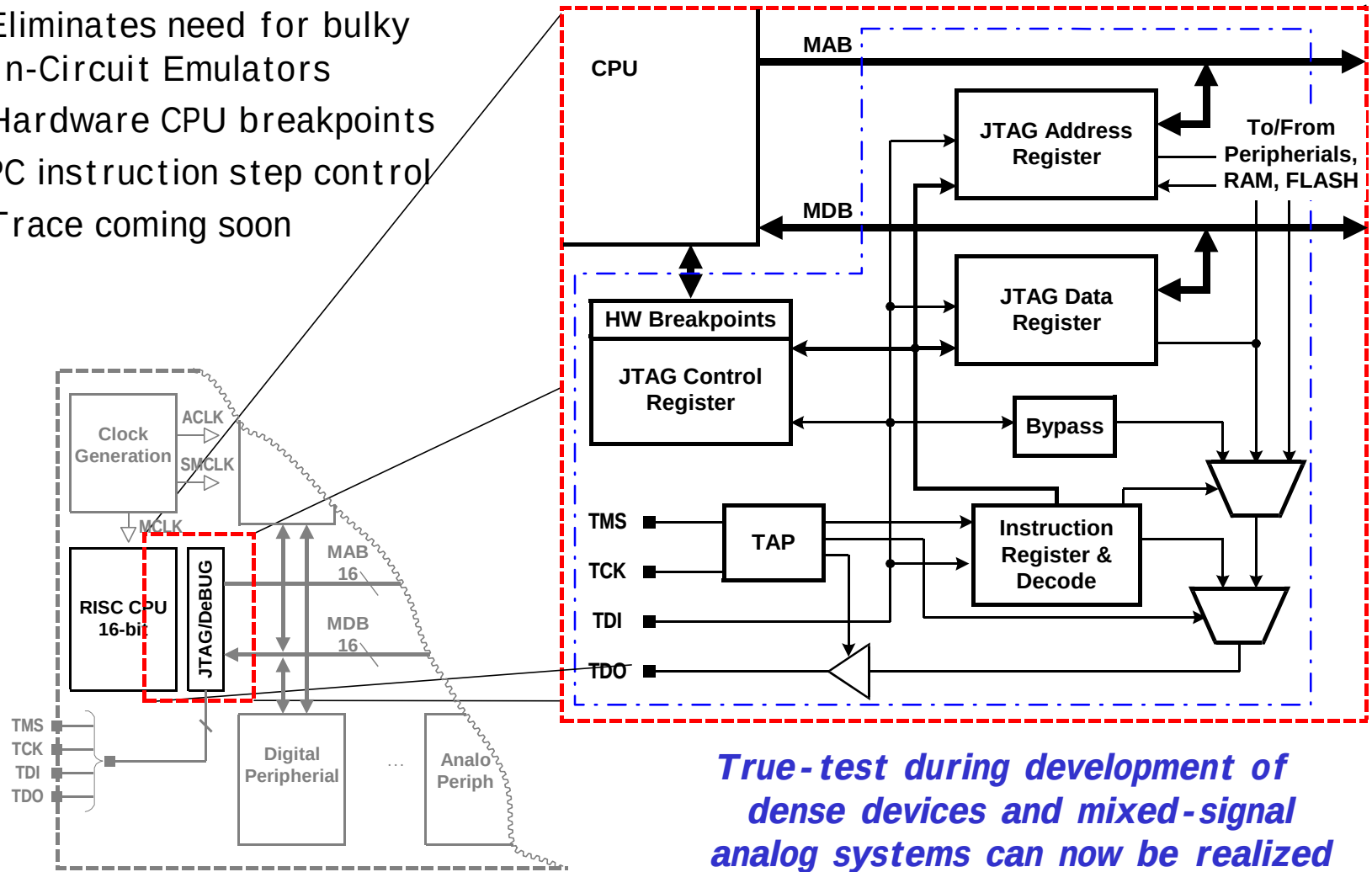
MSP430 and IEEE 1149.1 JTAG

- IEEE 1149.1 compliant *but no boundary scan*
- TAP Controller
- Instruction register
- Instruction decoder
- Eight data registers
- Emulation & breakpoint Logic
- JTAG security fuse
- JTAG data Output



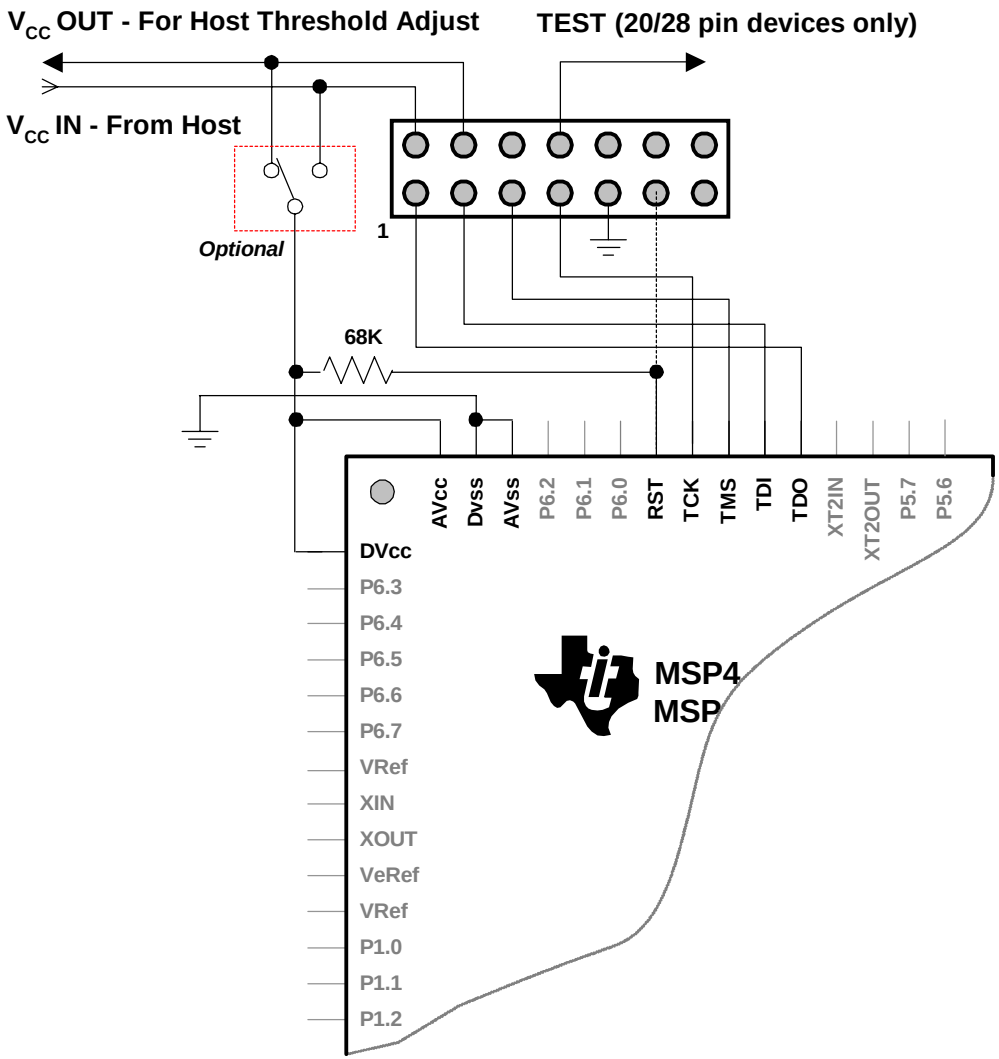
MSP430 On-Chip Emulation Logic

- Eliminates need for bulky In-Circuit Emulators
- Hardware CPU breakpoints
- PC instruction step control
- Trace coming soon



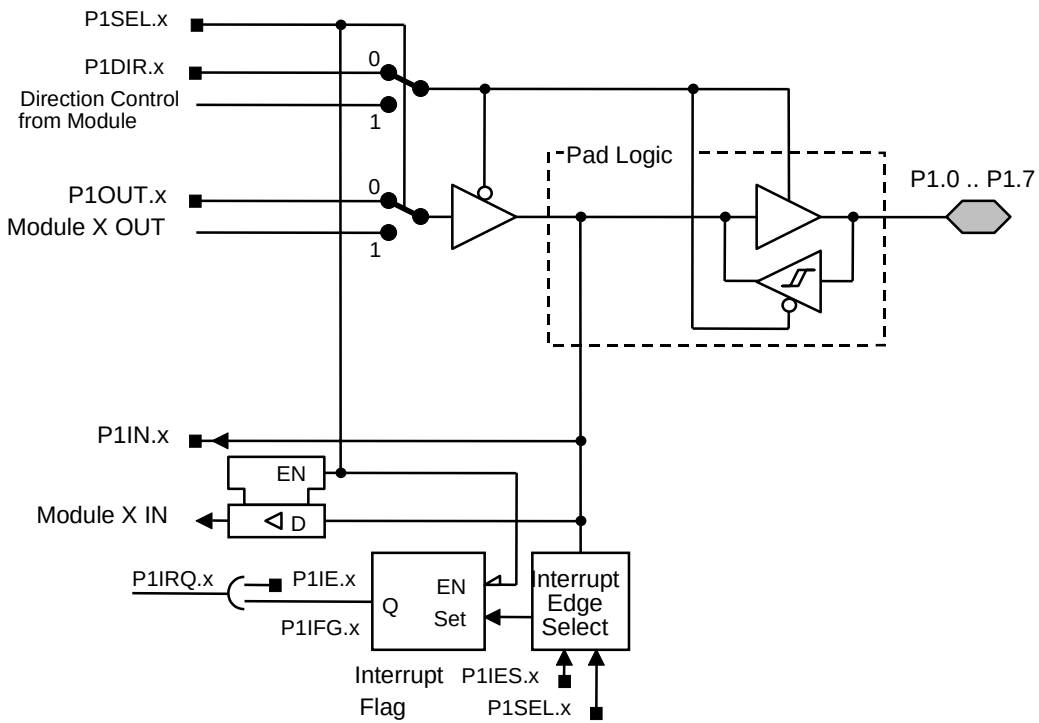
True-test during development of dense devices and mixed-signal analog systems can now be realized

Emulation Supported Design



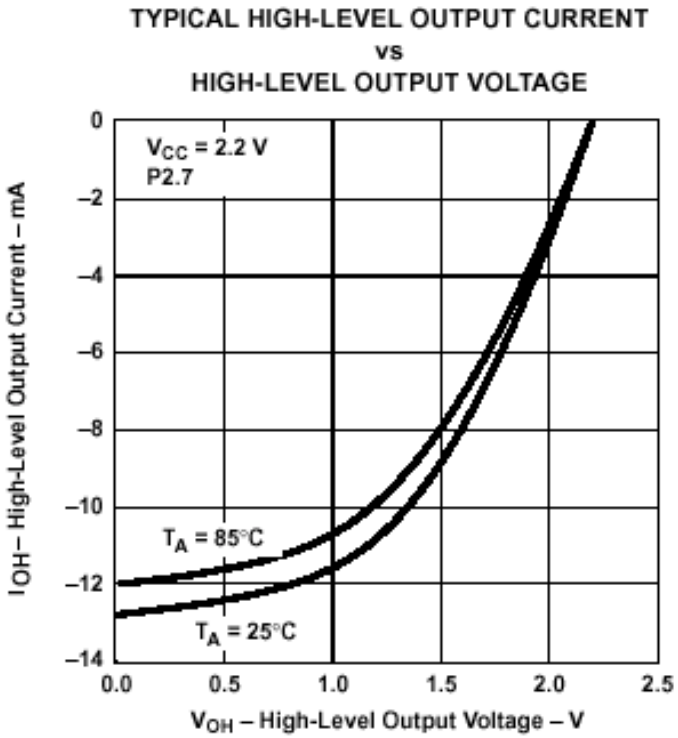
- TDO, TDI, TMS, TCK - leave unconnected if not used in-application
- PWR & GND - in-system Vcc feedback (or) programmer-supplied
- RST pull to DVcc w/ 68k (optional header connection)
- TEST required only for 20/28-pin devices
- JTAG security fuse programmed via TDI (or TEST for 20/28 -pin devices) 6-7V @ 100mA max applied for 1ms

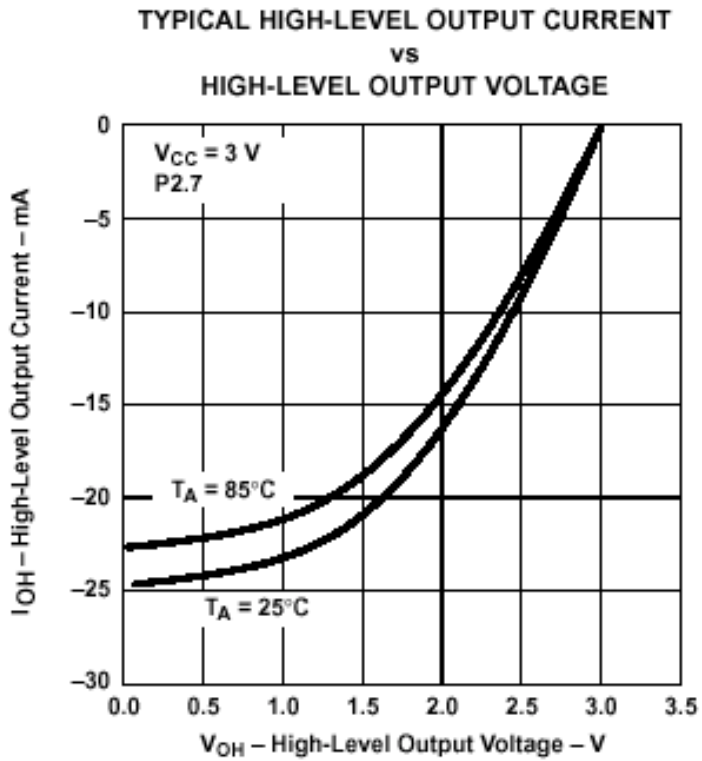
Port Schematic



Port output and input latches are collected in different registers

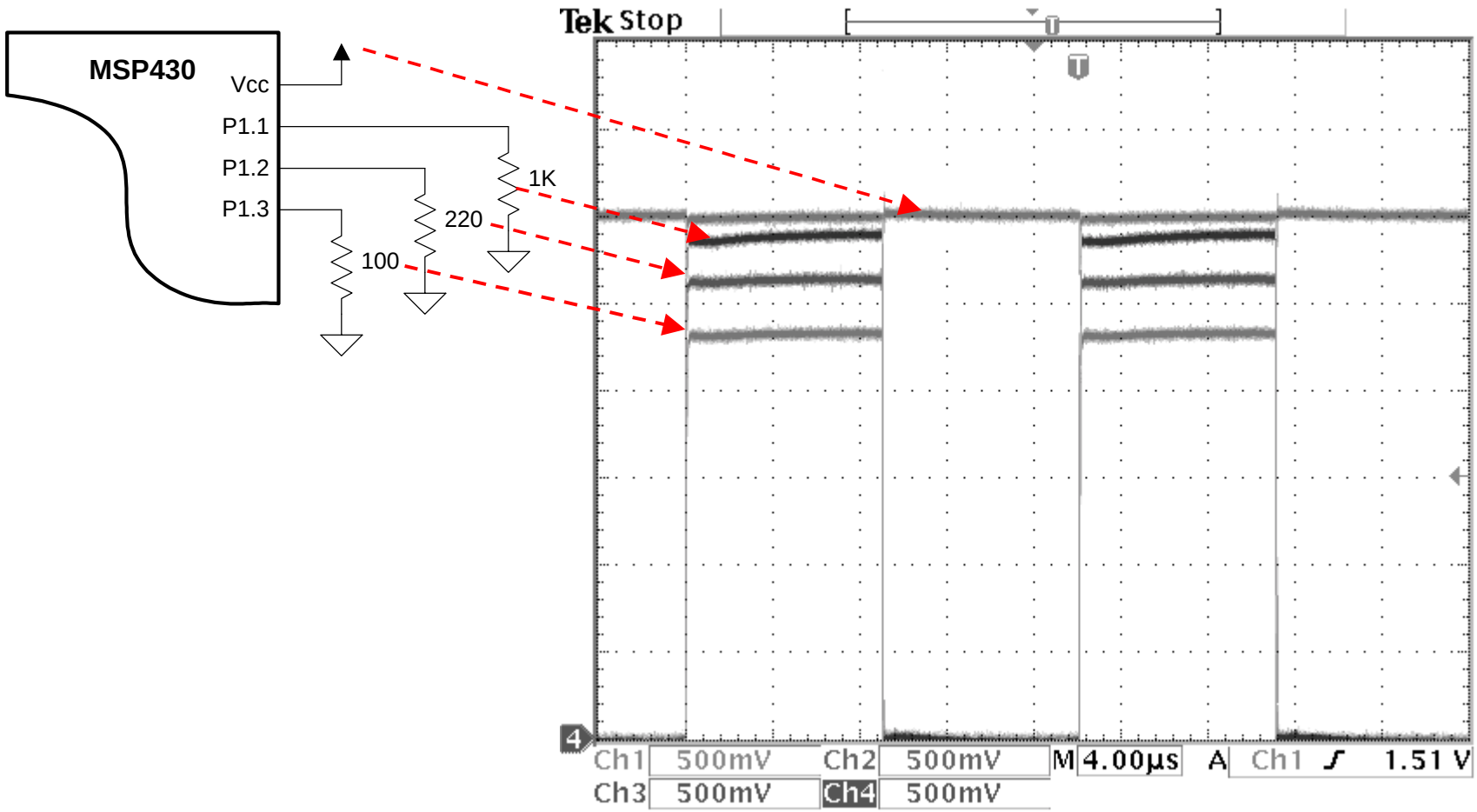
Port Drive Data Sheet Specification



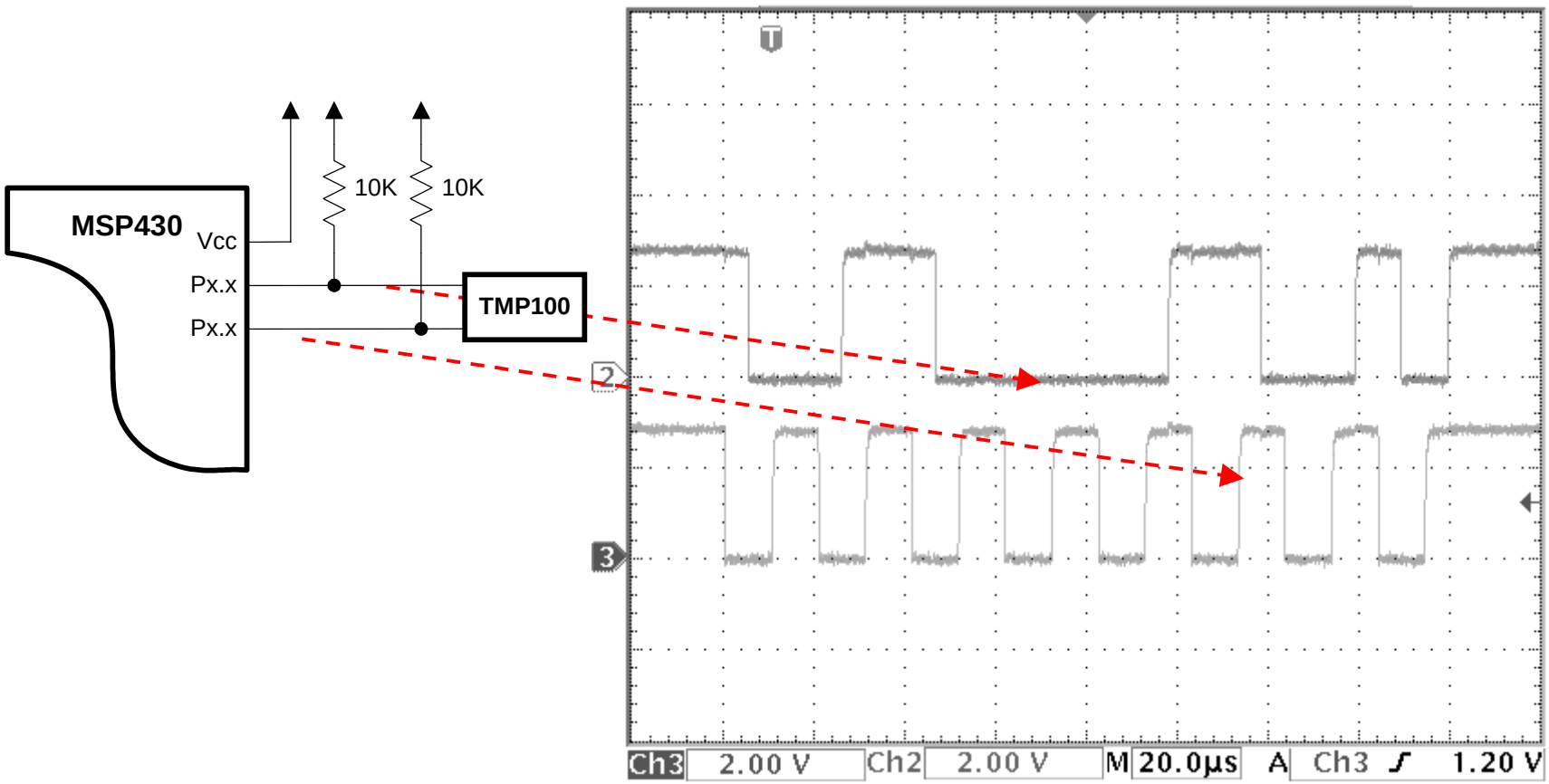


- The R_{DSON} of the CMOS elements limit the current that can be sourced or sunk from each output port pin

Port Driving A Load In The Real World



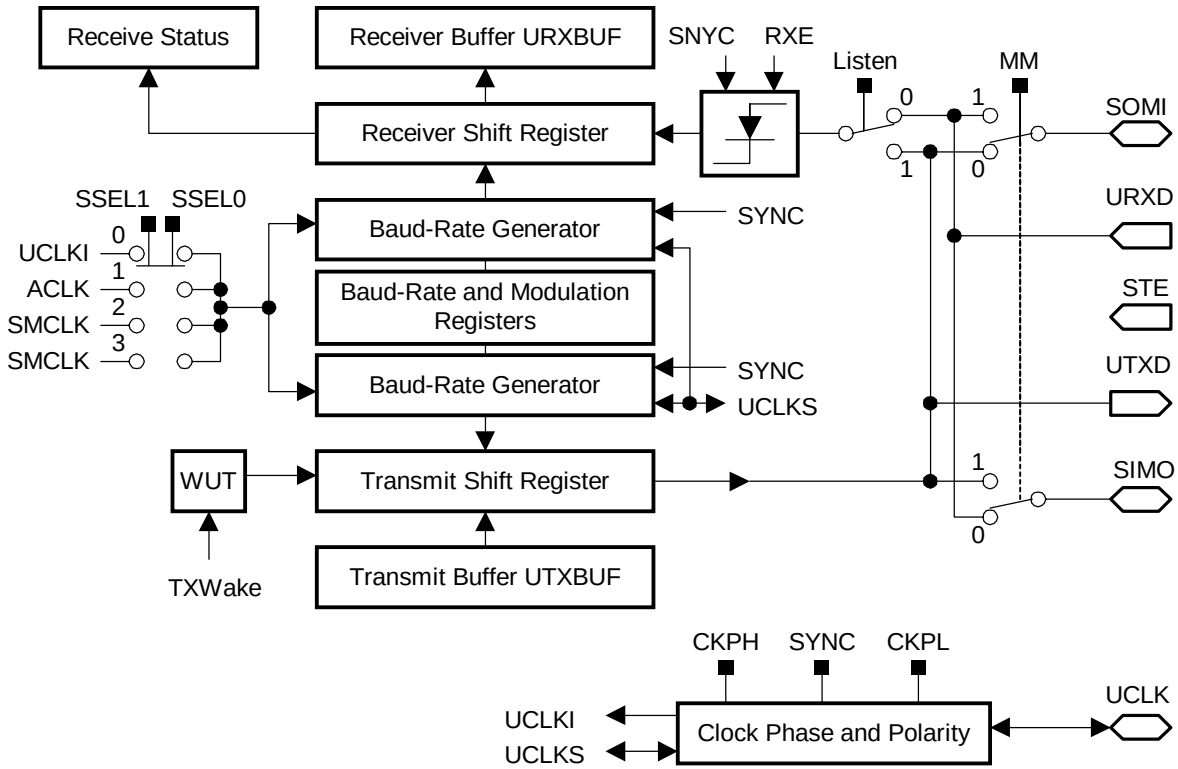
Port Configured As Virtual Open Drain / Collector



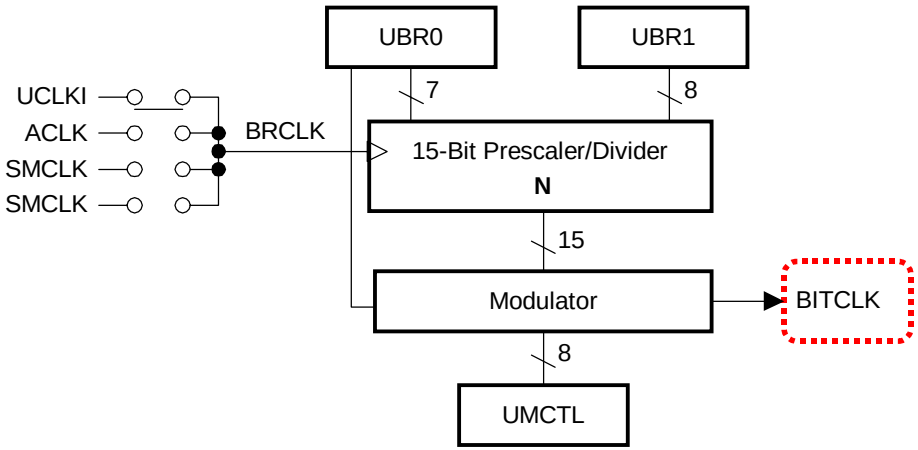

```
bic.b   #002h,&P1OUT    ; load zero for output pin
                        |
bis.b   #002h,&P1DIR    ; output direction = drive low
bic.b   #002h,&P1DIR    ; input direction = signal high
```

USART Serial Port

- Software selectable UART or SPI
- Auto-start from any LPMx
- Double buffered RX and TX shift registers
- Baud-rate generator
- 7 or 8-bit data
- 9-bit addressing mode available
- Error detection and suppression
- Two interrupt vectors with enable and flags in SFR and module register



USART UART Baud Rate Generator



Example:

ACLK = 32768

Baud = 2400 --> 32768/2400 = 13.68

UBR1 | UBR0 | UMCTL = 00h 0Ch 6Bh

BITCLK = ACLK/13 ACLK/14 ACLK/14 ACLK/13 ACLK/14 ACLK/13 ACLK/14 ACLK/14

Modulator mixes adjacent clock dividers to enable high baud rates even with low frequency XTAL.

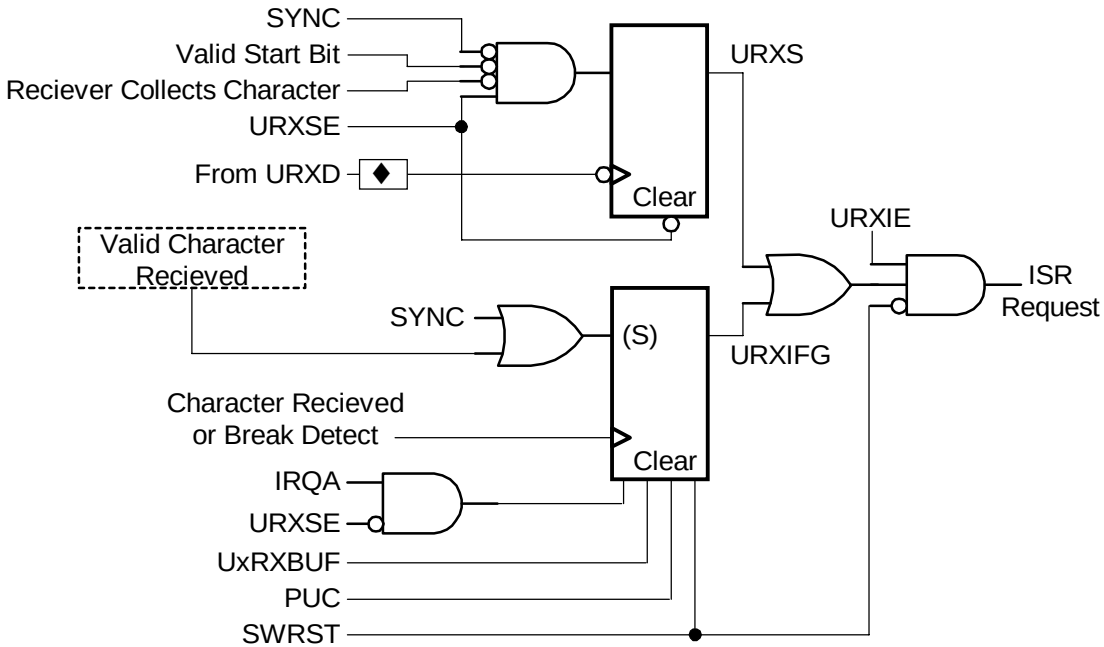
USART Configuration Sequence

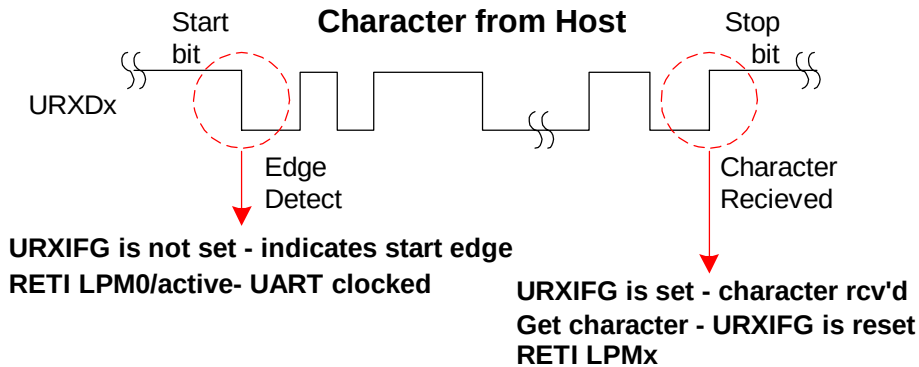
1. Configure all USART peripheral module registers and appropriate module enable (MEx) SFR keeping SWRST set
2. Reset SWRST - this loads USART state machine
3. Enable interrupt in appropriate interrupt SFR register

```
// MSP430F123 Example
ME2 |= UTXE0 + URXE0;                    // Enabled USART0 TXD/RXD
UCTL0 |= CHAR;                           // 8-bit character
UTCTL0 |= SSEL0;                         // UCLK = ACLK
UBR00 = 0xBA;                            // 3.58Mhz 19200 - 186
UBR10 = 0x00;
UMCTL0 = 0x00;                           // no modulation
UCTL0 &= ~SWRST;                         // Initialize USART state machine
IE2 |= URXIE0;                           // Enabled USART0 RX interrupt
```

USART module enable (MEx) bits are specific to each MSP430's - see device specific datasheet

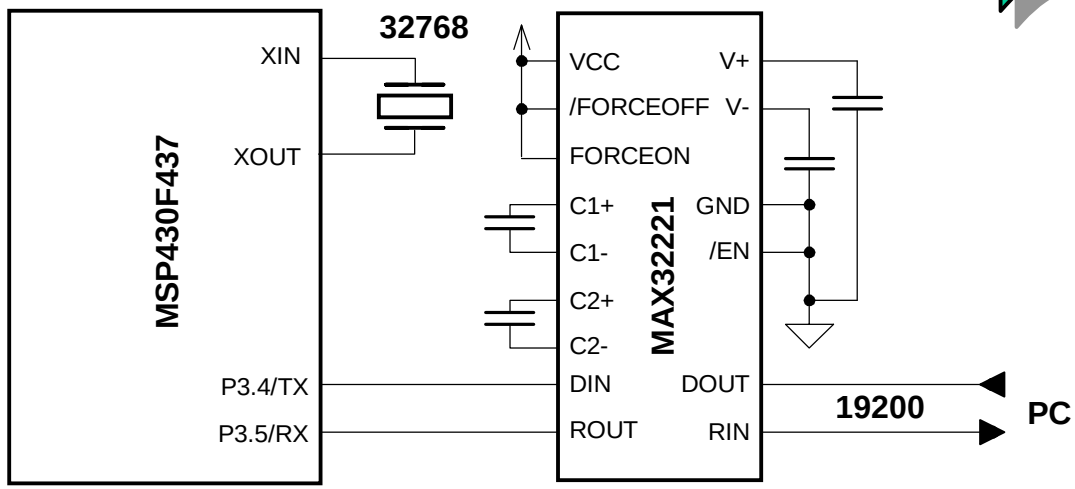
UART RX-Edge Detect Logic And Error Suppression





USART RX-Edge Detect Example

Instructor Demo



- MSP430 is normally in LPM3 ~ 1uA - hardware FLL automatically calibrates the DCOCLK with no CPU resources. The DCOCLK is used for UART baud rate generation.
- RX-Edge Detect will start DCO on start bit edge - clock is now available for balance of character.
- Periodically FLL must be on to calibrate DCOCLK.

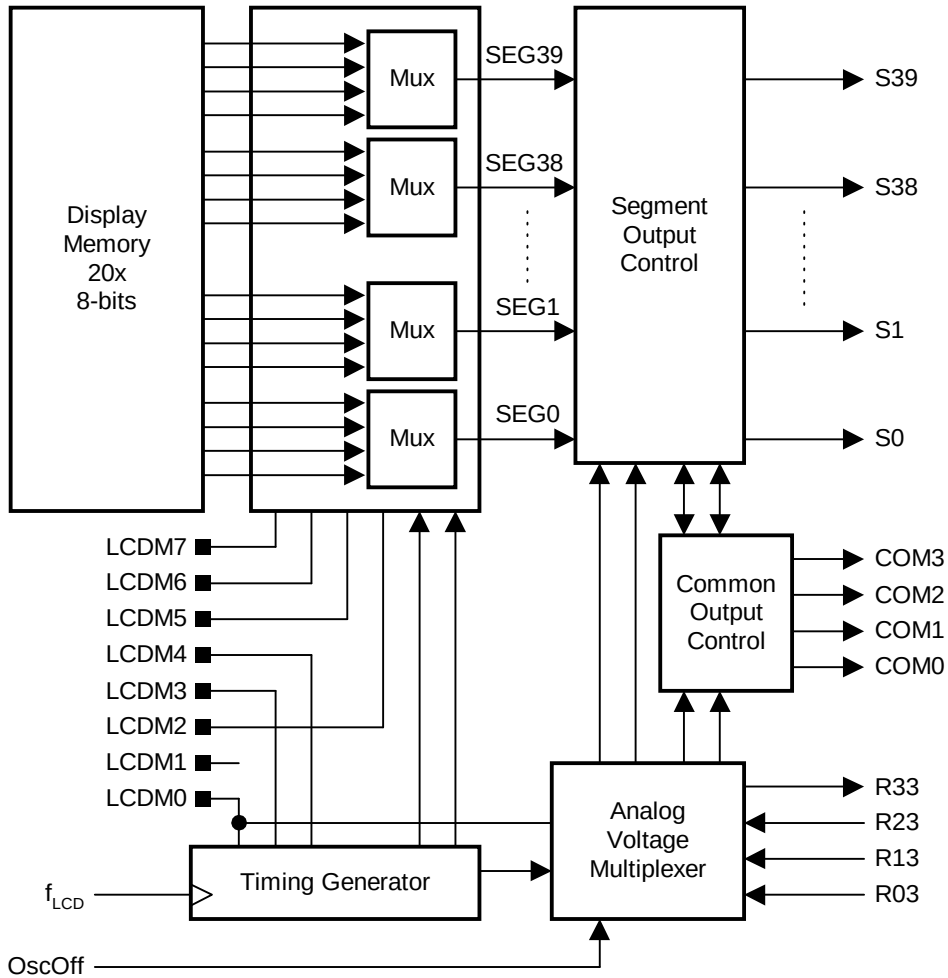
RX-Edge Start DCO on start bit - allow fast UART support from LPM3

What is The Maximum Speed of the MSP430 Serial?

- Asynchronous - UART or synchronous -SPI
- What clock is being used?
- XTAL?
- DCO?
- HW FLL?
- Software FLL?
- What baud rate error can be tolerated?
- What CPU loading can be tolerated?

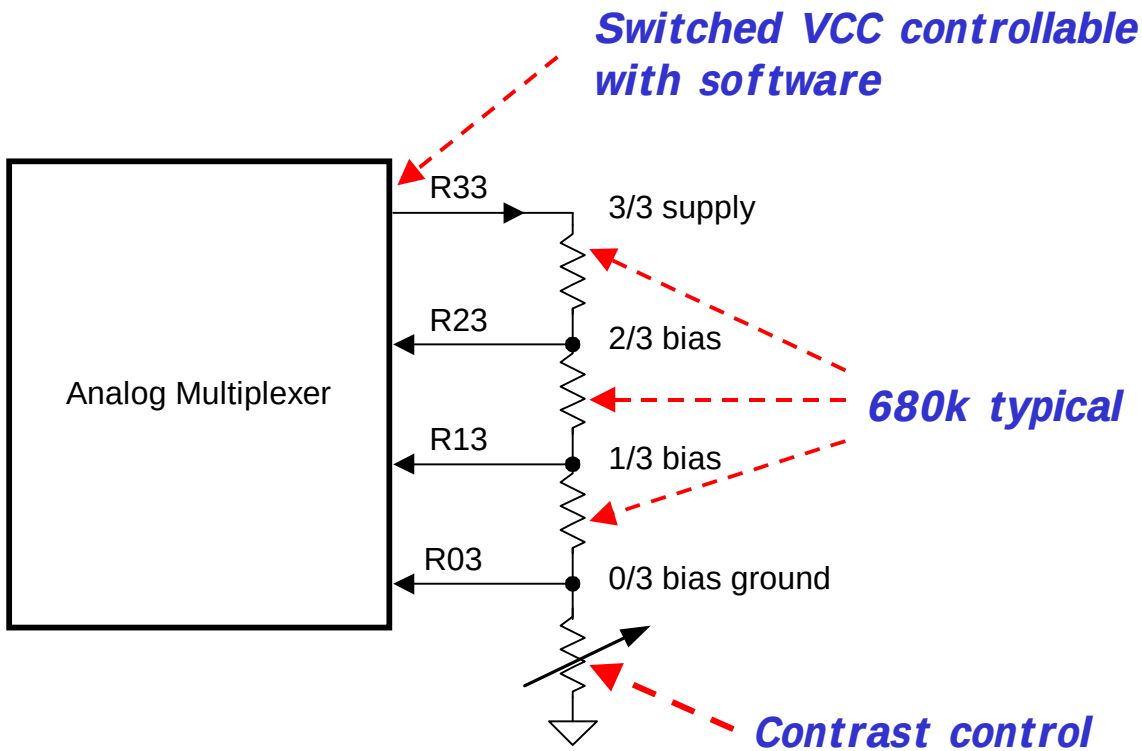
MSP430x4xx LCD Driver

- *Ultra-low* power
- Fully automatic
- 4/3/2 mux or static
- 44x = 160-bit display

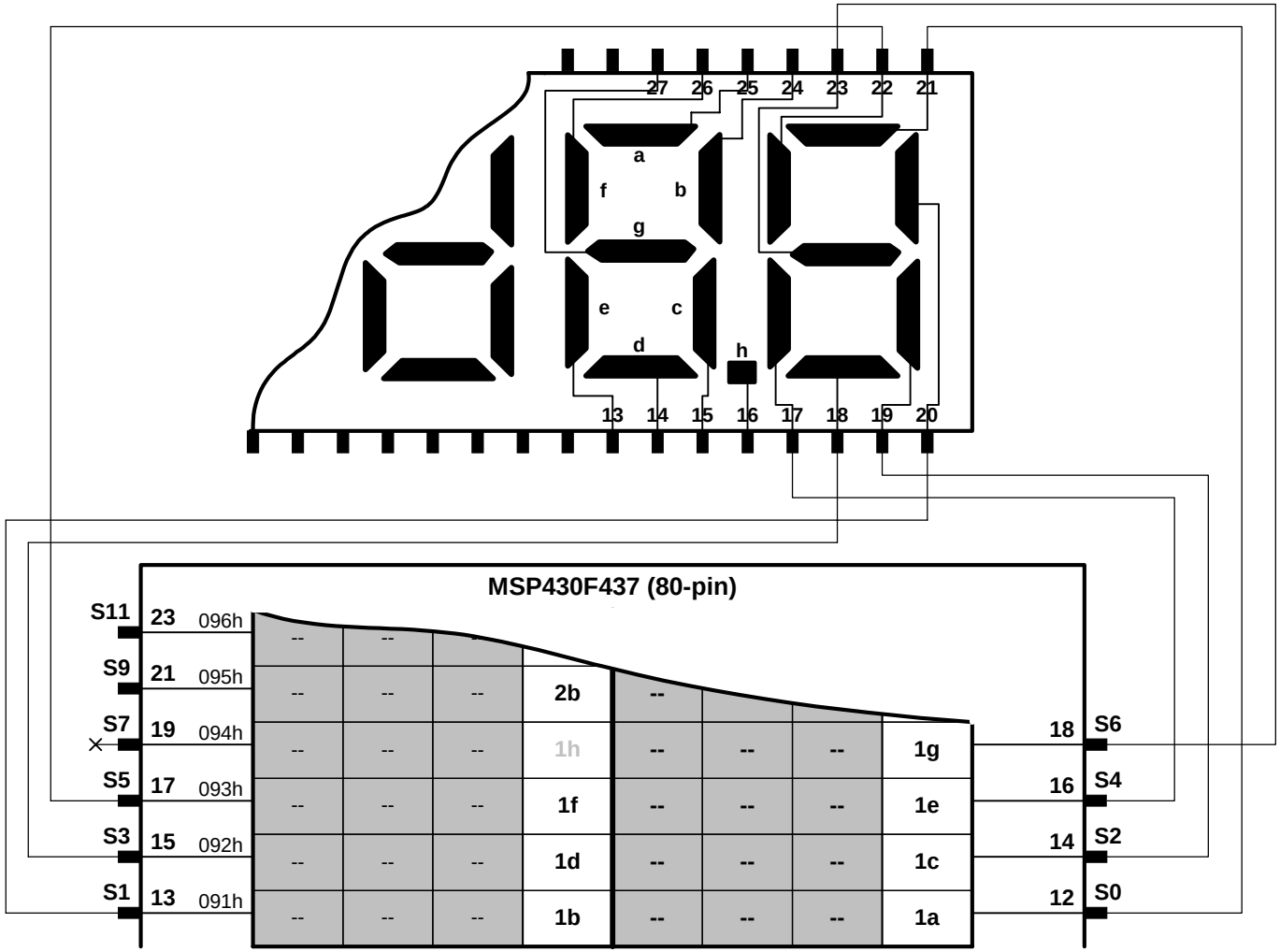


MSP430x4xx LCD Bias and Contrast Control

Instructor Demo



Mapping Segment Display Memory Effectively



Repeat for all additional digits

Notes

Agenda - Dallas, TX - November 2002

Next Generation Modules

I2C Hardware Module:

- Introduction to the I2C module
- Clock generation and synchronization

DAC12:

- Introduction to the DAC12 module
- Using hardware triggers to reduce signal aperture error

DMA:

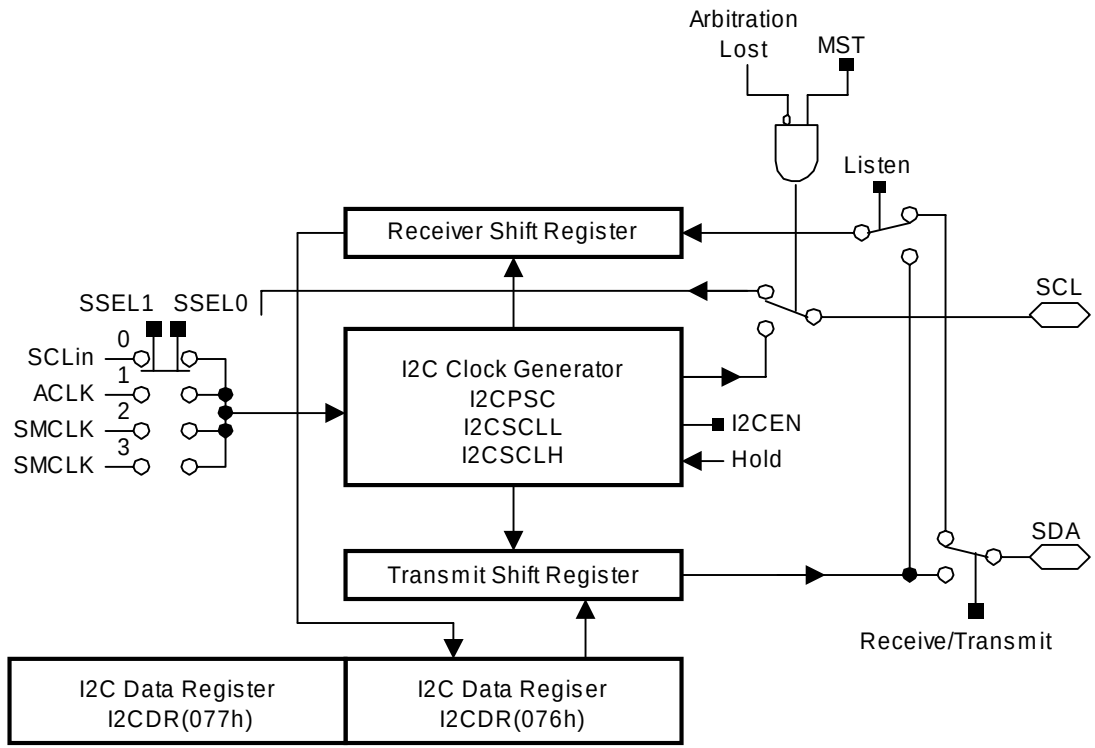
- Introduction to the DMA module
- Triggering DMA transfers
- Reducing CPU overhead with DMA
- Using DMA with low power modes

LAB Session 4

Lab: Demonstration of I2C hardware and software interface solutions
Lab: CPU-less signal generation using DMA

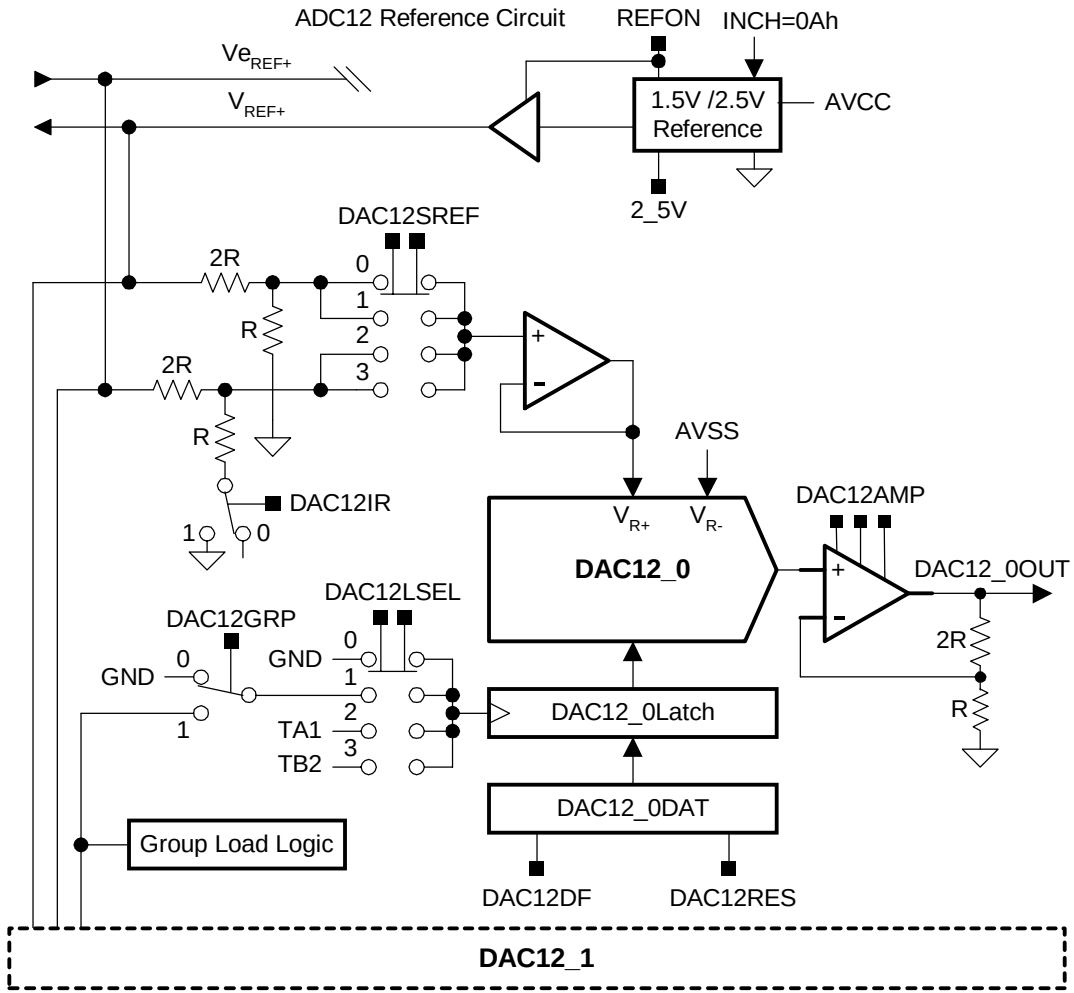
I2C Module

- I2C is an extension of USART0 on the ‘F15x/16x
- Support for 100 Normal 400kbps and Fast modes
- 7 and 10-bit addressing
- Full master/slave compliance
- Byte and word data format
- Useable to trigger DMA for high-speed support



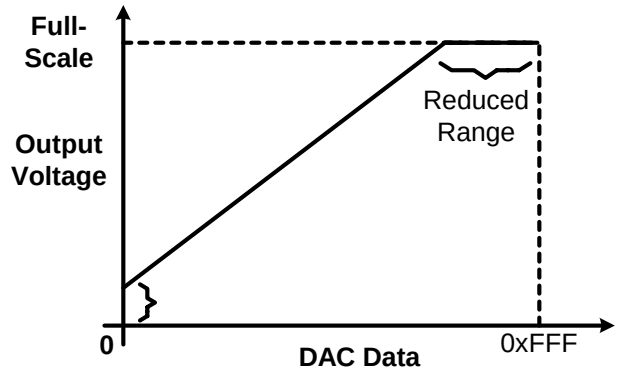
DAC12 Module

- Programmable settling time vs. power consumption
- 12-bit monotonic
- Internal or external VRef
- Binary or 2's compliment data format
- Self-calibration option for offset
- Synchronized or non-synchronized updates of multiple DAC12's
- Useable with the DMA module
- 8/12-bit voltage output

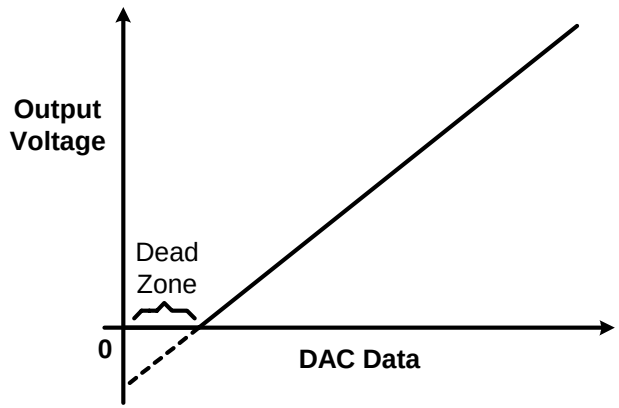


DAC12 Offset and Calibration

- Setup settling time
- Set bit DAC12CALON
- Will auto-reset upon calibration completion
- ~11ms calibration time
- ~10x reduction in offset error after calibration



Positive Offset

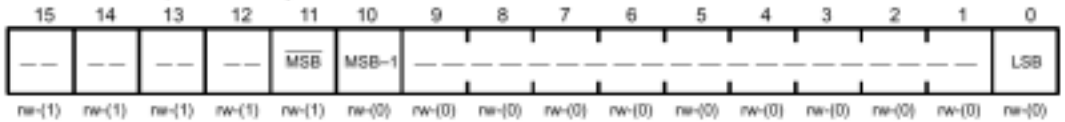


Negative Offset

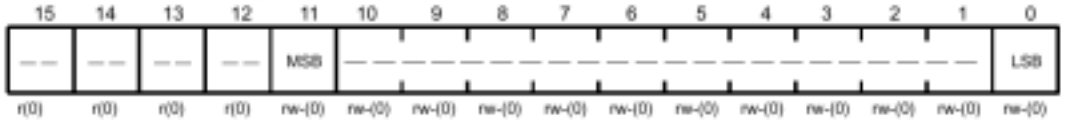
DAC12 Resolution & Data Formatting

- Upper 4 bits are truncated during word access
- When in 8-bit mode, upper 8 bits are ignored

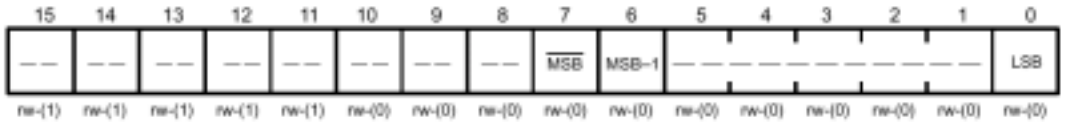
12-Bit Mode, 2s Compliment Data Format



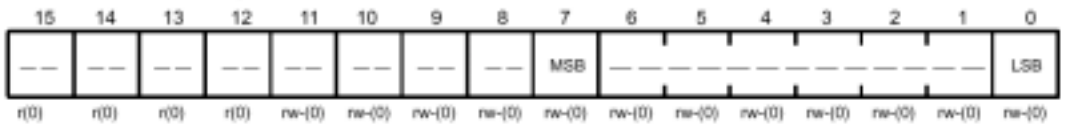
12-Bit Mode, Binary Data Format

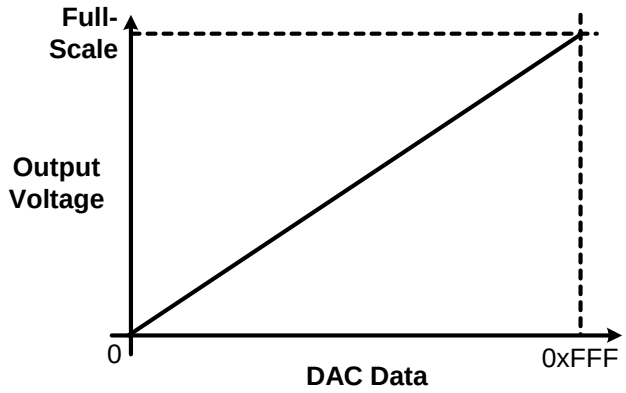


8-Bit Mode, 2s Compliment Data Format

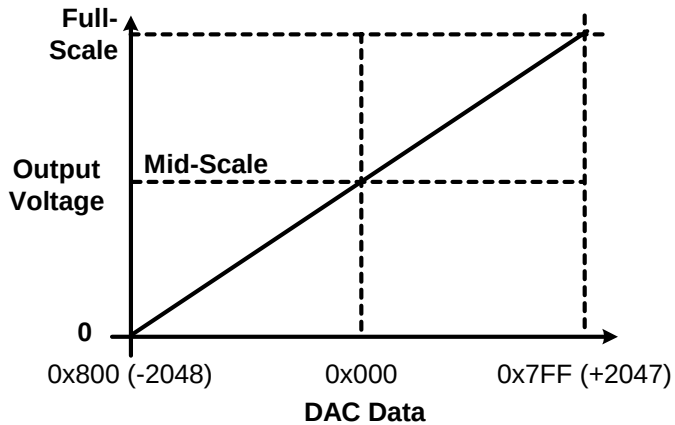


8-Bit Mode, Binary Data Format





12-bit, straight binary mode



12-bit, 2's compliment mode

Updating the DAC12

- Four update methods
  - Data in DAC12.xDAT is applied directly to the DAC core
  - DAC12.xDAT is latched into buffer when all grouped DACs have received new data
  - DAC12.xDAT is latched into buffer on next positive edge from Timer_A3 OUT3
    DAC12.xDAT is latched into buffer on next positive edge from Timer_B7 OUT2
- Data is latched only after DAC12.xDAT register(s) is updated (may re-write previous data)
- DAC12IFG signals data has been latched and new data can be written to DAC12.xDAT

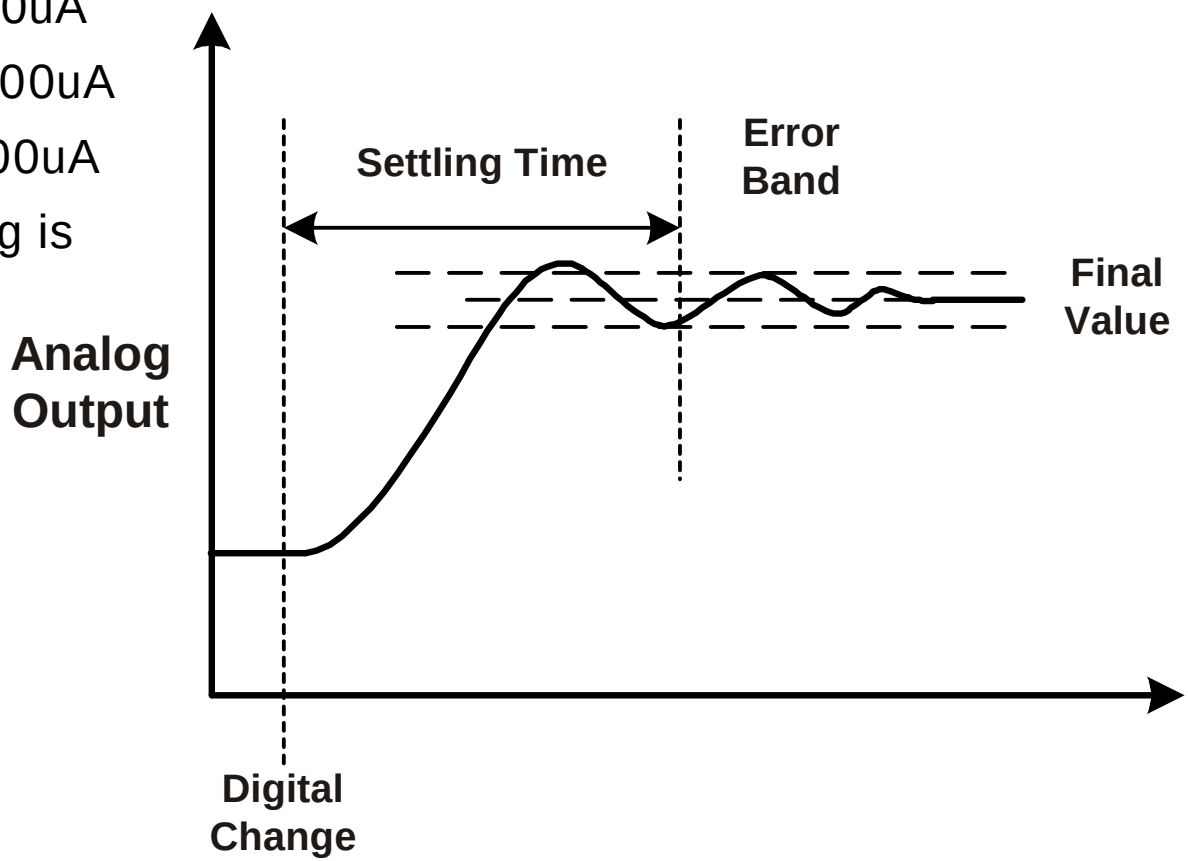
DAC12 Settling Time and Power Consumption

- Three programmable output amplifier settling time options:

  Low: t_S=60us, I_{DAC}=120uA

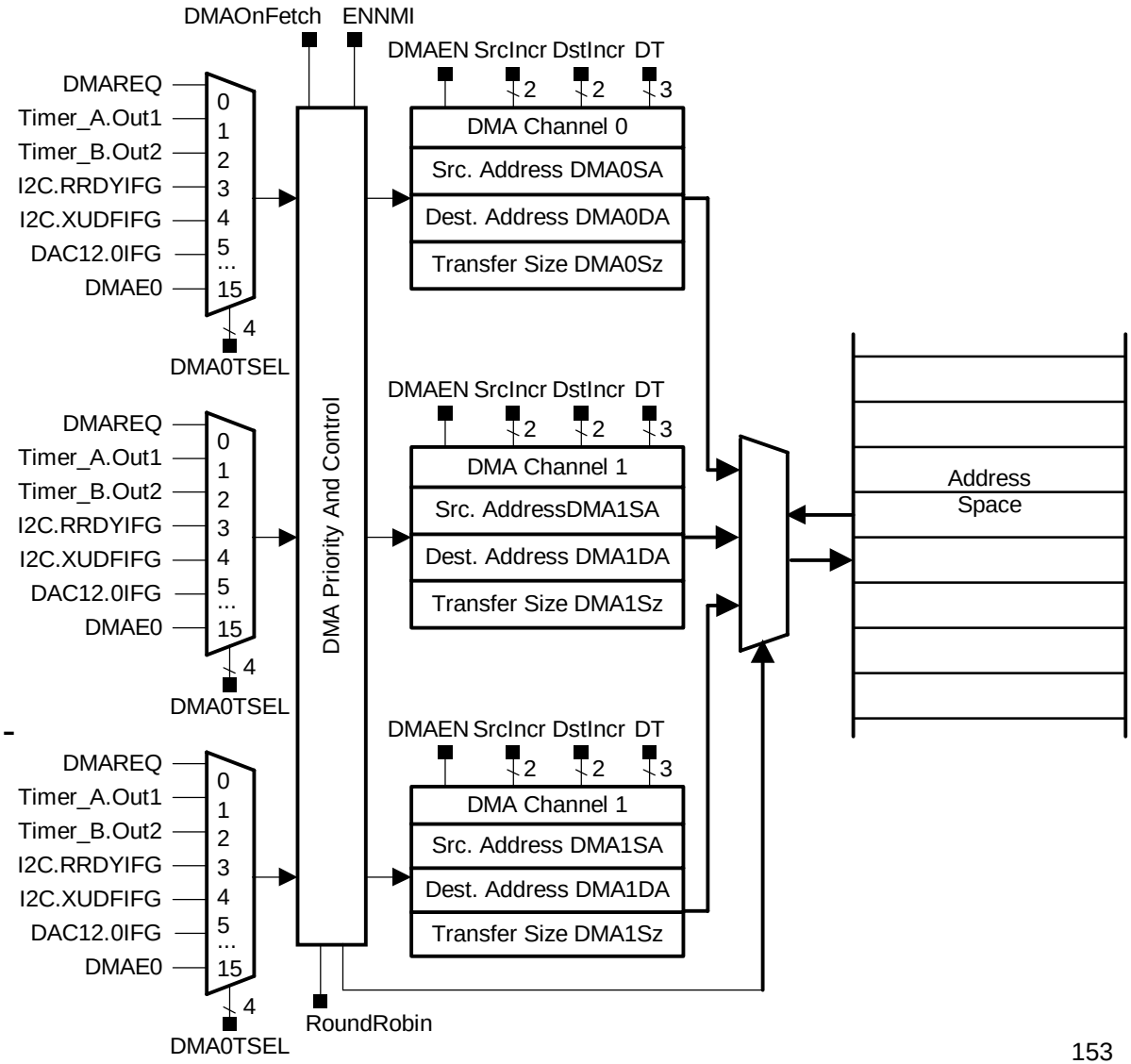
  Med: t_S=15us, I_{DAC}=200uA

  High: t_S=3us, I_{DAC}=700uA
- Code-to-Code settling is 10us, 5us and 2.5us, respectively



DMA Module Overview

- 3 transfer channels
- Configurable transfer trigger selections
- Configurable channel priority
- Configurable block size
- Single, block, or burst transfer modes
- Requires two MCLK cycles per transfer
- Byte or word transfer
- Selectable edge or level-triggered transfer



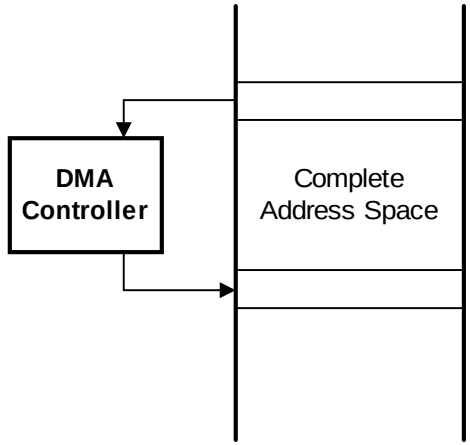
DMA Addressing & Transfer Modes

Addressing

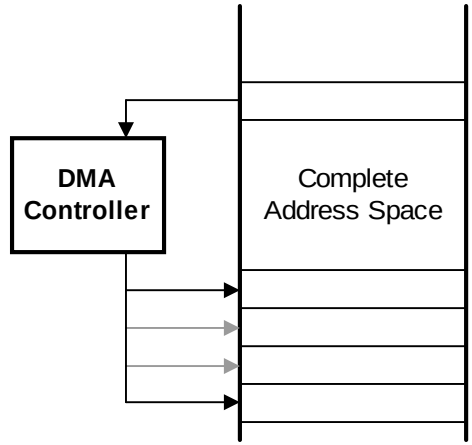
- Fixed address to fixed address
- Fixed address to block of addresses
- Block of addresses to fixed address
- Block of addresses to block of addresses

Transfer

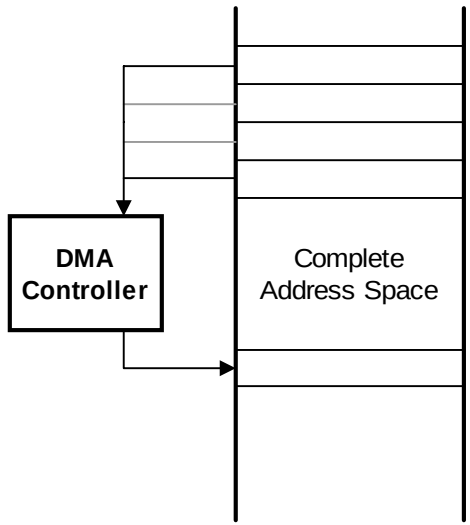
- Single transfer
- Block transfer
- Burst transfers
- Repeated single transfers
- Repeated block transfers
- Repeated burst transfers



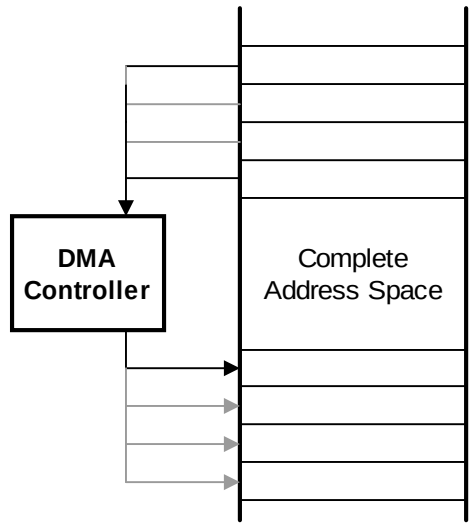
Fixed Address to Fixed Address



Fixed address to block addresses



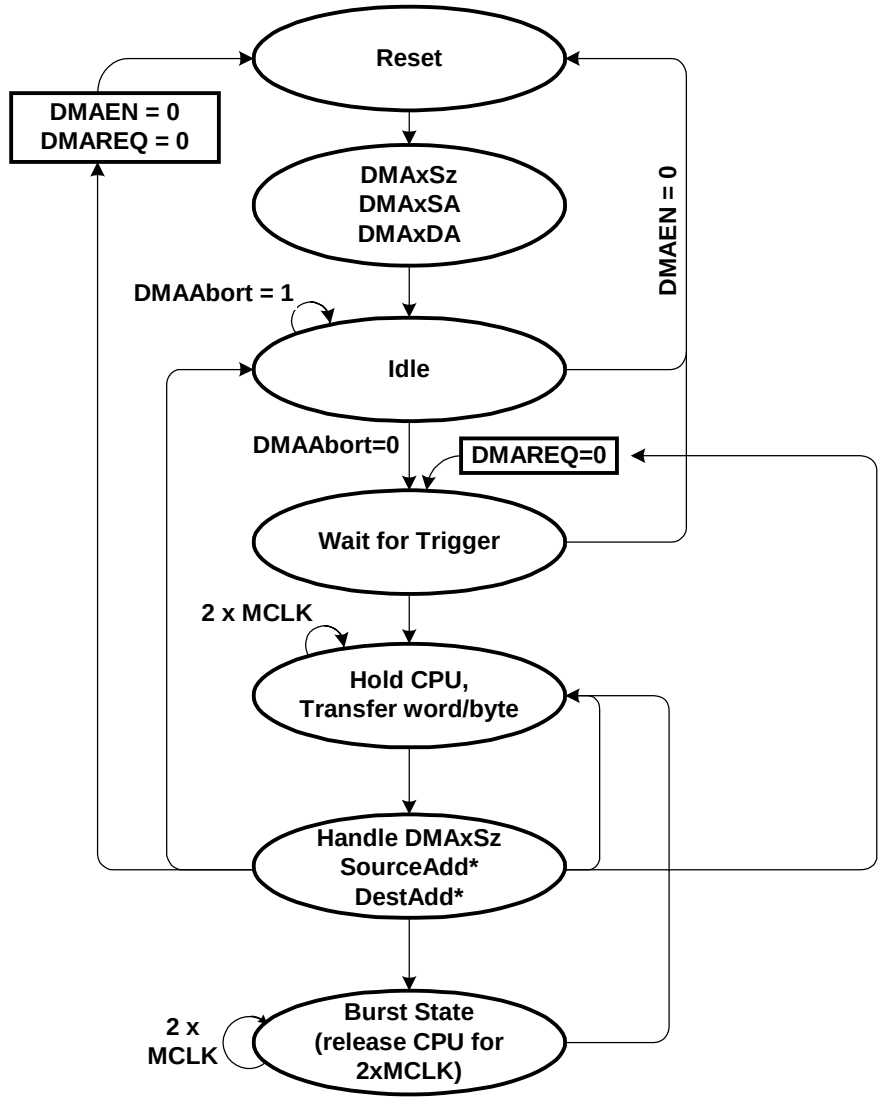
Block Addresses to Fixed Address



Block Addresses to Block Addresses

DMA Operational Flow

- Size, destination & source registers handled automatically by the DMA module
- Address registers are stored temporarily for Inc/Dec operations
- DMA can be disabled or re-enabled after each transfer depending on mode selected
- Single transfers require individual triggers to initiate
- Block mode halts CPU for entire transfer of DMAxSz words/bytes
- Burst mode releases CPU for 2 MCLKs after every 4 word/byte transfers



Starting & Stopping DMA Transfers

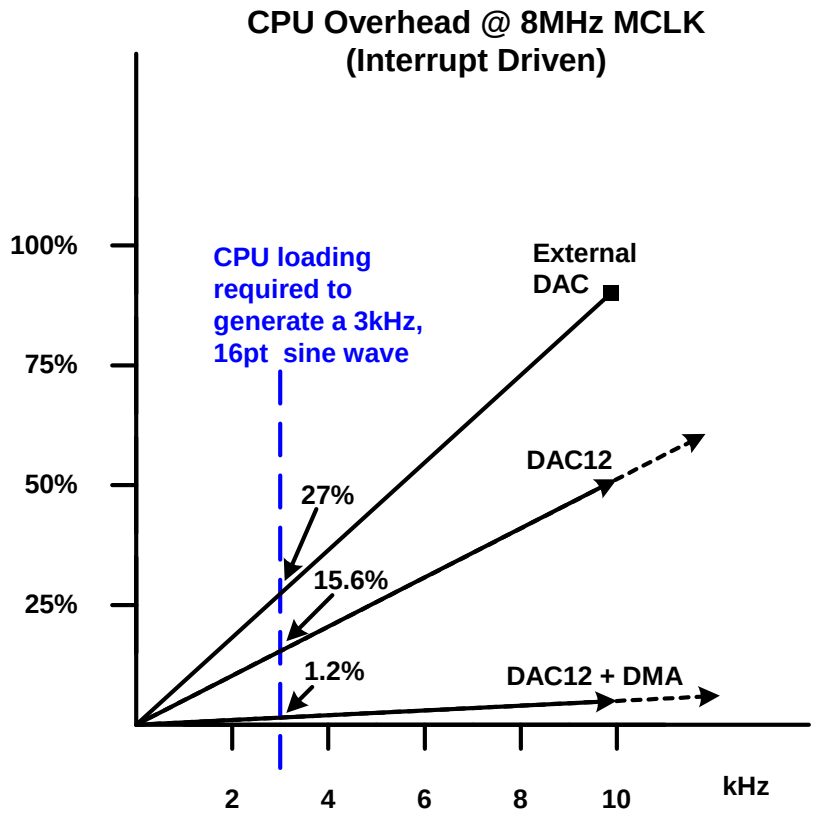
- DMA transfers can be triggered on:
  - DMAREQ bit set
  - Timer_A3 CCTL2 CCIFG set
  - Timer_B7 CCTL2 CCIFG set
  - I2C receiver has new data
  - I2C transmitter requests new data
  - DAC12IFG is set
  - External trigger - DMAE0
- In-progress DMA transfers are halted by:
  - NMI interrupt (ENNMI = set)
  - Clearing of DMAEN (burst mode only)

Interrupts and the DMA

- Other ISRs are interrupted by a DMA transfer
- DMA transfers are NOT interrupted by system INTs (except NMI case above)

CPU Overhead and the DMA

- Comparison of a typical serial, 12-bit DAC, DAC12, and DAC12 + DMA
- DAC12 offers nearly 2x performance increase over external DAC
- DAC12 + DMA provides >20x performance increase
- External DAC reaches max frequency limit at <100% CPU loading due to data I/F overhead



Low Power Modes and the DMA

- MSP430 all LPM's = MCLK Off
- MCLK restored automatically when a DMA trigger is received
- DMAonFetch

  When enabled, the DMA will wait until the next CPU instruction fetch to perform transfer when triggered

  Eliminates low power mode exit for DMA transfer alone
- When DCO = MCLK

  LPM0/1 = DCO always on

  LPM3/4 = DCO disabled: <6us latency to initiate a DMA transfer required to power-up DCO
- DCO = Default DMA clock if MCLK is not equal to the DCO and MCLK is not present

DAC12 Channel Grouping

- DAC_0 thru n-1:

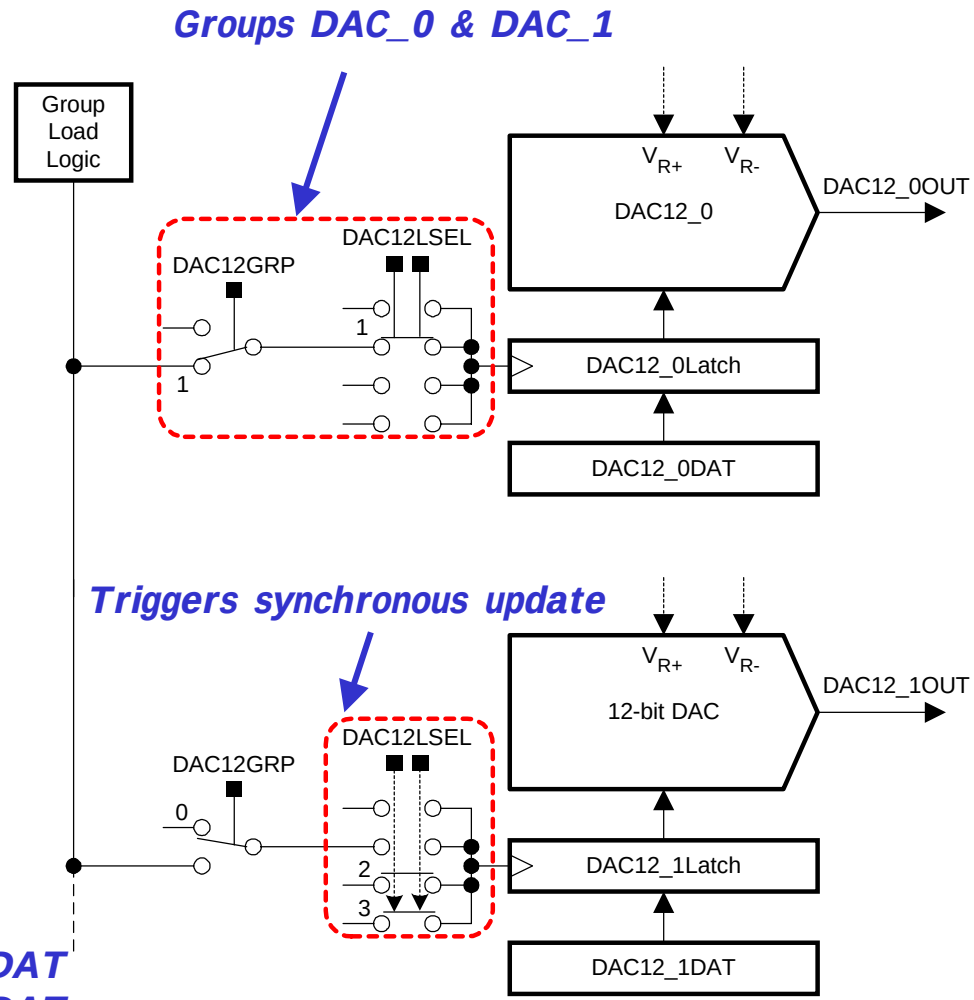
  DAC12GRP & DAC12SEL = 1
- DAC_n:

  DAC12GRP = 0

  DAC12SEL = Group Trigger SRC
- DAC12GRP groups the DAC with the next channel in the module
- Grouped DACs will update synchronously on a trigger
- DAC data latches after:

  All grouped channels have received new data &…

  DAC trigger is received



Example:

- ***Move data to DAC12_0DAT***
- ***Move data to DAC12_1DAT***
- ***Trigger DAC12_1***

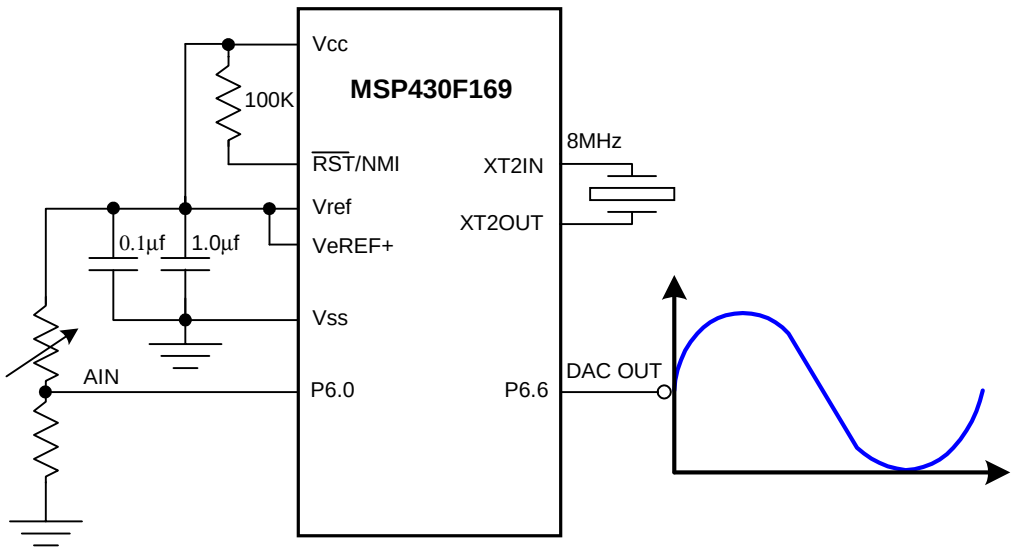
(Data load is independent of order)

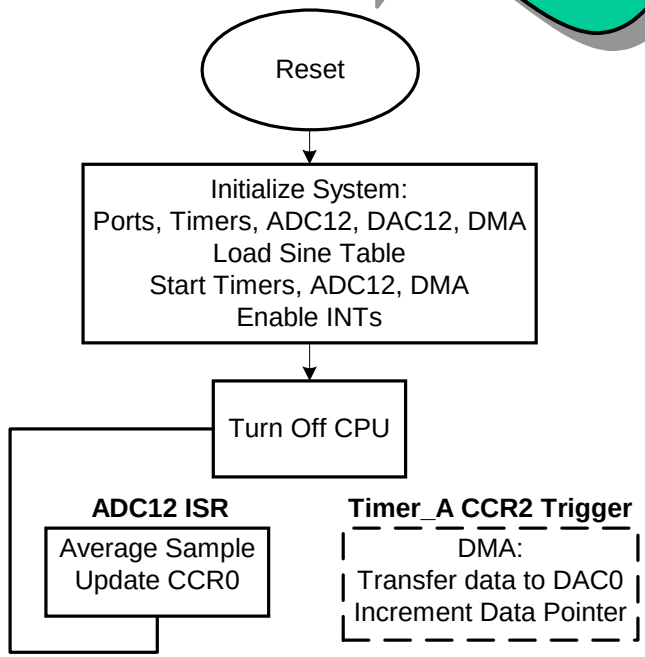
Sine Wave Generation - CPU-Free

Group Demo

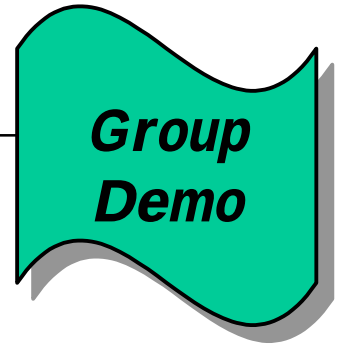
Goal - Output variable sine wave using the 'F169 DAC12 module

- Sample a variable voltage
- Translate to a frequency-dependent sine wave
- Compare External A/D & D/A to 'F169 ADC12 & DAC12 using DMA





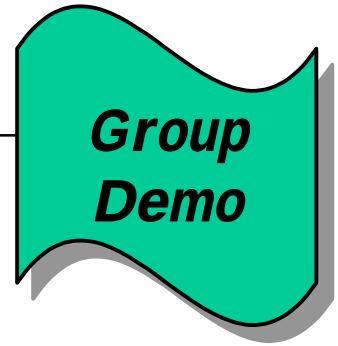
Sine Generation C Code -ADC12 Setup




```
//Setup Timer B
TBCTL = TBSSEL1 + TBCLR;                  // SMCLK, clear TBR
TBCCTL0 = OUTMOD_7;
TBCCR0 = 0x03E8;
//Setup ADC12
P6SEL |= BIT2;                            // P6.2 = ADC CH2
ADC12CTL0 = ADC12ON + SHT0_5;             // ADC12 on, set sampling time
ADC12CTL1 = SHP + SHS_2 + ADC12SSEL_3
            + ADC12DIV_7 + CONSEQ_2;      // Use sampling timer, set mode
ADC12MCTL0 = INCH_2;                      // REF+=AVcc, ADC12MEM0 = A2
ADC12IE = BIT0;                           // Enable ADC12IFG.0
ADC12CTL0 |= ENC;                         // Enable conversions
```

- Timer_B Clock = XT2 = 8MHz
- TBCCR0 = 1000 - TBCCR0 trigger at an 8ksps rate
- ADC12 begins a new sample/convert cycle each TBCCR0 roll-over
- Upon conversion completion, the CPU executes ADC12 ISR, averages 8 samples and updates Timer_A CCR0

Sine Generation C Code - DAC12 Setup



- P6.6 configured for DAC0 output
- DAC12 -VeREF+, 12-bit Mode, Fast Settling
- Data written to DAC12_0DAT auto updates R-string

```
//Setup DAC12 Ch0
  P6SEL = BIT6;
  P6DIR = BIT6;
  DAC12_0CTL = DAC12SREF_3 + DAC12IR  // DAC_0 REF = VeRef+,
               + DAC12AMP_7;          // fast settling
```

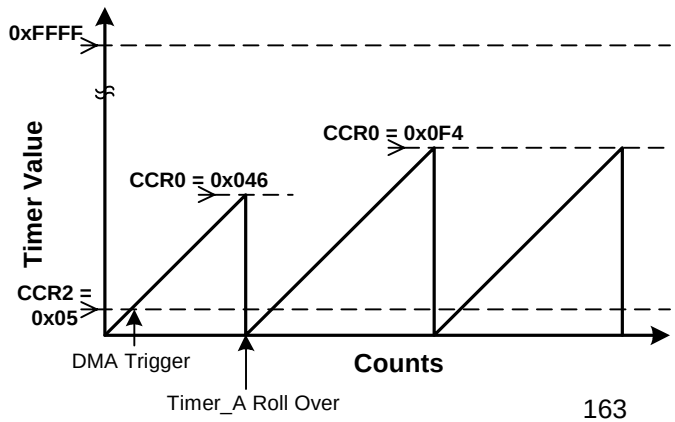
```
// Data Table for Sine Out
const unsigned int
DAC_Table[16]=
{   0x0000,
    0x009A,
    0x0256,
    0x04EF,
    0x07FE,
    0x0B0E,
    0x0DA7,
    0x0F63,
    0x0FFF,
    0x0F63,
    0x0DA7,
    0x0B0E,
    0x07FF,
    0x04EF,
    0x0256,
    0x009A,
};
```

Sine Generation C Code: DMA Setup

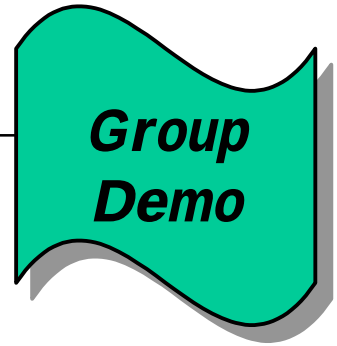
Group Demo

- Timer_A CCR2 triggers DMA0
- DMA0 moves data from sine table with auto-increment source address, single word transfer mode to DAC12_0

```
//Setup Timer A
TACTL = TASSEL1 + TACLR;                          // SMCLK, clear TAR
CCTL2 = OUTMOD_0;                                 // Toggle Mode
CCR2 = 0x00;
//Setup DMA to transfer Sine data
DMACTL0 = DMA0TSEL_1;                             // TACCTL2 CCIFG trigger
DMA0CTL = DMASRCINCR_3 + DMADT_4  + DMAEN;        // Incr SRC, HW trig/xfer,
DMA0SA = (int) DAC_Table;                         // Pointer to Sine data
DMA0DA = 0x01C8;                                  // Destination Address: DAC0
DMA0SZ = 0x010;                                   // 16 words to transfer
```



Sine Generation - CPU Resources - ADC



- CPU is off when not handling ADC12 data
- ADC12 ISR execution time:

  Samples 1-7: 21 MCLKs to execute + 11 MCLKs to enter/exit ISR

  Sample 8 : 50 MCLKs to execute + 11 MCLKs to enter/exit ISR
- Total loading due to ADC12 ISR = 0.29MIPs

```
for (;;)
  { _BIS_SR(CPUOFF);                    // CPU off
    _NOP();                             // Required only for C-spy
  }
}

// ADC12 interrupt service routine
interrupt[ADC_VECTOR] void ADC12ISR (void)
{ Result = Result + (ADC12MEM0);        // Accumulate ADC result
  Samples++;                            // Increment sample counter
  if (Samples == 8)                     // Eight samples?
  { Samples = 0;                        // Reset counter
    CCR0 = Result/8;                    // CCR0 = Average Result
    Result = 0;                         // Clear result
  }
}
```

$$\frac{(21+11)\times 7000+(50+11)\times 1000}{1{,}000{,}000} = 0.29\text{Mips}$$

Sine Generation - CPU Resources - DMA & DAC

Group Demo

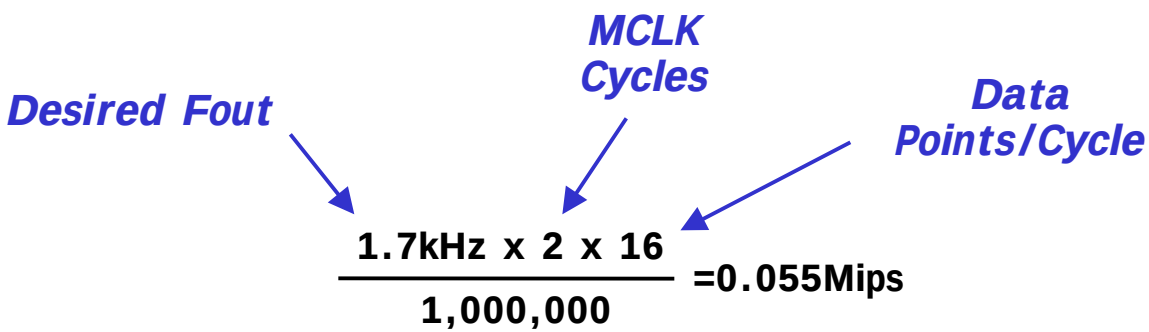
- DAC12 is updated transparent to the CPU

  NO INSTRUCTIONS ARE EXECUTED FOR DAC UPDATES
- CPU "Loading" due to DMA

  Each DMA data transfer takes 2 MCLKs - 1 for data read, 1 for data write

  CPU is halted 2 MCLKs for each DMA transfer

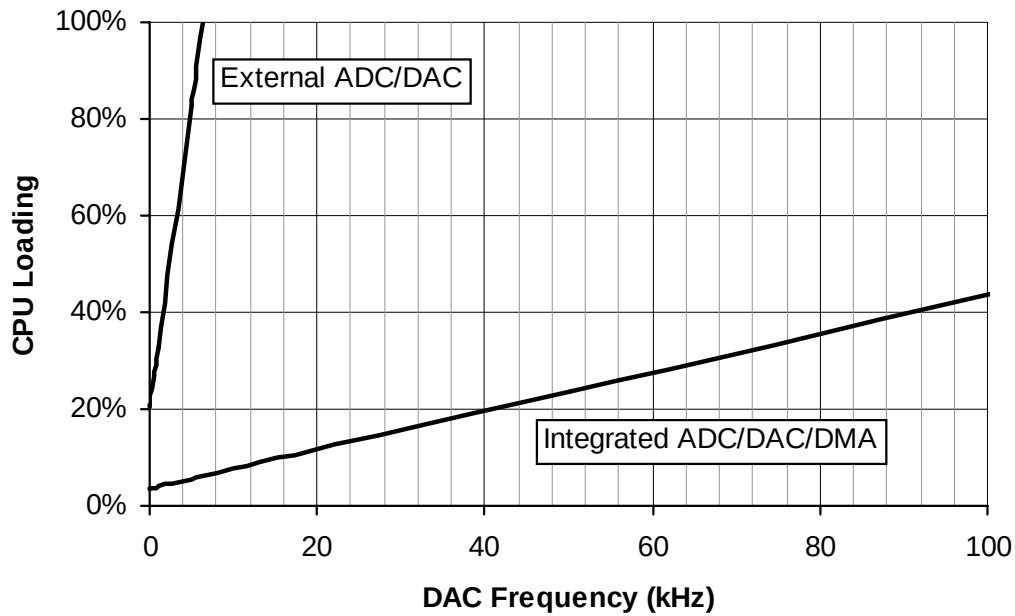
  MAB & MDB control belongs to the DMA module
- Total "loading" due to each DAC12 data update for a given f_{OUT}:



Sine Generation - Total CPU Loading Results

Group Demo

- Typical External ADC/DAC
  - ADC (8ksps): 1.62MIPs
  - DAC (1.7kHz): 1.7MIPs (~62 MCLKs/transfer)
- Integrated ADC12 & DAC12 using the DMA
  - ADC (8ksps): 0.29MIPs
  - DAC (1.7kHz): 0.055MIPs (2 MCLKs/transfer)
- Fewer MCLKs/data fetch
- Data I/O time = fetch MCLKs
- Reduced power



Notes

Agenda - Dallas, November 2002

Fast Track Production

Overview of the different Flash programming options:

- Production programming with the PRGS JTAG programming adapter
- Production programming with a user-developed JTAG device replicator
- Accessing the BSL for production programming

ROM:

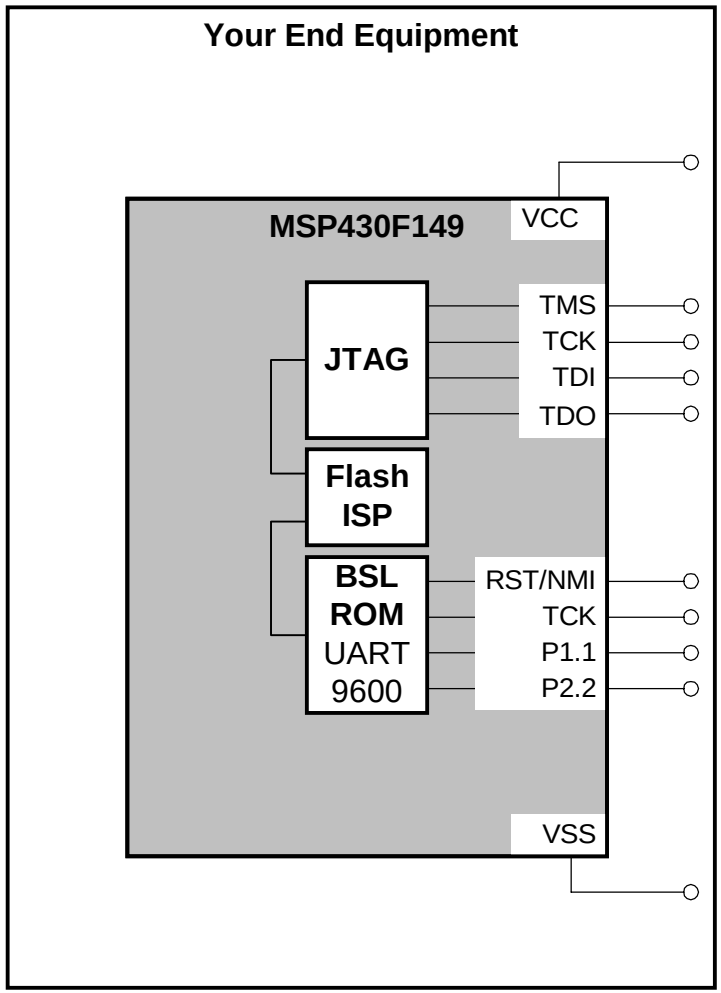
- Devices offered
- ROM process description

System Design and Best Practices:

- TI ESD testing specifications
- PCB layout fundamentals and suggestions
- Solutions for supply voltage brownout

In-Application Flash Production Programming Options

- Fast
- Simple
- Low cost
- Flexible

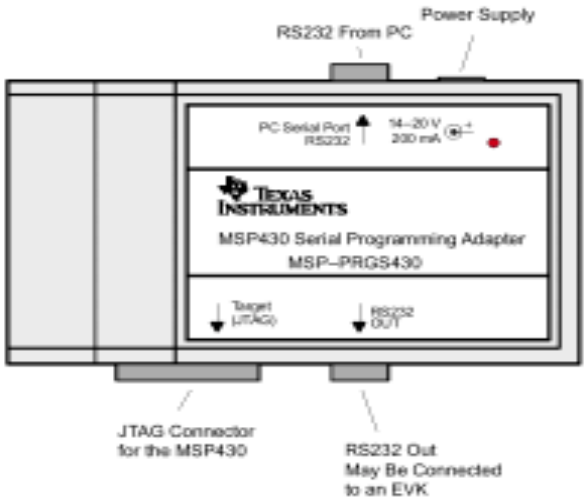


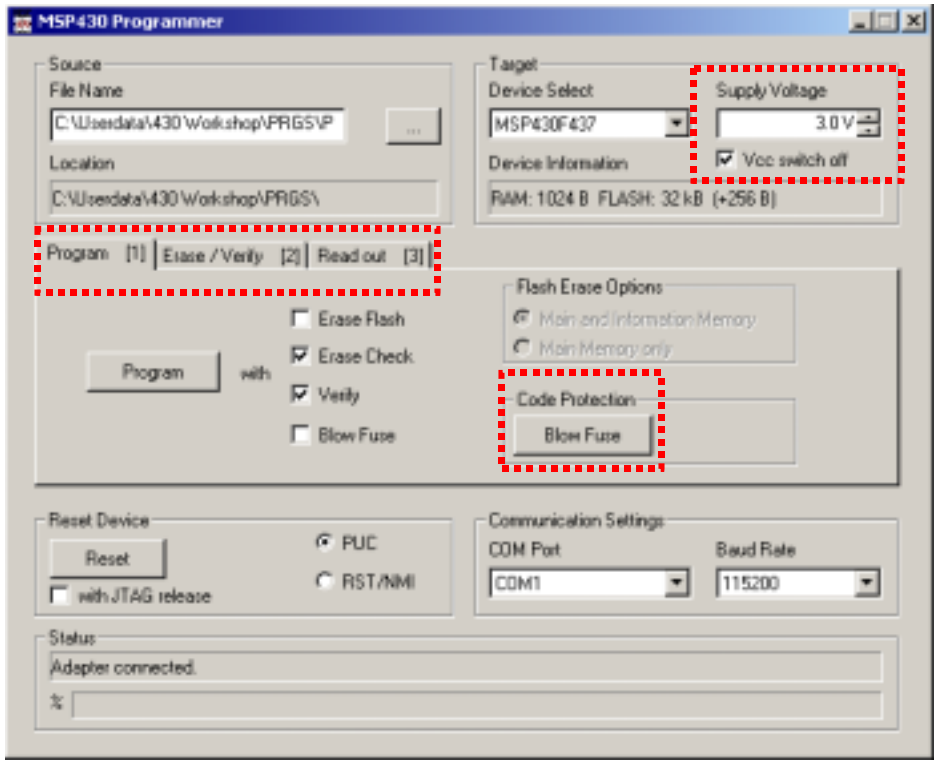
- PRGS / GANG
- Softbaugh Replicator
- Your solution using App. Report SLAA149

- Gessler/Softbaugh
- Your solution using App. Report SLAA096 and SLAA089

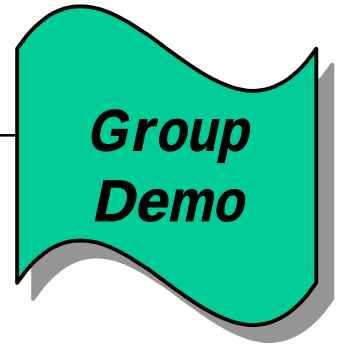
PRGS Description

- Uses JTAG in or out of application
- Programs single unit at a time
- Erase, Program, Verify, Read-Out ~58s for 60kBytes (1.0 kByte/s)
- Requires a PC via RS-232
- GUI and /or DLL
- User's Guide: "MSP430 Family Serial Programming Adapter Manual" SLAU048B





Available through TI and Distribution



Readout Attendee's D437 With PRGS

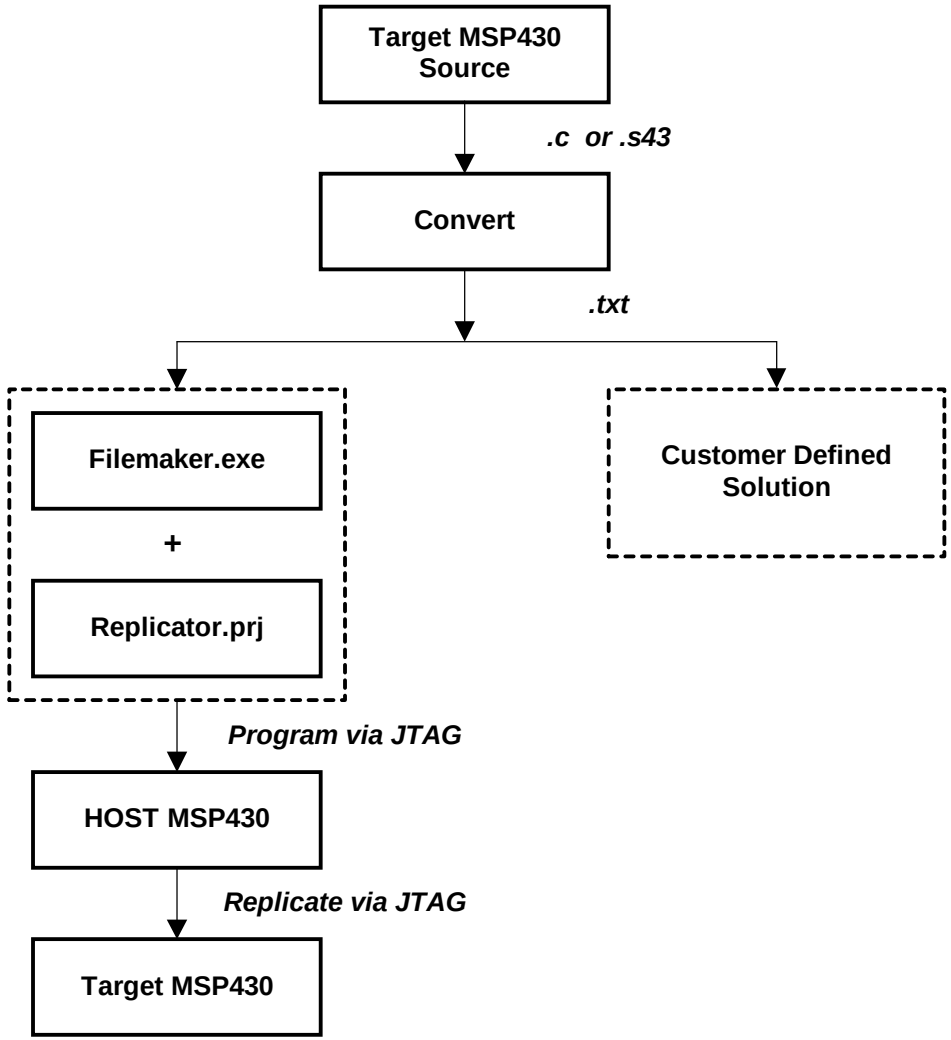
MSP430 JTAG Commands For Programming

Macro Name	Function
IR_SHIFT(8-bit Instruction)	Shifts an 8-bit JTAG instruction into the JTAG instruction register. At the same time, the 8-bit value is shifted out through TDO.
DR_SHIFT16(16-bit Data)	Shifts a 16-bit data word into a JTAG data register. At the same time, the 16-bit value is shifted out through TDO.
Delay (time)	Wait for the specified time in ms
SetTCLK	Set TCLK to 1
ClrTCLK	Set TCLK to 0
TDOvalue	Variable containing the last value shifted out on TDO

Instruction Name	8-Bit Instruction Value (Hex)
Controlling the MAB (Memory Address Bus)	
IR_ADDR_16BIT	0x83
IR_ADDR_CAPTURE	0x84
Controlling the MDB (Memory Data Bus)	
IR_DATA_TO_ADDR	0x85
IR_DATA_16BIT	0x41
IR_DATA_QUICK	0x43
IR_BYPASS	0xFF
Controlling the CPU	
IR_CNTRL_SIG_16BIT	0x13
IR_CNTRL_SIG_CAPTURE	0x14
IR_CNTRL_SIG_RELEASE	0x15
Memory Verification (via Signature Analysis)	
IR_DATA_PSA	0x44
IR_SHIFT_OUT_PSA	0x46
Fuse Programming	
IR_Prepare_Blow	0x22
IR_Ex_Blow	0x24

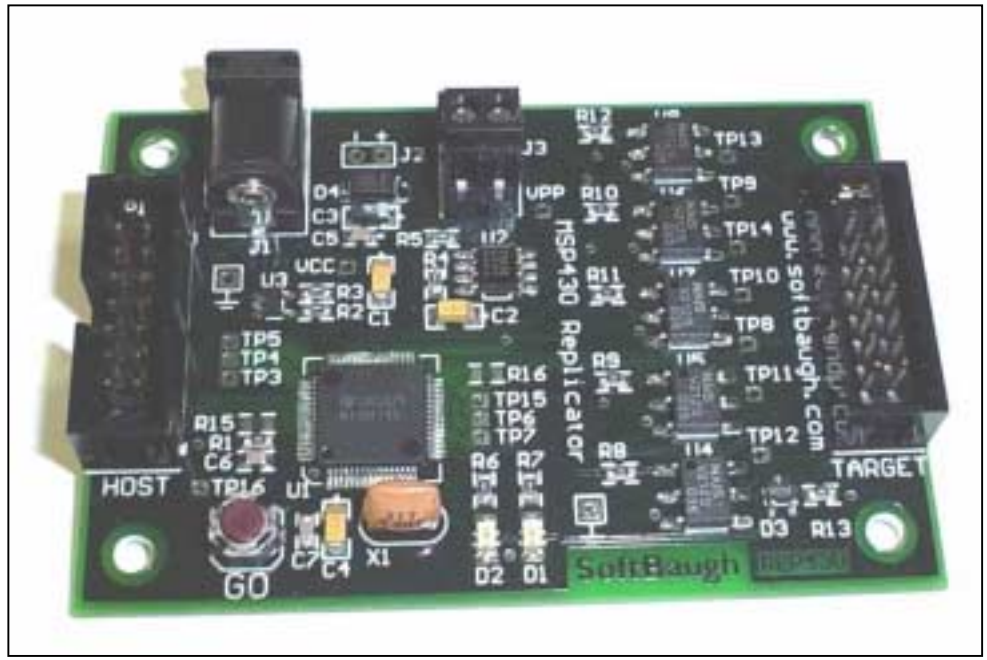
Building A Device Replicator Using JTAG

- App Report: “Programming a Flash-based MSP430 Using the JTAG Interface” - SLAA149
- System description
- Host Software in C
- A proven “Replicator” schematic
- Filemaker.exe
- Complete hardware solution available from Softbaugh

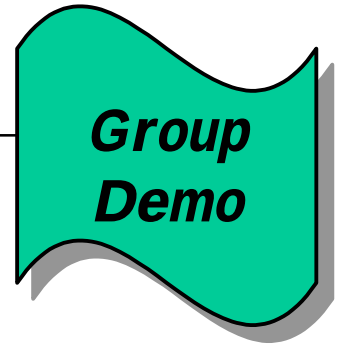


An MSP430 “Replicator” Spec

- HW as described in “Programming a Flash-based MSP430 Using the JTAG Interface” available from SoftBaugh
- Standalone production Programmer
- Compatible with all MSP430 Flash-based devices
- Maximum target device program code size: ~57 KB
- Programming speed (Erase, Program, Verify): ~8 KB in 1.5 sec, 48 KB in 8 sec
- Fast verify and erase check: ~17 KB/10 ms

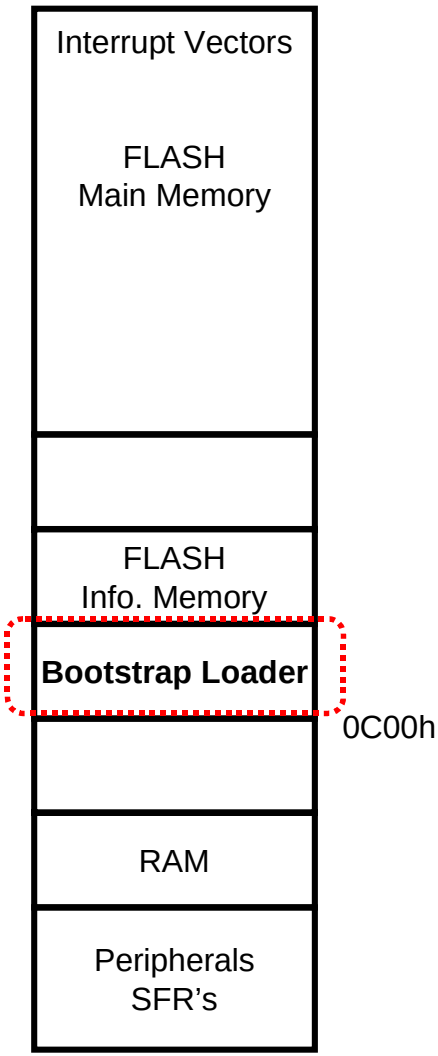


Replicator

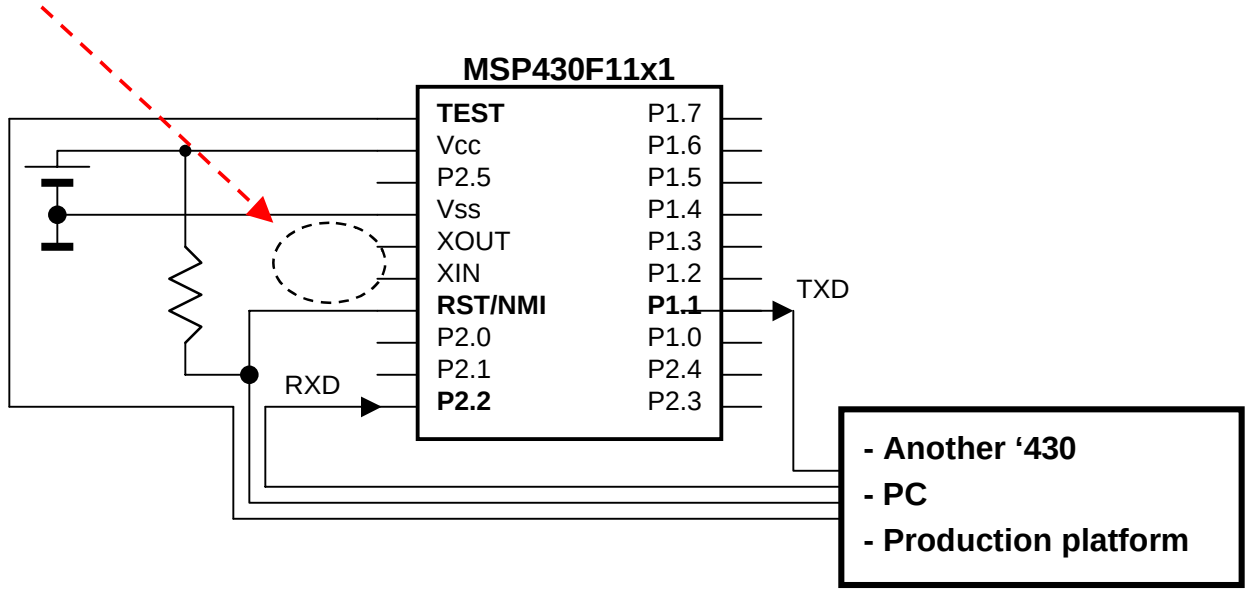


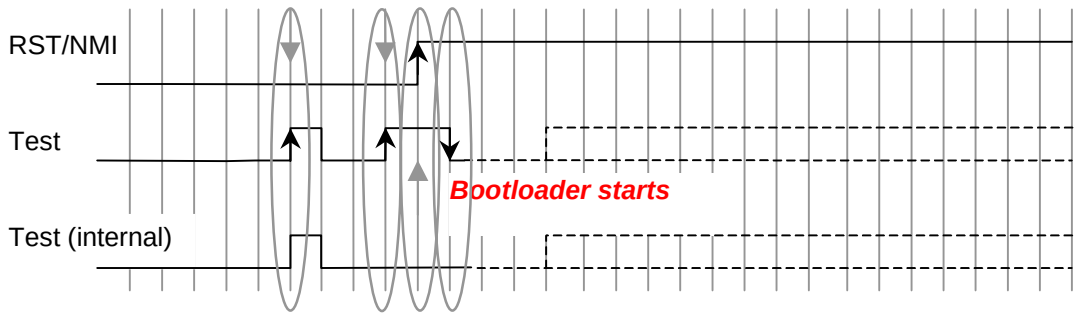
Program attendee's D437 with Replicator

MSP430x11x Flash BSL Overview



- 9600 "auto baud" UART
- No XTAL or *any XTAL* can be in place





Flash BSL Function Summary

- Programs single units in application
- Uses 9600 / 38400 baud UART protocol 8E1
- Erase, Program, Verify, Read-Out
- V1.60: 20s for 60kBytes (3.00kByte/s)
- App Reports “Application of Bootstrap Loader in MSP430 w/Flash Hardware, Software Proposal” (SLAA096B) and“Features of the MSP430 Bootstrap Loader” (SLAA089A)
- PC-based solutions available from SoftBaugh & Gessler

Flash BSL Functions & Password

Password Protected Functions

- Receive data block to program flash memory, RAM, or peripherals
- Transmit data block
- Erase segment
- Erase check
- Load program counter and start user program

Unprotected Functions

- *Receive password*
- *Mass erase*
- *Transmit BSL version*
- *Change baud rate*

Max Code Crack Duration:

~ 134e66 years @ 9600 baud
~ 33e66 years @ 38400 baud

Code Specific BSL Password = 16, 16-bit Interrupt Vectors:

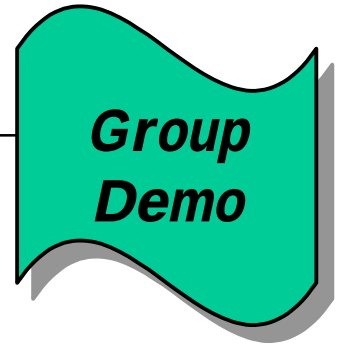
$$2^{256}\ \text{combinations} \times \frac{44\ \text{bytes} \times 8\ \text{bits}}{9600\ \text{baud}} \approx 4.245\text{e}75\ \text{seconds}$$

BSL Command Structure

BSL Command	HDR	CMD	L1	L2	AL	AH	LL	LH	D1	D2…Dn	CKL	CKH	ACK
RX Data Block	0x80	0x12	n	n	AL	AH	n-4	0x00	D1	D2…Dn-4	CKL	CKH	ACK
RX Password	0x80	0x10	24	24	xx	xx	xx	xx	D1	D2…D20	CKL	CKH	ACK
Erase Segment	0x80	0x16	4	4	AL	AH	0x02	0xA5	-	---	CKL	CKH	ACK
Mass Erase	0x80	0x18	4	4	xx	xx	xx	xx	-	---	CKL	CKH	ACK
Erase Check	0x80	0x1C	4	4	AL	AH	LL	LH	-	---	CKL	CKH	ACK
Chg Baud Rate	0x80	0x20	4	4	D1	D2	D3	xx	-	---	CKL	CKH	ACK
Load PC	0x80	0x1A	4	4	AL	AH	xx	xx	-	---	CKL	CKH	ACK
TX Data Block	0x80	0x14	4	4	AL	AH	n	0x00	-	---	CKL	CKH	-
BSL Response	0x80	xx	n	n	D1	D2…	…	…	…	…Dn	CKL	CKH	-
TX BSL Version	0x80	0x1E	4	4	xx	xx	xx	xx	-	---	CKL	CKH	-
BSL Response	0x80	xx	10	10	D1	D2…	…	…	…	…D10	CKL	CKH	-

- All transfers begin with header - 0x80
- L1 = L2 = Length number of bytes to RX/TX - AL thru Dn
- L1 < 255, and must be even
- 0xAH|AL = start address
- 0xLH|LL = # data bytes(<=250) or erase check size (<=0xFFFF)
- 0xCKH|CKL = Checksum

BSL

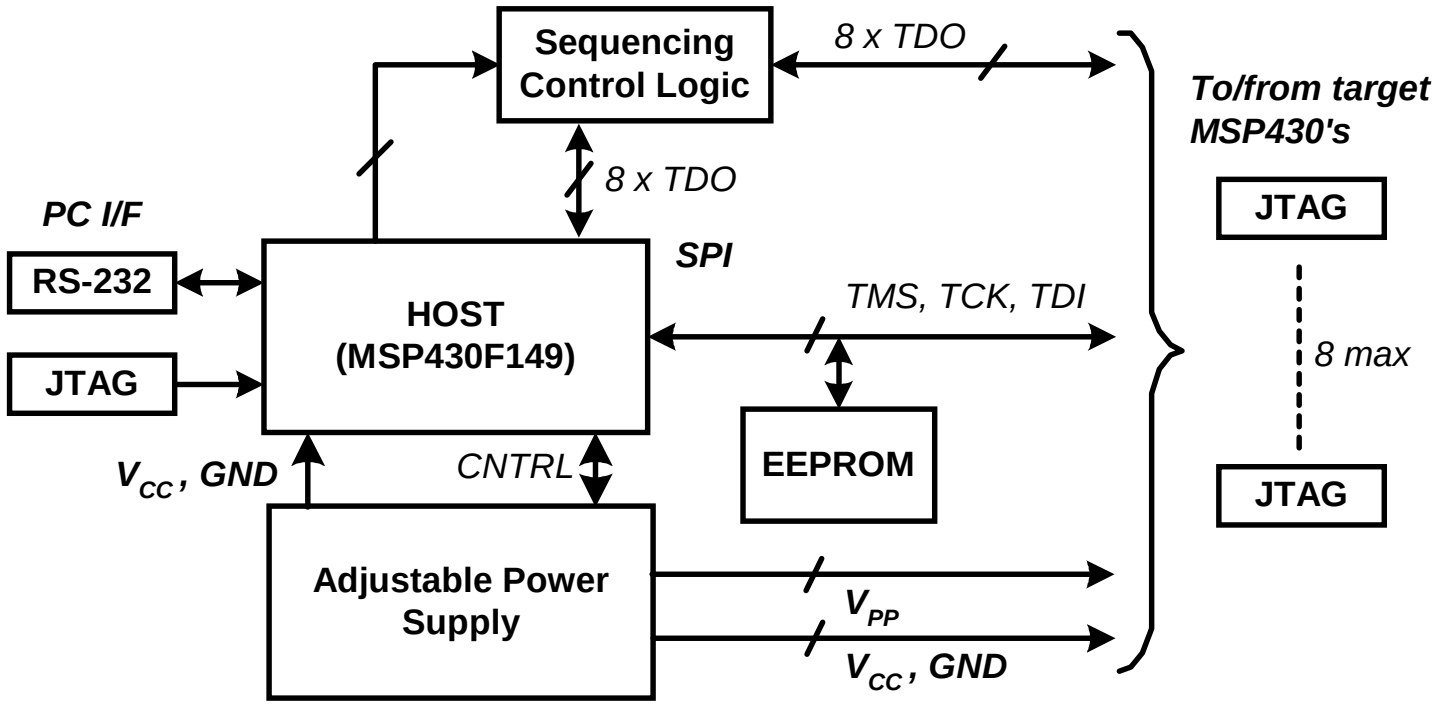


Re-program attendee's D437 with the BSL

MSP430 Gang Programmer Concept

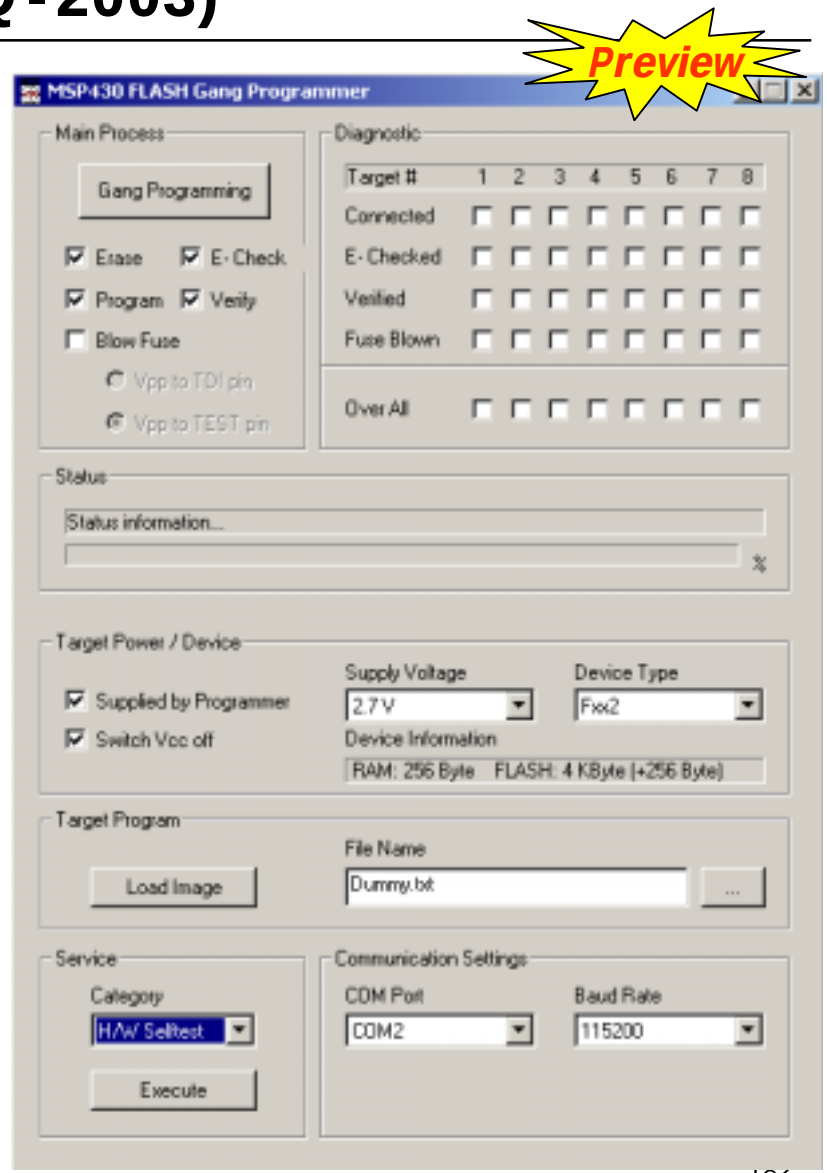
Preview

- Parallel flash programming
- Serial verify & fuse-blow
- Non-volatile memory for standalone programming



MSP430 Gang Programmer (2Q-2003)

- Extension of Replicator Concept
- Programs up to 8 targets simultaneously
- Stand-alone or controlled via RS232-Port
- ~5kB/sec (60kB in 12sec)
- Storage capability for target code: 4MB
- Firmware can be updated via the RS232-Port
- Multiple programmers can be operated in parallel
- Supports programming of the security fuse

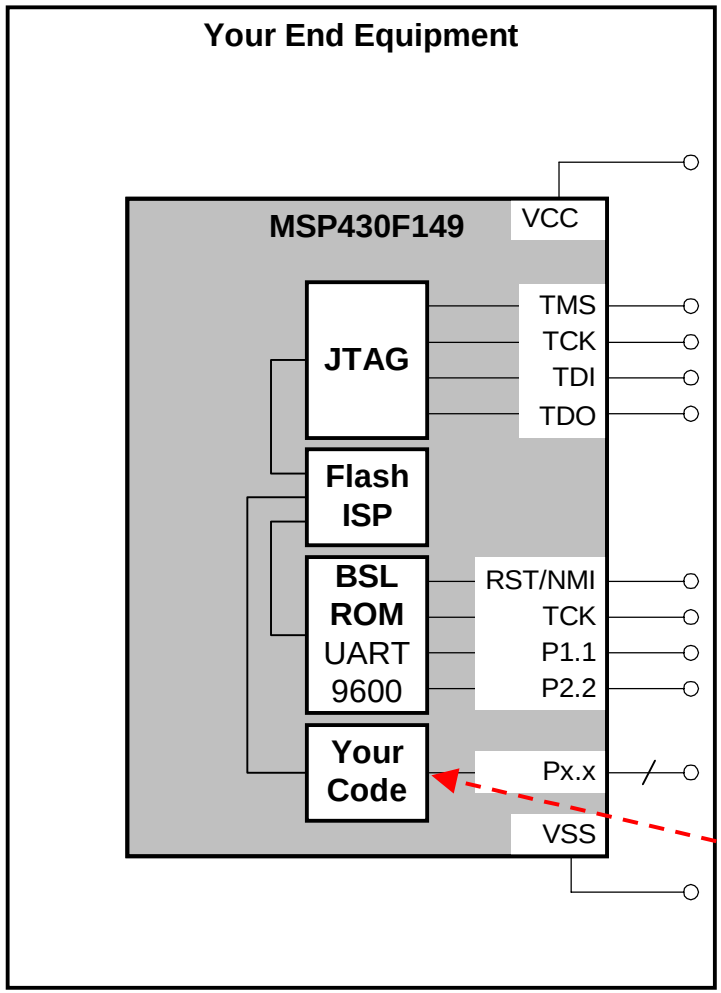


Gang Programmer

Gang Programmer Demo

Flash Field Upgrades

- Fast
- Simple
- Low cost
- Flexible

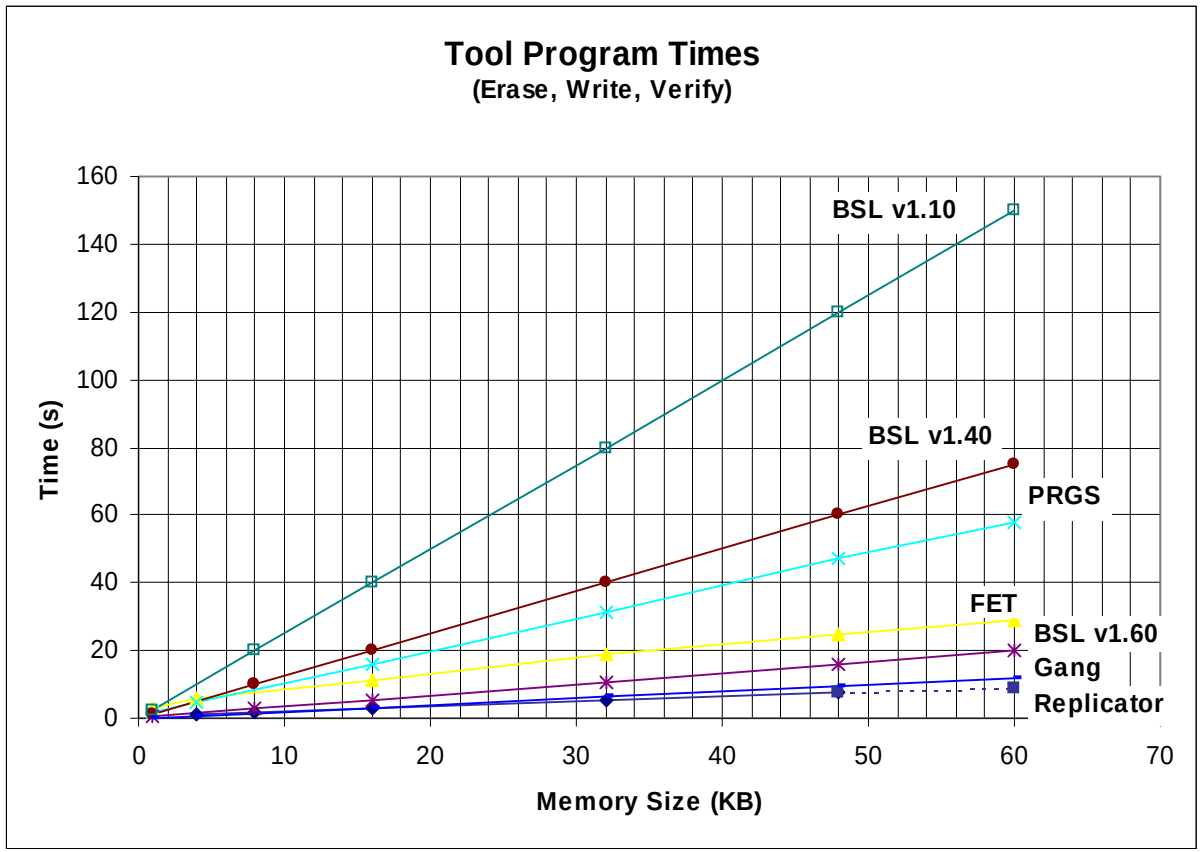


- PRGS / GANG
- Softbaugh Replicator
- Your solution using App. Report SLAA149

- Gessler/Softbaugh
- Your solution using App. Report SLAA096 and SLAA089

Flash Programmer Summary Q & A

- Why is the FET not recommended for production?
- What tools are available to aid development of a custom solution?
- I want to use the BSL in production, how do I protect my IP?



ROM Option

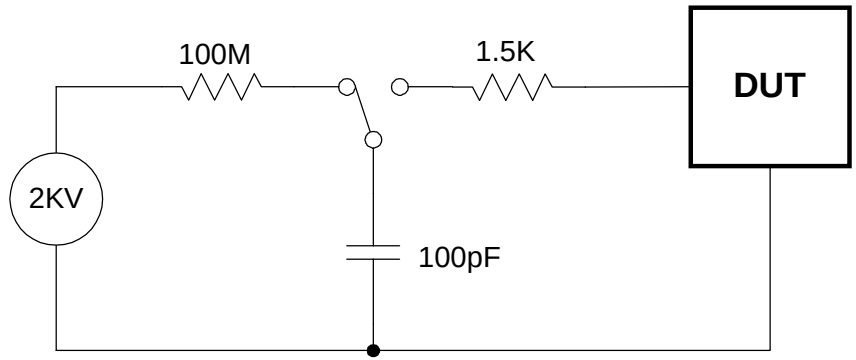
- Factory masked ROM is available for the following device families:
  - MSP430C11x1
  - MSP430C13x1
  - MSP430C31x
  - MSP430C32x
  - MSP430C33x
  - MSP43C41x
- 10-week lead-time for first samples
- 50ku / year minimum volume
- $3k USD NRE

Production Checklist

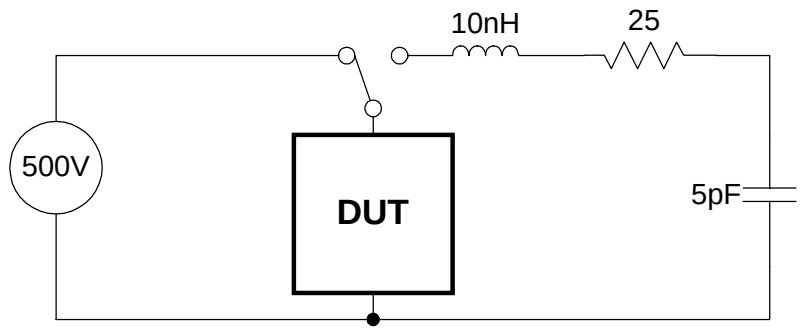
- Is the supply voltage glitch-free and within the specified device limits
- Is an initial software delay provided for the DCO and FLL to settle
- Is the watchdog timer disabled or configured properly for the application
- Is the crystal can grounded
- Are Dvcc and Avcc at the same potential
- No signal paths are passing near the crystal
- Have the recommended ESD handling guidelines been followed
- Are all unused I/O pins configured as outputs and not connected
- Are all JTAG and Test (20 and 28-pin devices) pin left unconnected
- On a ‘4xx device, have the OSC_CAP internal capacitors been enabled
- If an HF-XTAL is used, has the proper turn on sequence been followed
- Has the USART been configured properly

TI Device-Level ESD Specification

- 2KV Human Body Model

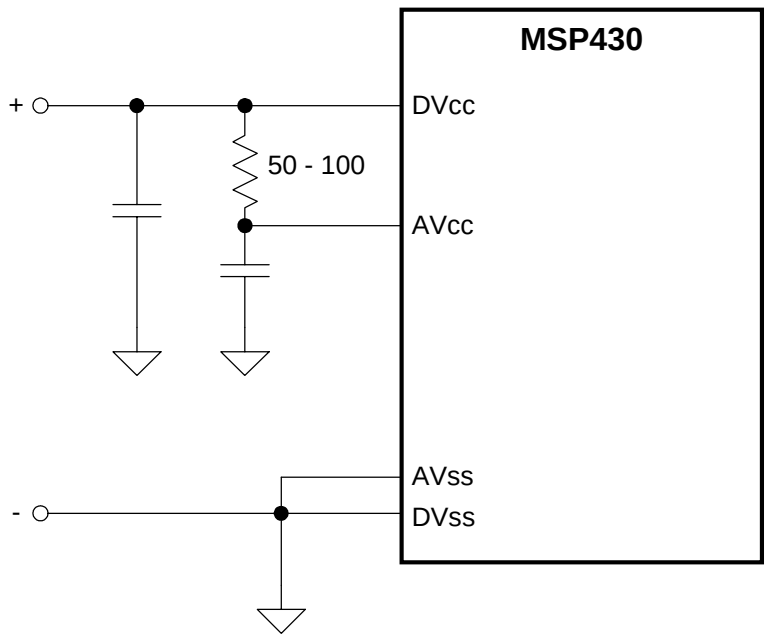


- 500V Charged Device Model



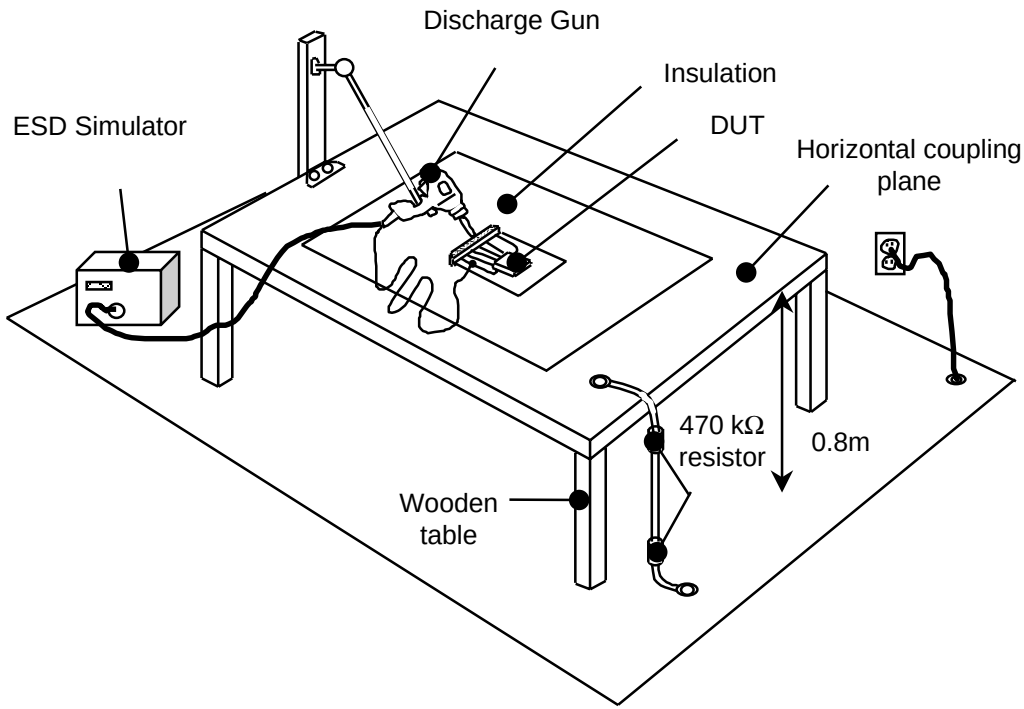
Proper Supply Pin Configuration

- AVcc and DVcc connected internally by diodes
- DVcc - AVcc << 0.3V - always! - **do not power down Avcc**
- DO NOT power down DVcc and AVcc separately
- AVcc must not come up before DVcc
- Avss and DVss connected internally - always connect them on your board



System-Level ESD Testing

- Your system-level ESD requirements can sometimes reach as high as 15KV (IEC61000-4-2).
- System-level specification - not device level specification.



Decoupling Capacitors

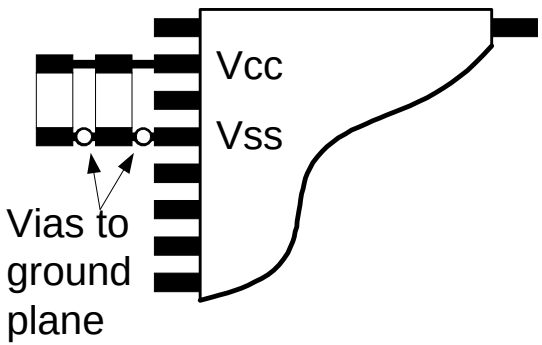
- Most common is .1uF - often, 0.01uF more effective against ESD. Combinations of values are commonly used.
- Lead length is critical because of inductance

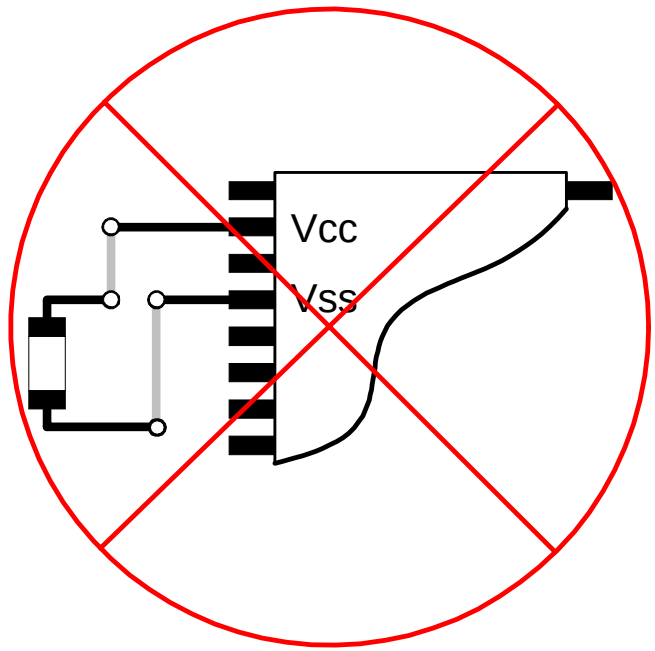
  V=L*di/dt

  L for leads and PCB = 20nh/in

  ESD hits can induce di/dt of 10a/500ps

  V = 400V/in

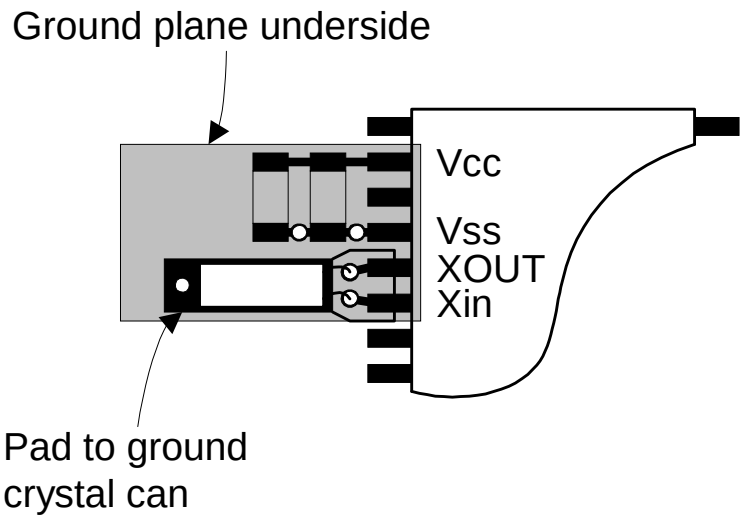


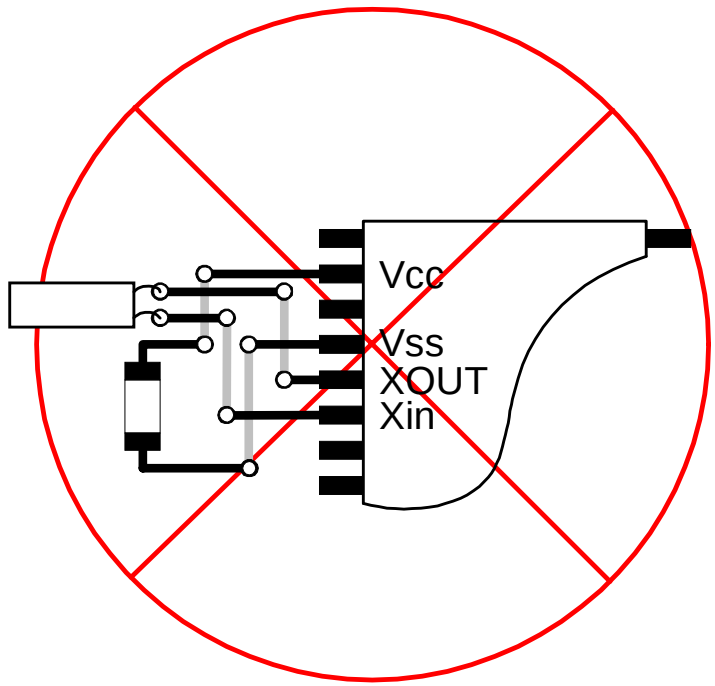


Keep traces/leads absolutely as short as possible. Also keep supply lines short because inductance can cause overshoot.

PCB Layout Fundamentals - Crystal Critical!

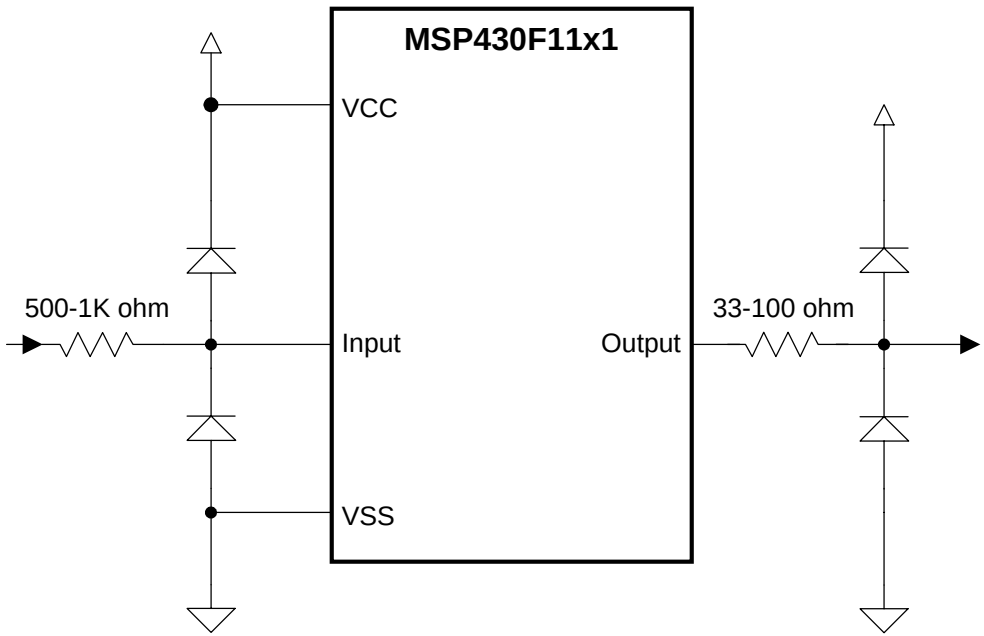
- Keep crystal lines short - less than 1cm - remember length = inductance
- Ground crystal can
- Ground ring around crystal lines
- No traces under crystal
- Ground plane underneath crystal





External ESD Suppression

- Series R most basic. Also helps reduce Vcc ringing at power-up (inductance) - can also be combined with diodes.
- Suppression devices such as varistors, thyristors, TVS diodes, etc. should be used in extreme cases (www.littlefuse.com).

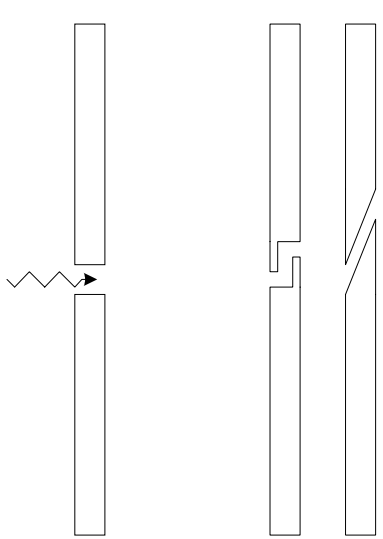


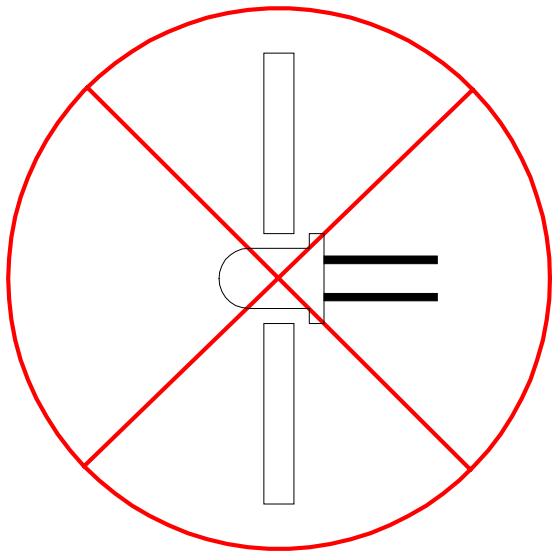
PCB Layout Fundamentals

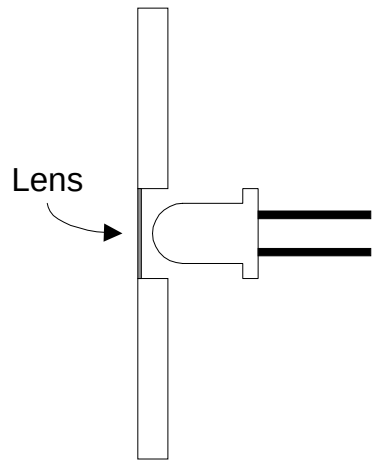
- Use ground plane where possible to lower current-path inductance
- Properly terminate all unused ‘430 pins - See user’s guide
- No floating copper islands on the PCB - they can induce noise and arc in the presence of ESD
- Avoid crossing breaks in the ground plane with traces - this increases loop inductance
- Keep the ‘430 out of the path of ESD

Enclosure Openings - Close Them!

- No direct openings or keep PCB away from openings
- Use gasket around LCD opening
- LEDs are particularly vulnerable - direct path to PC board

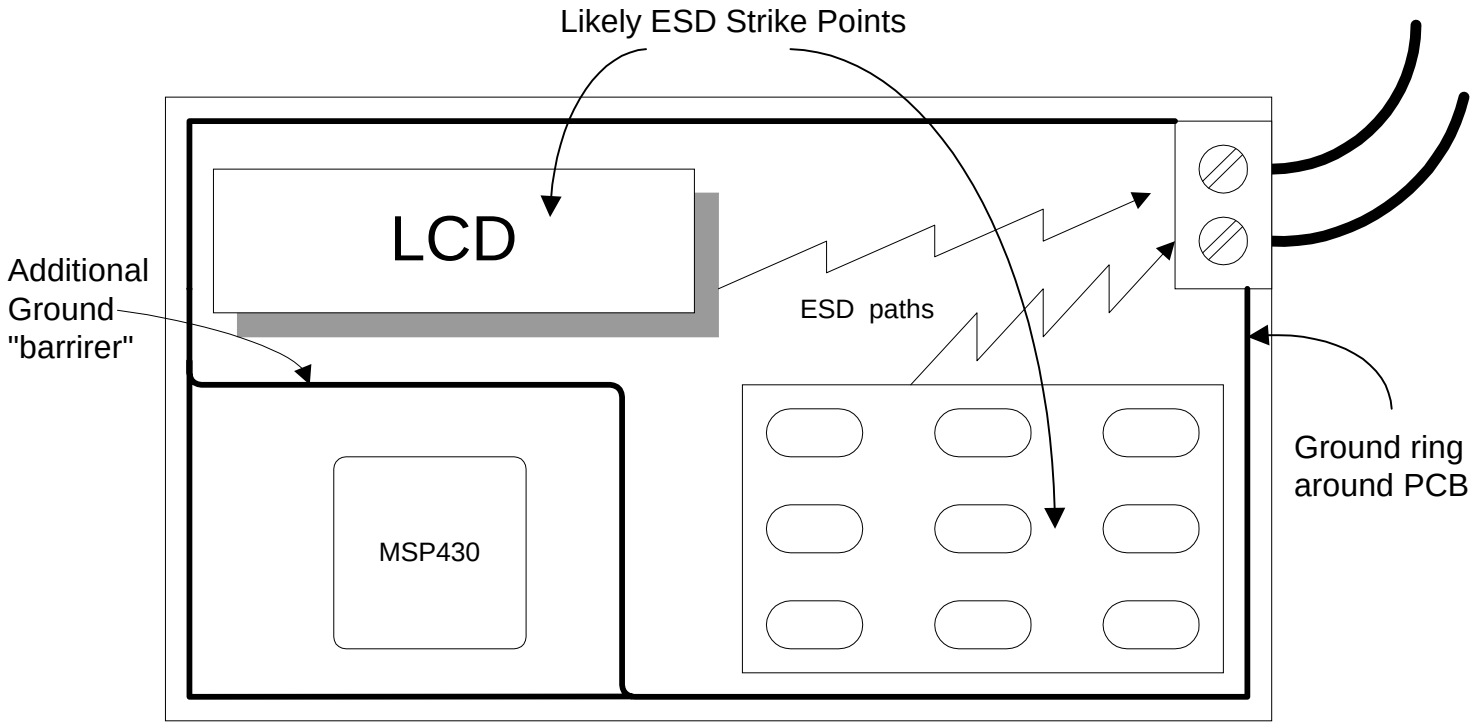




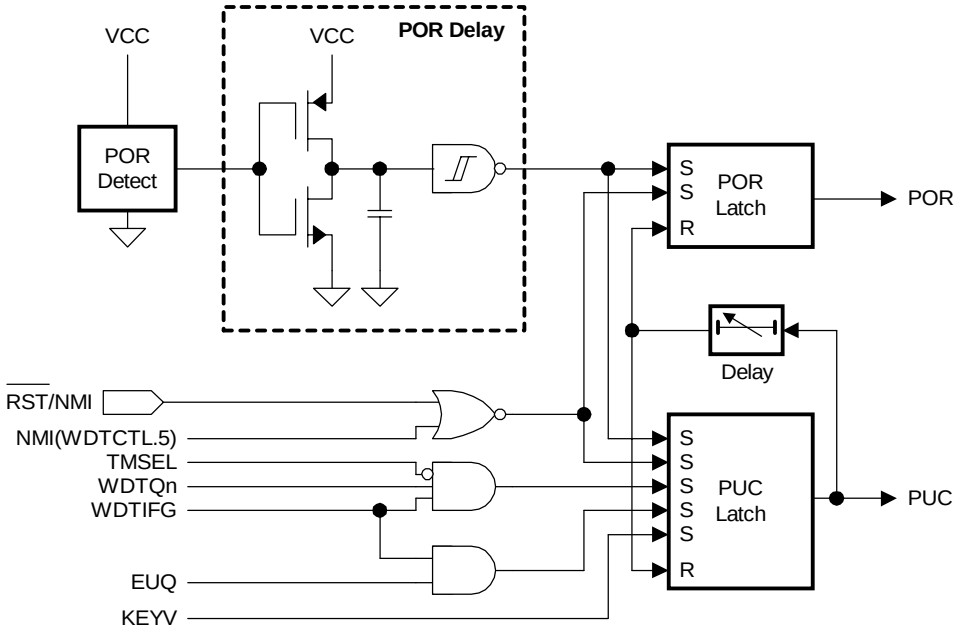


Enclosures - Grounding Paths

- Provide ESD a path to ground
- Keep MSP430 out of path of ESD
- Ground the connector shrouds
- Ground the enclosure



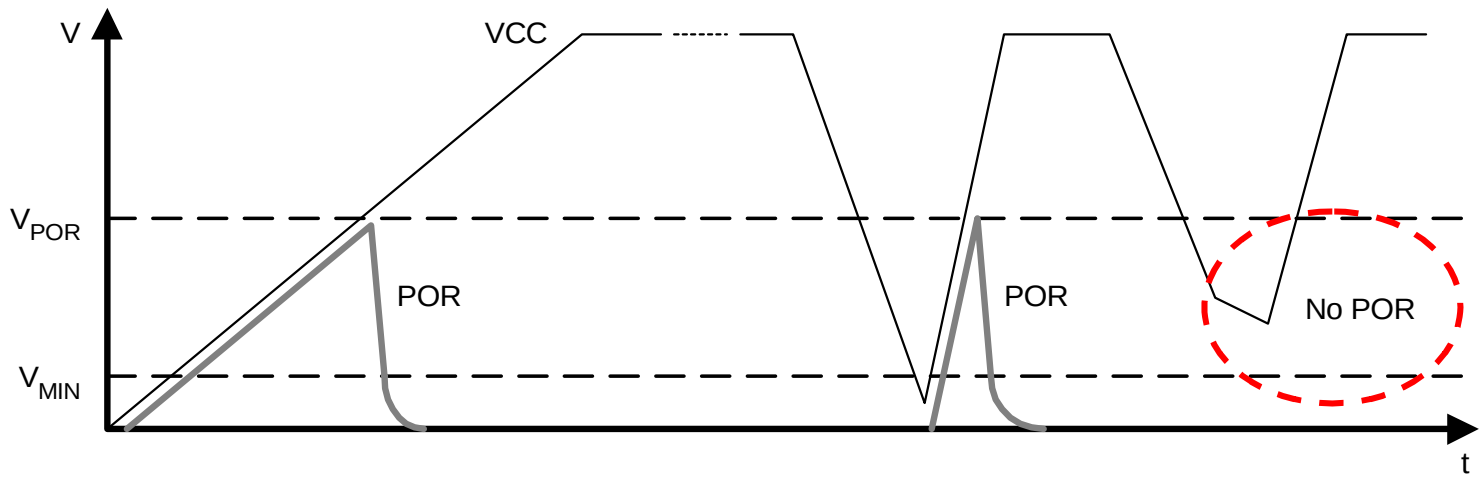
MSP430 Power On Reset (POR) Circuitry



The two parts of the power-on circuitry are:

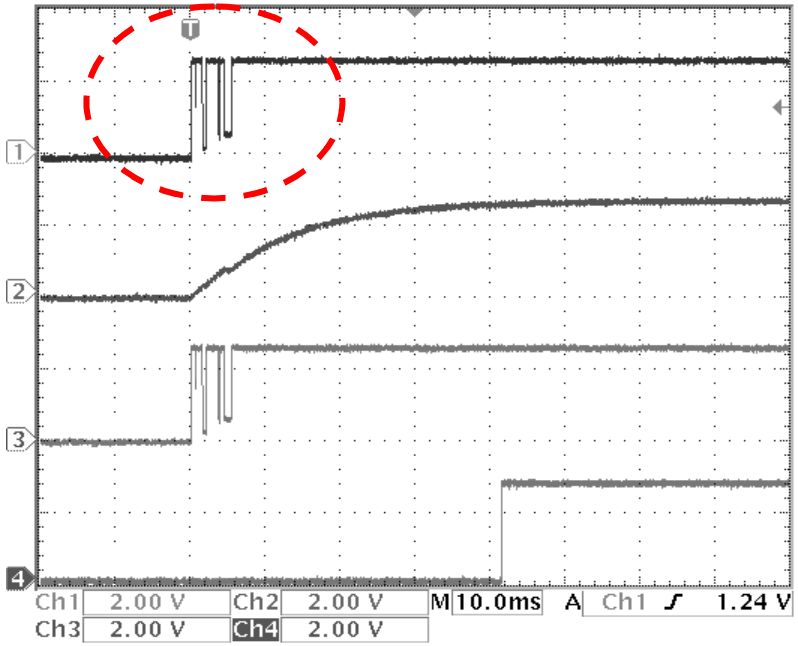
- Power-on reset detection
- Power-on reset delay

Valid POR Versus Supply Voltage “Brown-Out”



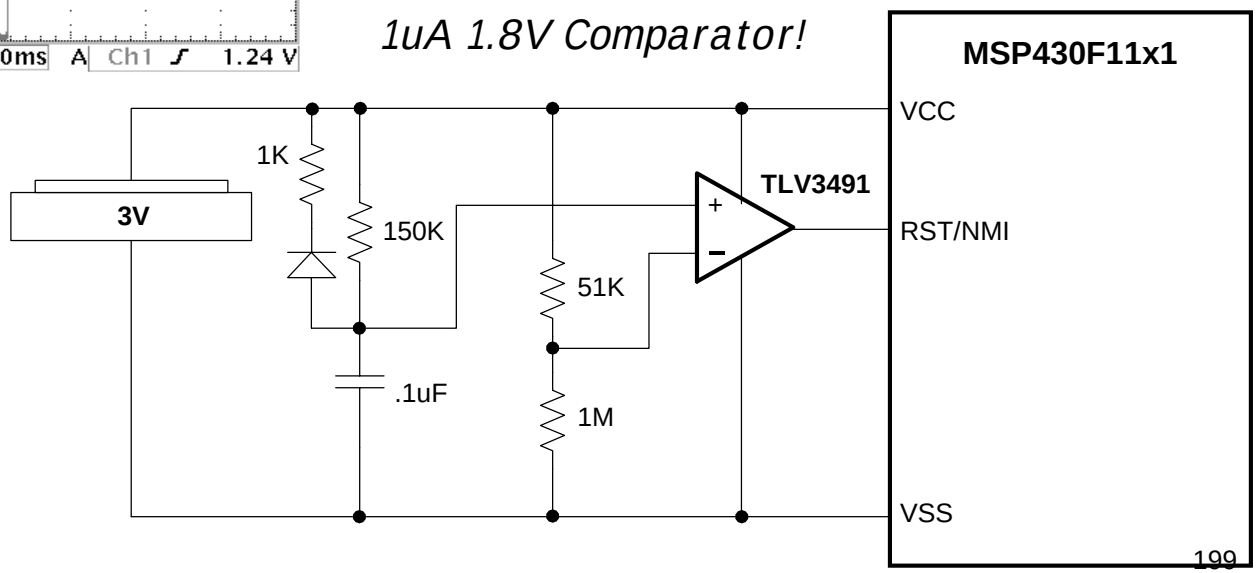
Caution: Power-on Reset is not a voltage supervising circuit!

Supply Voltage Protection Using Discrete Solutions



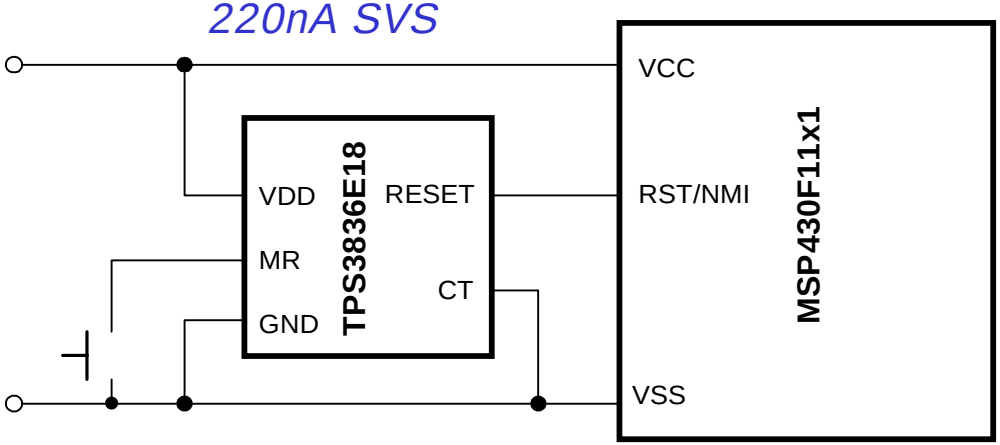
Caution: Battery Insertion is NOT a clean event!

1uA 1.8V Comparator!

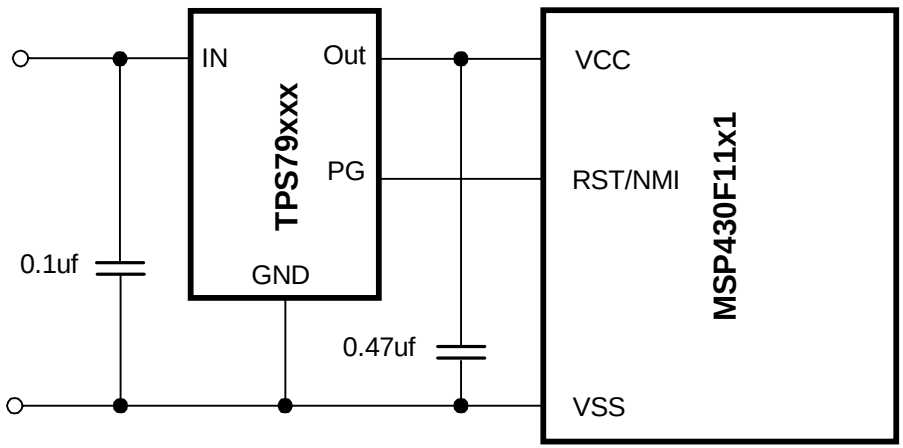


Supply Voltage Protection Using Integrated Solutions

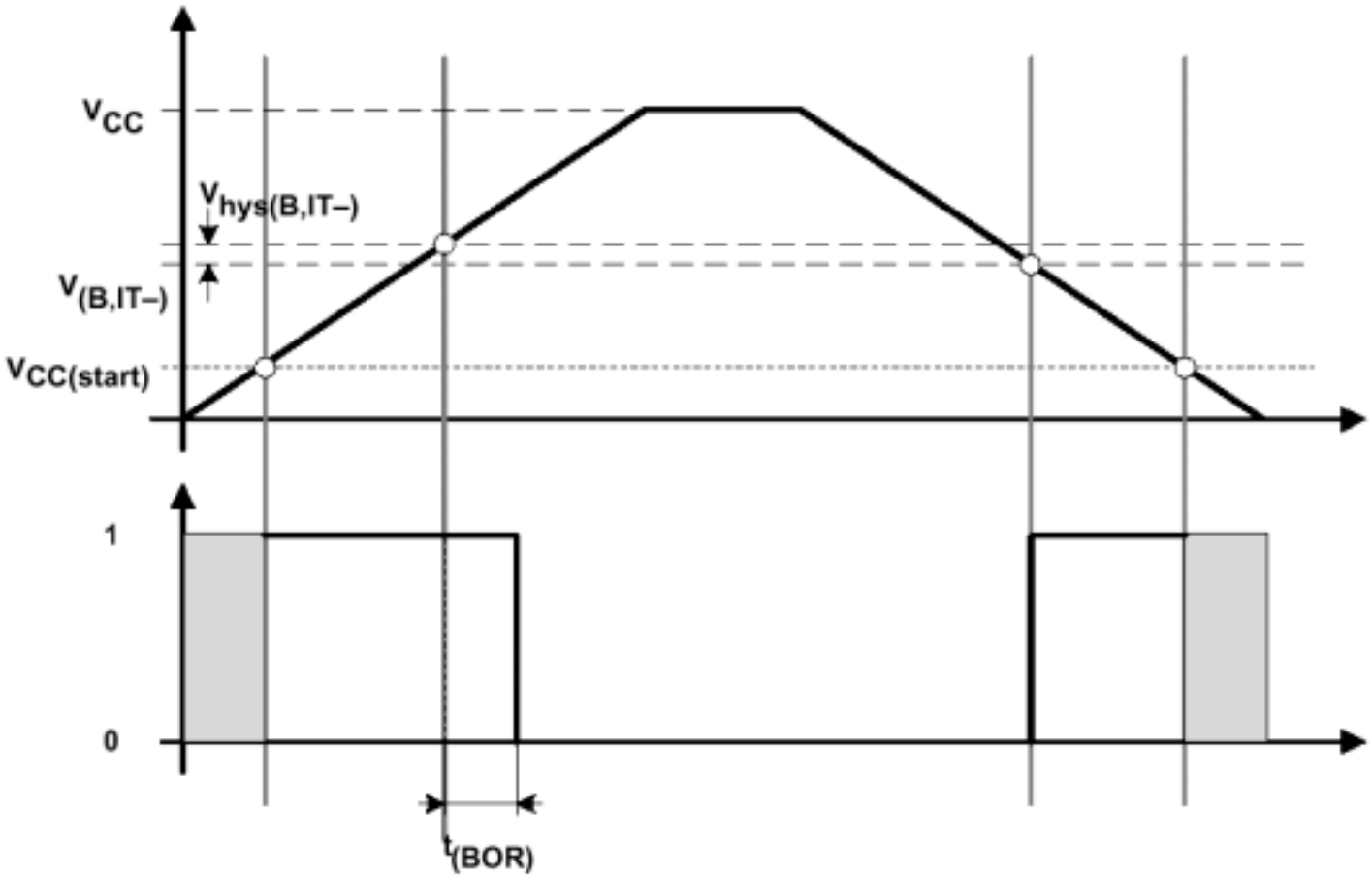
220nA SVS



1.2uA LDO + power good



MSP430F1xx2/x4xx/15x/16x New Brownout Protection



POR/Brownout Reset (BOR) vs Supply Voltage

Brownout Parameters

POR brownout, reset (see Notes 25 and 26)

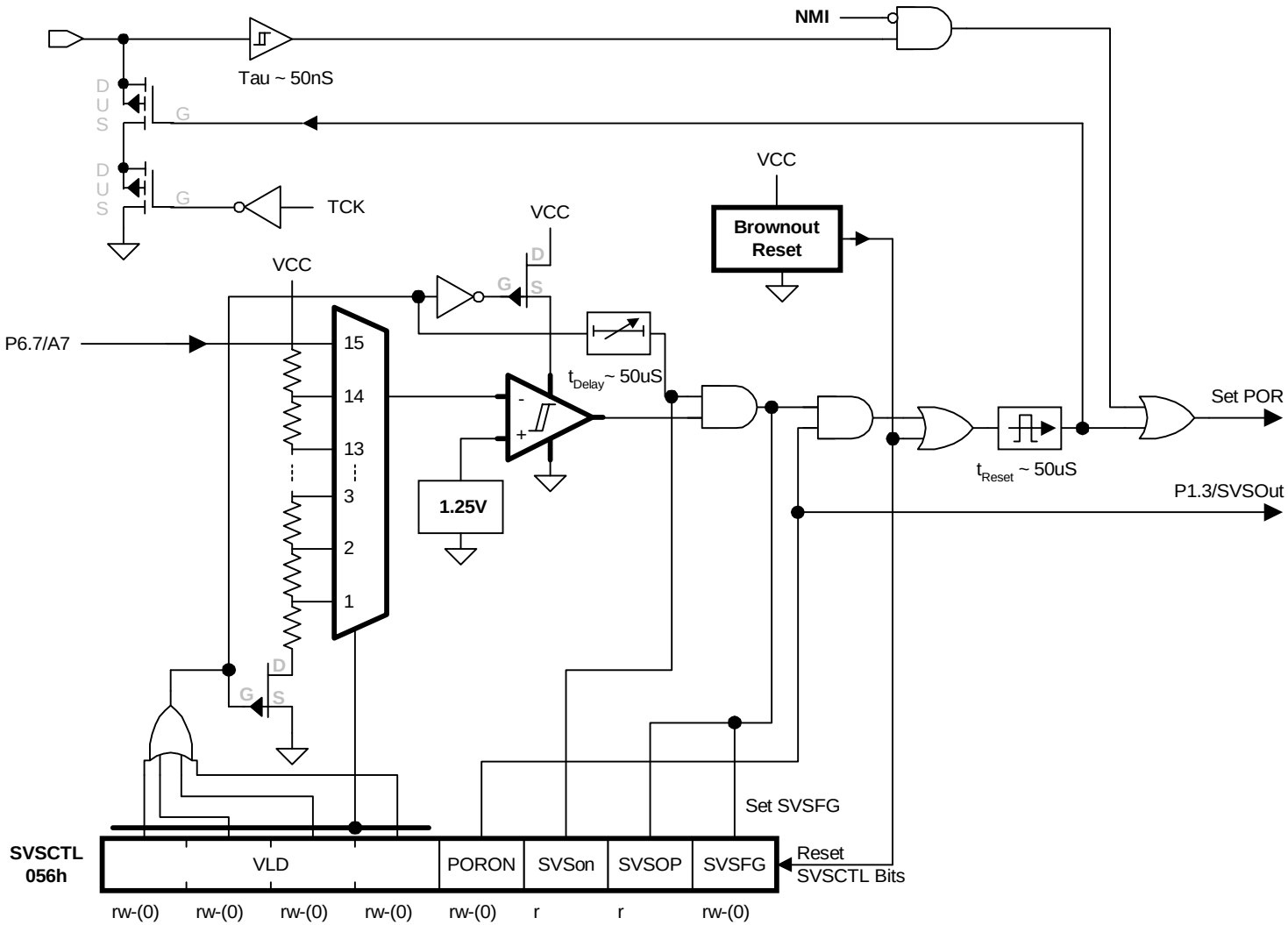
PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
$t_{BOR(delay)}$	Brownout				2000	µs
$V_{CC(BOR)}$		$dV_{CC}/dt \le 3$ V/s (see Figure 13)		$0.7 \times V_{B\_IT-}$		V
$V_{(B,IT-)}$		$dV_{CC}/dt \le 3$ V/s (see Figure 13, Figure 14, Figure 15)			1.71	V
$V_{hys(B,IT-)}$		$dV_{CC}/dt \le 3$ V/s (see Figure 13)	70	130	180	mV
$t_{(reset)}$		Pulse length needed at RST/NMI pin to accepted reset internally, V_{CC} = 2.2 V/3 V	2			µs

NOTES: 25. The current consumption of the brownout module is already included in the I_{CC} current consumption data. The voltage level $V_{(B,IT-)} + V_{hys(B,IT-)}$ is ≤ 1.8 V.

26. During power up, the CPU begins code execution following a period of $t_{BOR(delay)}$ after $V_{CC} = V_{(B,IT-)} + V_{hys(B,IT-)}$. The default FLL+ settings must not be changed until $V_{CC} \ge V_{CC(min)}$. See the *MSP430x4xx Family User's Guide* for more information on the brownout/SVS circuit.

Brownout circuit assures device startup because threshold is above the starting threshold of the CPU.

MSP430F4xx/15x/16x Supply Voltage Supervisor



Agenda - Dallas, TX - November 2002

Future Products

Get a sneak peak of what's coming in the next 12 months. Hear about the integrated sigma delta converter, programmable state machine, MEMX memory expansion, and increased processor clock.