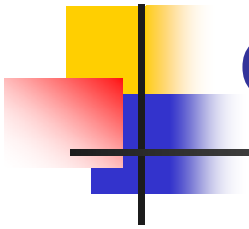
CPE 323 Introduction to Embedded Computer Systems: The MSP430 Introduction

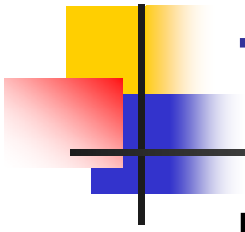
Instructor: Dr Aleksandar Milenkovic



Outline

MSP430: An Introduction

- The MSP430 family
- Technology Roadmap
- Typical Applications
- The MSP430 Documentation
- MSP430 Architecture
  - Registers
  - Addressing Modes
  - Instruction Set
  - Instruction Formats and Encodings
  - Address Space
- MSP430 Devices
- Getting Started with EasyWeb2
- MSP430 RISC core



The Family (cont'd)

- Broad family of TI's 16-bit microcontrollers (over 150 different configurations)
  - From 1 KB to 256 KB of flash memory
  - From 1 KB to 120 KB of ROM memory
  - From 128 B to 16 KB of RAM memory
  - With clock frequency of 8 KHz, 16 KHz, or 18 KHz

The Family (cont'd)

- Non-LCD based subfamilies
  - MSP430x1xx – Flash/ROM based MCUs offering 1.8V to 3.6V operation, up to 60kB, 8MIPS and a wide range of peripherals.
  - MSP430F2xx – Flash-based family featuring even lower power and up to16MIPS with 1.8 to 3.6V operation. Additional enhancements include ±1% on-chip very low power oscillator, internal pull-up/pull-down resistors and low-pin count options.
  - MSP430x5xx – New Flash-based family featuring the lowest power consumption up to 25 MIPS with 1.8 to 3.6V operation starting at 12 MIPS. Features include an innovative Power Management Module for optimizing power consumption, an internally controlled voltage regulator, and 2x more memory than previous devices.
- LCD based subfamilies
  - MSP430x3xx – Older family of ROM/OTP devices offering 2.5V-5.5V operation, up to 32kB and 4MIPS.
  - MSP430x4xx – Flash/ROM based devices offering 1.8V-3.6V operation, up to 120kB/ Flash/ ROM 8MIPS with FLL + SVS along with an integrated LCD controller. Ideal for low power metering and medical applications.

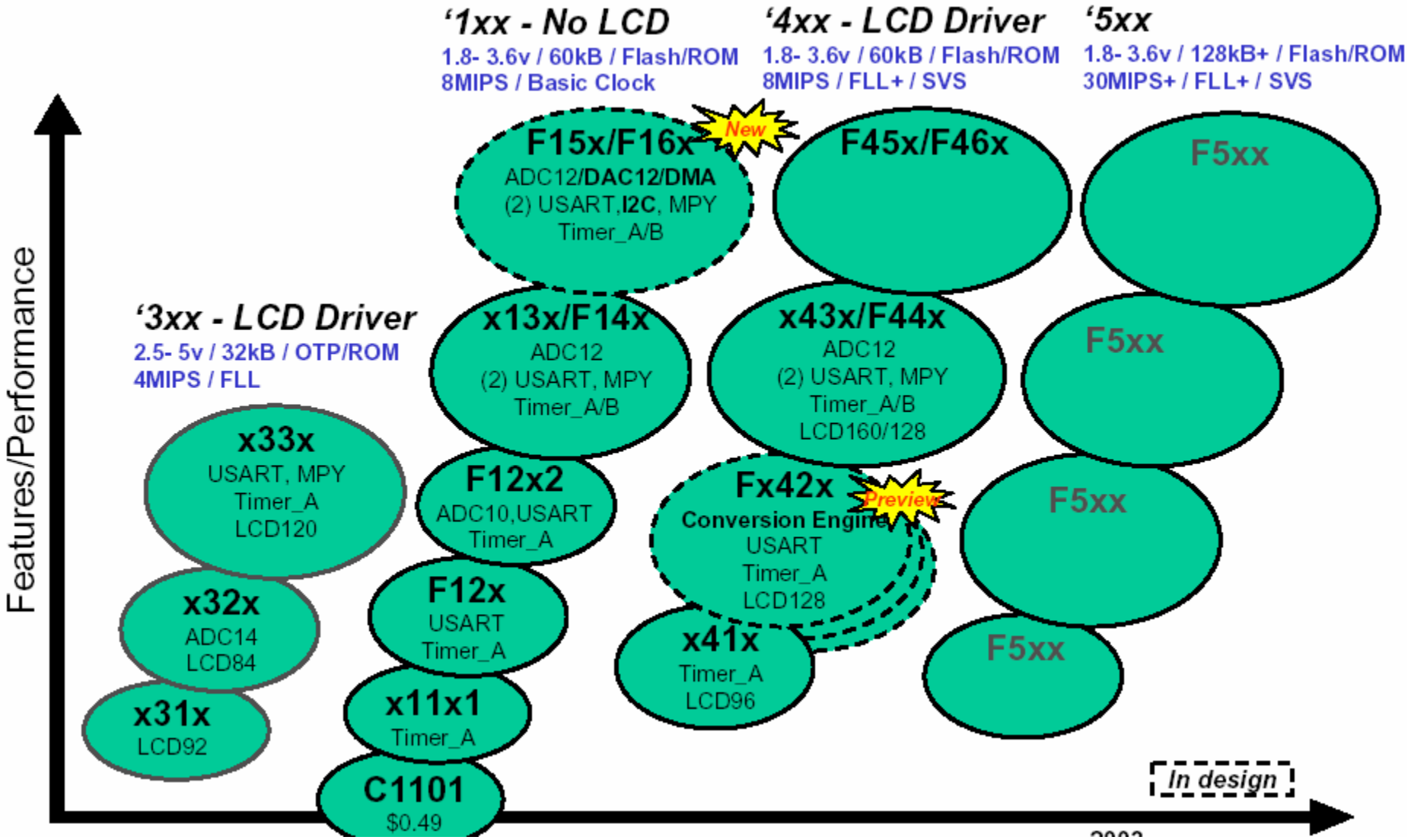
Part numbering convention

- $MSP430M_tF_aF_bM_c$
  - Mt : Memory type
    - C – ROM, F – Flash, P – OTP, E – EPROM
  - Fa,Fb
    - 10, 11 – basic
    - 12, 13 – HW UART
    - 14 – HW UART, HW multiplier
    - 31, 32 – LCD Controller
    - 33 – LCD controller, HW UART, HW multiplier
    - 41 – LCD controller
    - 43 - LCD controller, HW UART
    - 44 - LCD controller, HW UART, HW multiplier

Part numbering convention

- $MSP430M_tF_aF_bM_c$
  - Mc : Memory capacity
    - 0: 1 Kb ROM, 128 b RAM
    - 1: 2 KB ROM, 128 b RAM
    - 2: 4 KB ROM, 256 b RAM
    - ....
    - 9: 60 KB ROM, 2 Kb RAM

MSP 430 Roadmap



MSP430 Typical Applications

Handheld Measurement
- Air Flow measurement
- Alcohol meter
- Barometer
- Data loggers
- Emission/Gas analyser
- Humidity measurement
- Temperature measurement
- Weight scales

Medical Instruments
- Blood pressure meter
- Blood sugar meter
- Breath measurement
- EKG system

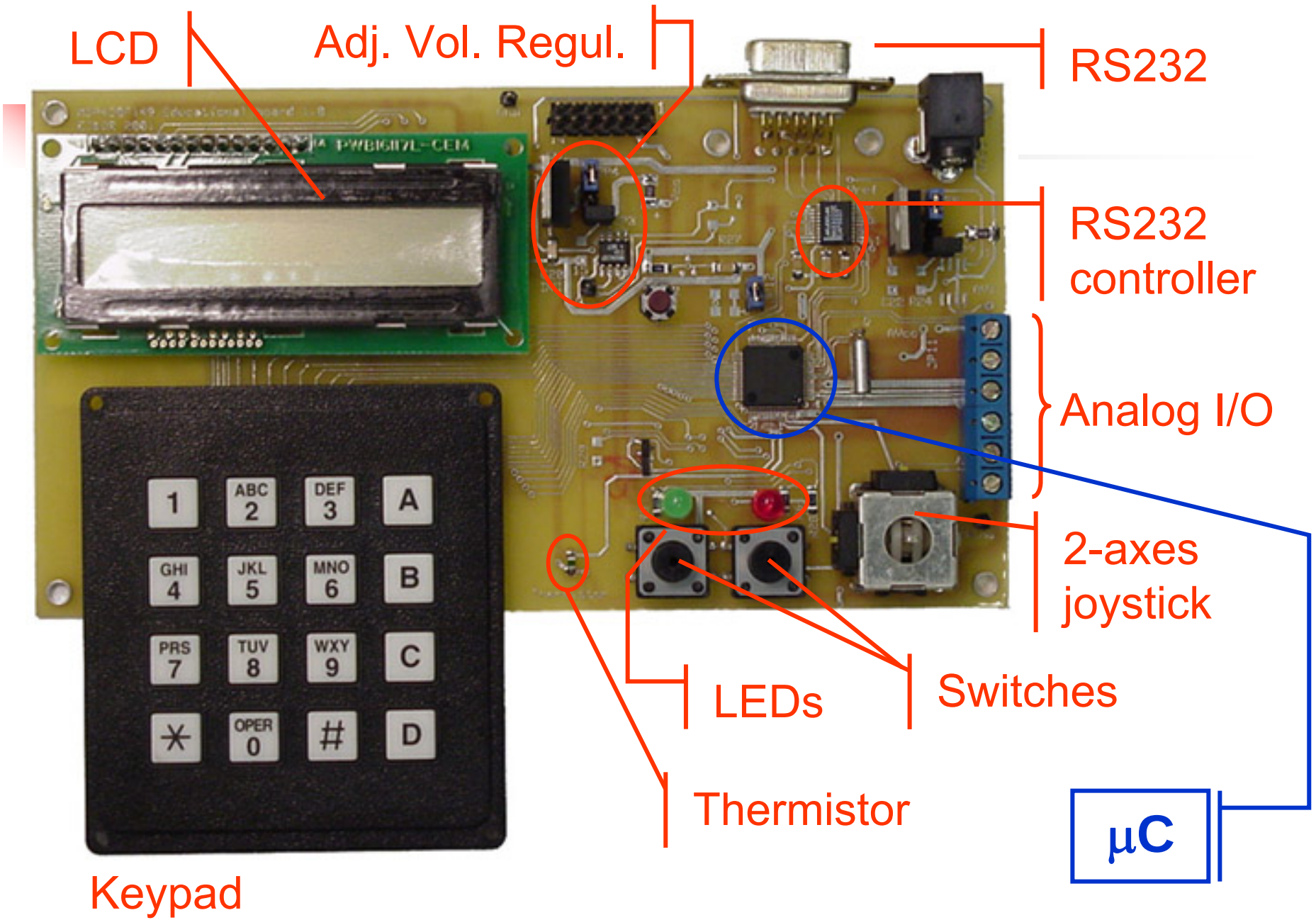
Utility Metering
- Gas Meter
- Water Meter
- Heat Volume Counter
- Heat Cost Allocation
- Electricity Meter
- Meter reading system (RF)

Sports equipment
- Altimeter
- Bike computer
- Diving watches

Security
- Glass break sensors
- Door control
- Smoke/fire/gas detectors

Home environment
- Air conditioning
- Control unit
- Thermostat
- Boiler control
- Shutter control
- Irrigation system
- White goods (Washing machine,..)

Misc
- Smart card reader
- Taxi meter
- Smart Batteries

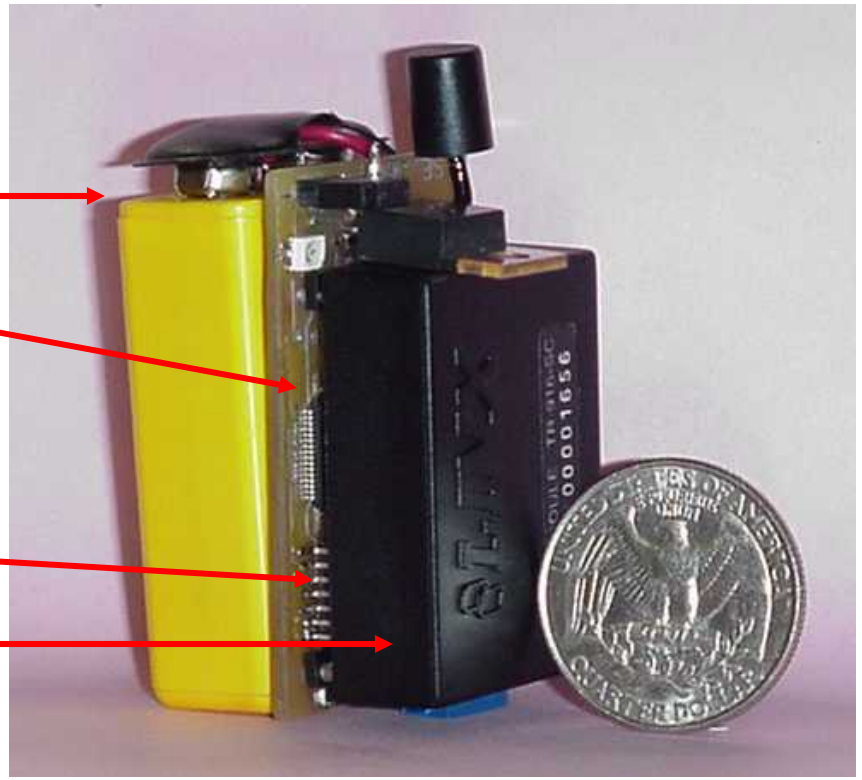


An MSP430-Based System

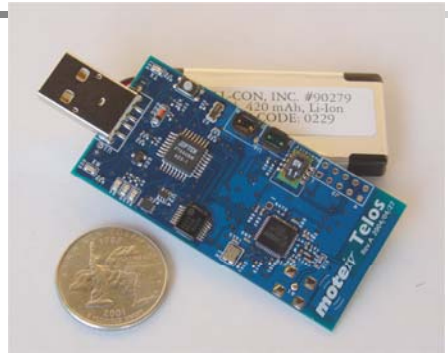
Another MSP430-Based System

Basic WISE

- Battery
- Microcontroller
  - TI MSP430F149
  - 8-channel 12-bit AD conv.
- Accelerometer
  - Movement detection
  - Analog Device ADXL202
- Transceiver
  - LINX 916 MHz

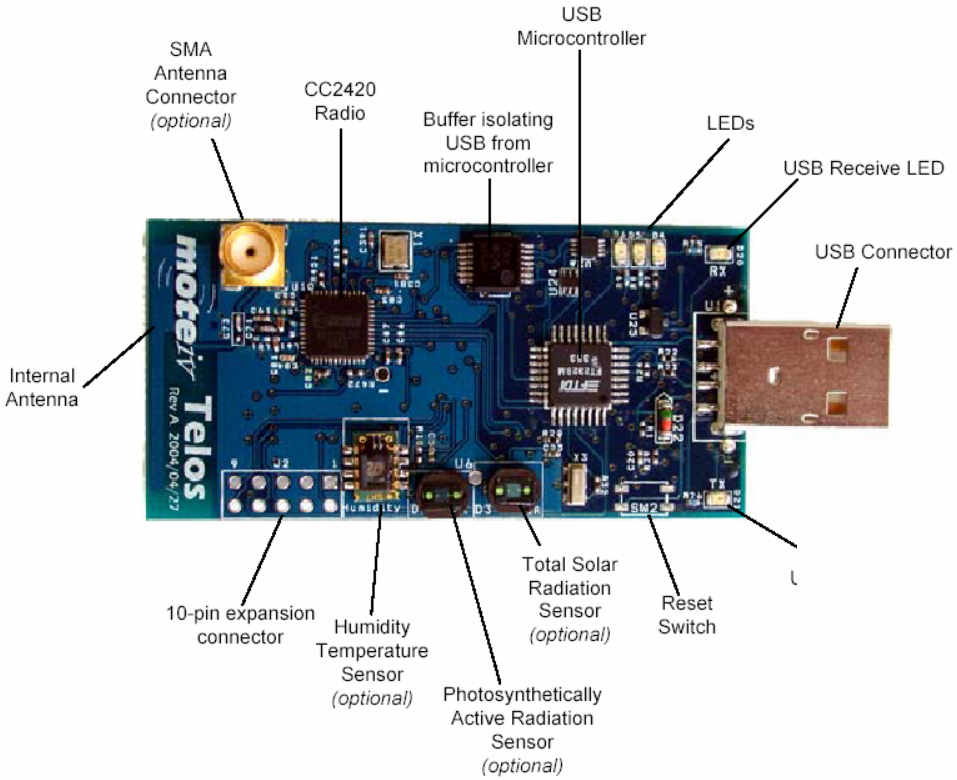


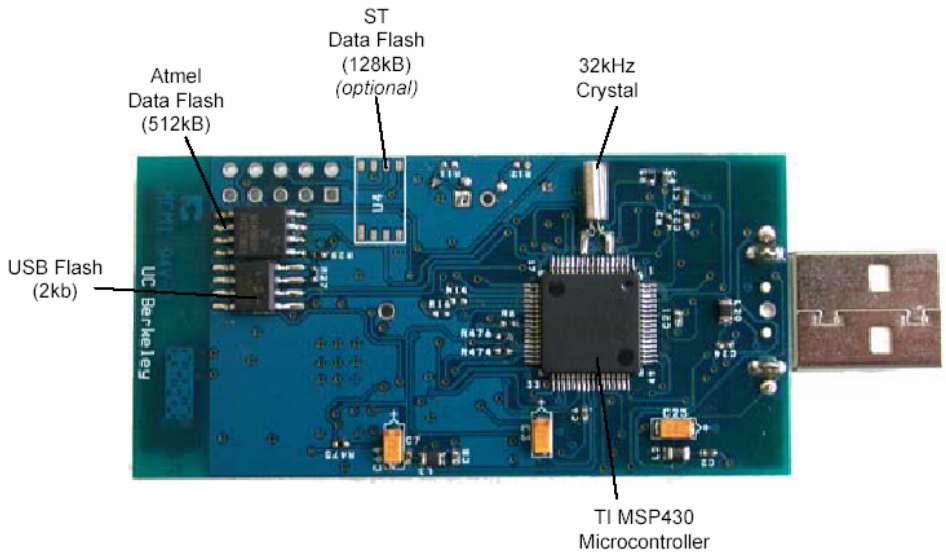
Tmote Sky Platform



- Texas Instruments 16-bit MSP430F149 microcontroller (2KB RAM, 60KB ROM)
- Chipcon 2420, 250kbps, 2.4GHz, IEEE 802.15.4 compliant wireless transceiver with programmable output power
- Integrated onboard antenna with 50m range indoors and 125m range outdoors
- Integrated humidity, temperature, and light sensors

Tmote Sky Platform





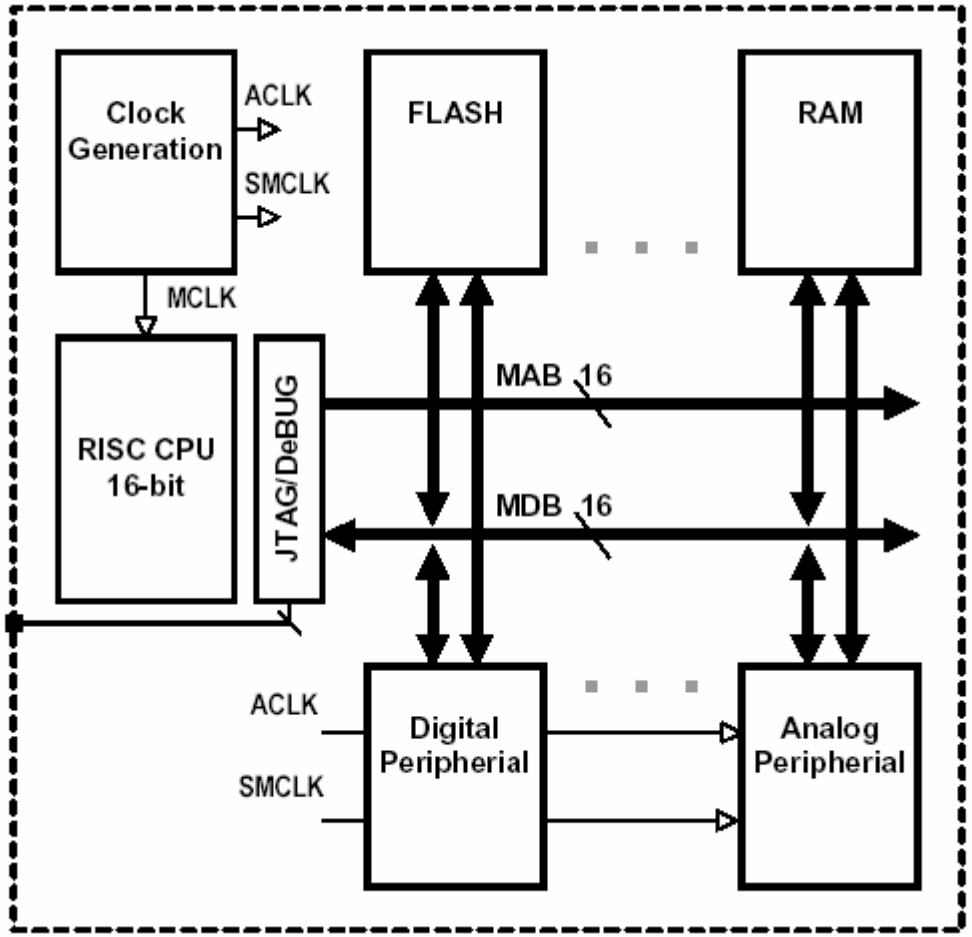
http://www.moteiv.com

MSP430 Documentation

- MSP430 home page (TI)
  - www.ti.com/msp430
- User's manual for MSP430x1xx family of devices
  - http://www.ece.uah.edu/~milenka/cpe323-08F/docs/slau049f.pdf
- User's manual for MSP430x4xx family of devices
  - http://www.ece.uah.edu/~milenka/cpe323-08F/docs/slau056g.pdf
- Datasheets
  - http://www.ece.uah.edu/~milenka/cpe323-08F/docs/msp430f149.pdf
  - http://www.ece.uah.edu/~milenka/cpe323-08F/docs/msp430f1611.pdf
  - http://www.ece.uah.edu/~milenka/cpe323-08F/docs/msp430fg4619.pdf
- TI Workshop document
  - http://www.ece.uah.edu/~milenka/cpe421-06S/docs/msp430/430_2002_atc_workshop.pdf

MSP 430 Modular Architecture

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



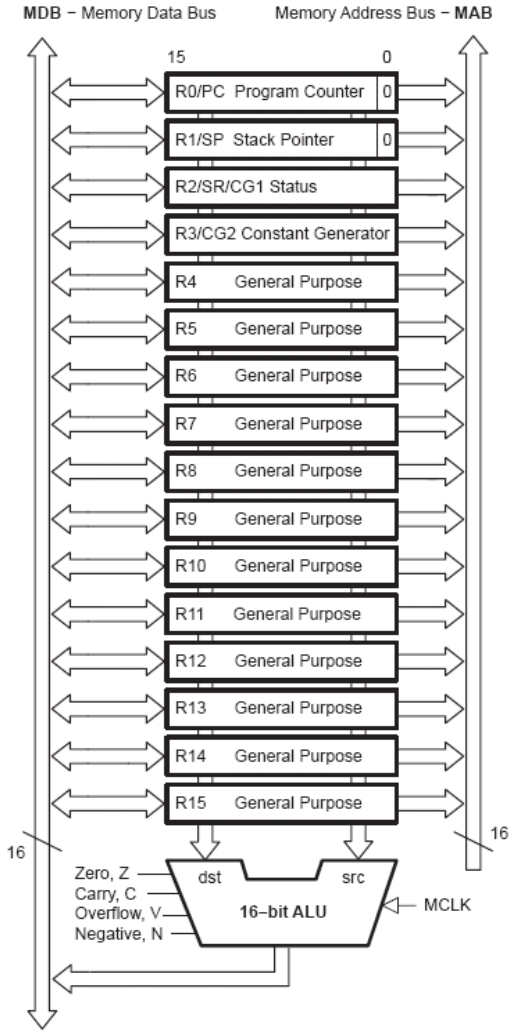
Architecture reduces power consuming, noise generating fetches to memory

Embedded emulation accessed in-application with JTAG

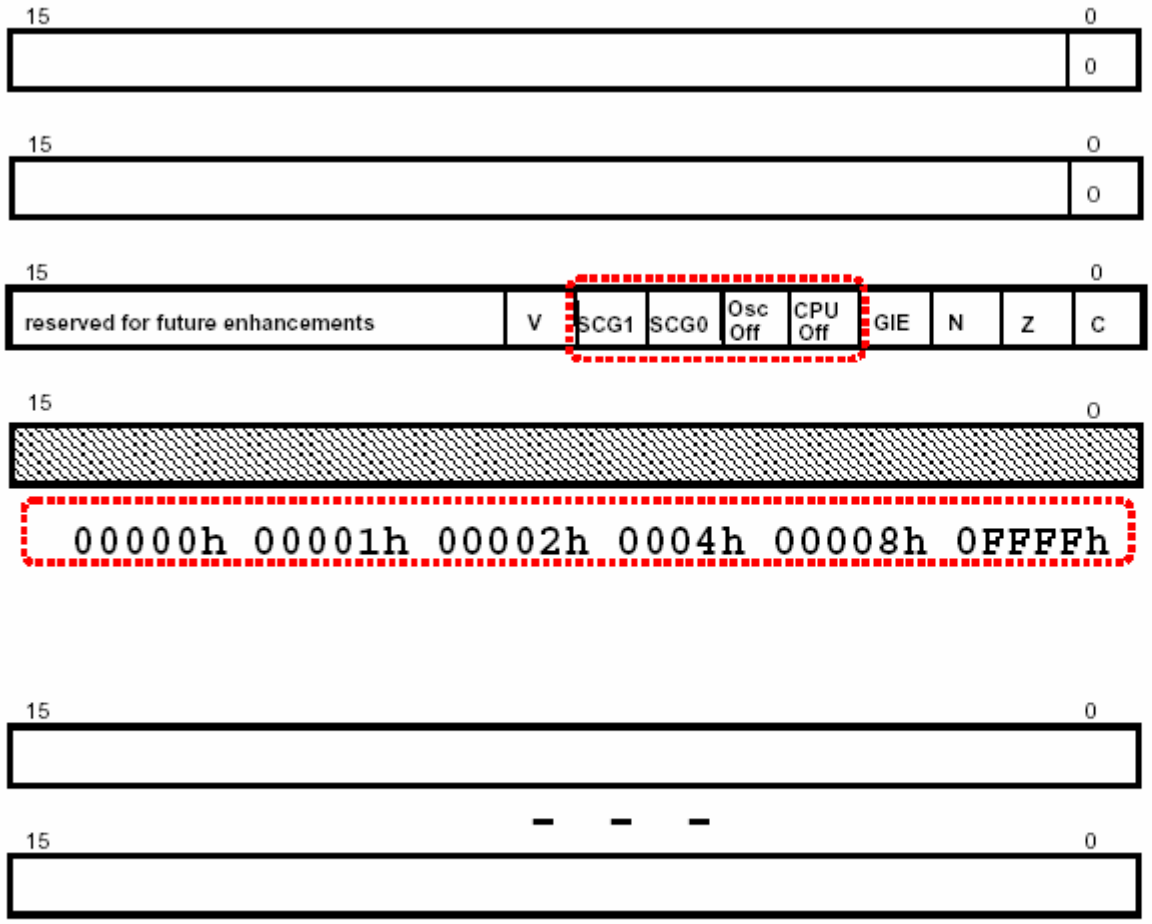
16-bit bus handles wide-width data much more effectively

MSP430 16-bit RISC

- 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



Registers



R0 - PC Program Counter
16-bit = no paging

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

R2 - SR Status Register
Define LPMx

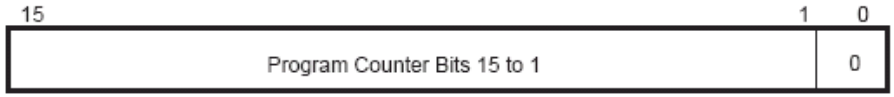
R3/R2 - CG Constant Generator
automatic generation of common used values reduces code size 30%

R4 - General Purpose

R15 - General Purpose

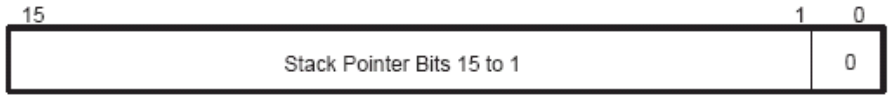
R4 through R15 are single-cycle, general purpose and identical in all respects - used for math, storage, and addressing modes.

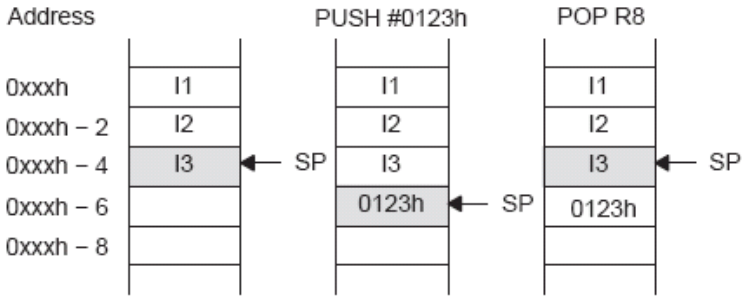
PC/R0 – Program Counter



- The 16-bit program counter (PC/R0) points to the next instruction to be executed
- Each instruction uses an even number of bytes (two, four, or six), and the PC is incremented accordingly. Instruction accesses in the 64-KB address space are performed on word boundaries, and the PC is aligned to even addresses
- PC can be addressed by all instructions and all addressing modes
  - `MOV #LABEL,PC ; Branch to address LABEL`
  - `MOV LABEL,PC ; Branch to address contained in LABEL`
  - `MOV @R14,PC ; Branch indirect to address in R14`

SP/R1 – Stack Pointer

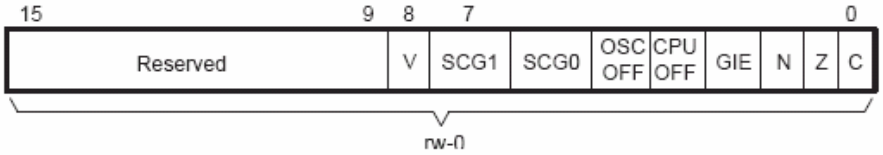




- The stack pointer (SP/R1) is used by the CPU to store the return addresses of subroutine calls and interrupts. It uses a predecrement, postincrement scheme.
- In addition, the SP can be used by software with all instructions and addressing modes.
- Examples
  - `MOV 2(SP),R6 ; Item I2 -> R6`
  - `MOV R7,0(SP) ; Overwrite TOS with R7`
  - `PUSH #0123h ; Put 0123h onto TOS`
  - `POP R8 ; R8 = 0123h`

- Question: Illustrate the stack contents after PUSH SP and POP SP instructions are executed?

SR/R2 – Status Register



- The status register (SR/R2), used as a source or destination register, can be used in the register mode only addressed with word instructions.
- The remaining combinations of addressing modes are used to support the constant generator.

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

Constant Generation

- Six commonly-used constants are generated with the constant generator registers R2 and R3,
  - Adv.: No special instructions, no special code, no extra memory access
- Assembler uses the constant generator automatically if one of the six constants is used as an immediate source operand. Registers R2 and R3, used in the constant mode, cannot be addressed explicitly; they act as source-only registers.
- The constants are selected with the source-register addressing modes (As), as described below.

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

Constant Generation

- Constant generator allows for additional 24 instructions that are emulated
- Examples
  - `CLR dst` `MOV R3,dst`
  - `INC dst` `ADD 0(R3),dst`

General-Purpose Registers

- The twelve registers, R4−R15, are general-purpose registers. All of these registers can be used as data registers, address pointers, or index values and can be accessed with byte or word instructions as shown below

Example Register-Byte Operation

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

```
ADD.B     R5,0(R6)
```

```
   08Fh
+ 012h
------
   0A1h
```

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

(Low byte of register)
+ (Addressed byte)
->(Addressed byte)

Example Byte-Register Operation

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

```
ADD.B     @R6,R5
```

```
   05Fh
+ 002h
------
 00061h
```

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

(Addressed byte)
+ (Low byte of register)
->(Low byte of register, zero to High byte)

Addressing Modes

- Seven addressing modes for the source operand and four addressing modes for the destination operand can address the complete address space with no exceptions.

Table 2. Address Mode Descriptions

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

NOTE: S = source D = destination

Addressing Modes

- The bit numbers in the table below describe the contents of the As (source) and Ad (destination) mode bits.

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

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 Addressing Mode (cont'd)

Example: `MOV R10,R11`

	Before:		After:
R10	0A023h	R10	0A023h
R11	0FA15h	R11	0A023h
PC	PC_{old}	PC	PC_{old} + 2

Note: Data in Registers

The data in the register can be accessed using word or byte instructions. If byte instructions are used, the high byte is always 0 in the result. The status bits are handled according to the result of the byte instruction.

Register-Indexed Addressing Mode

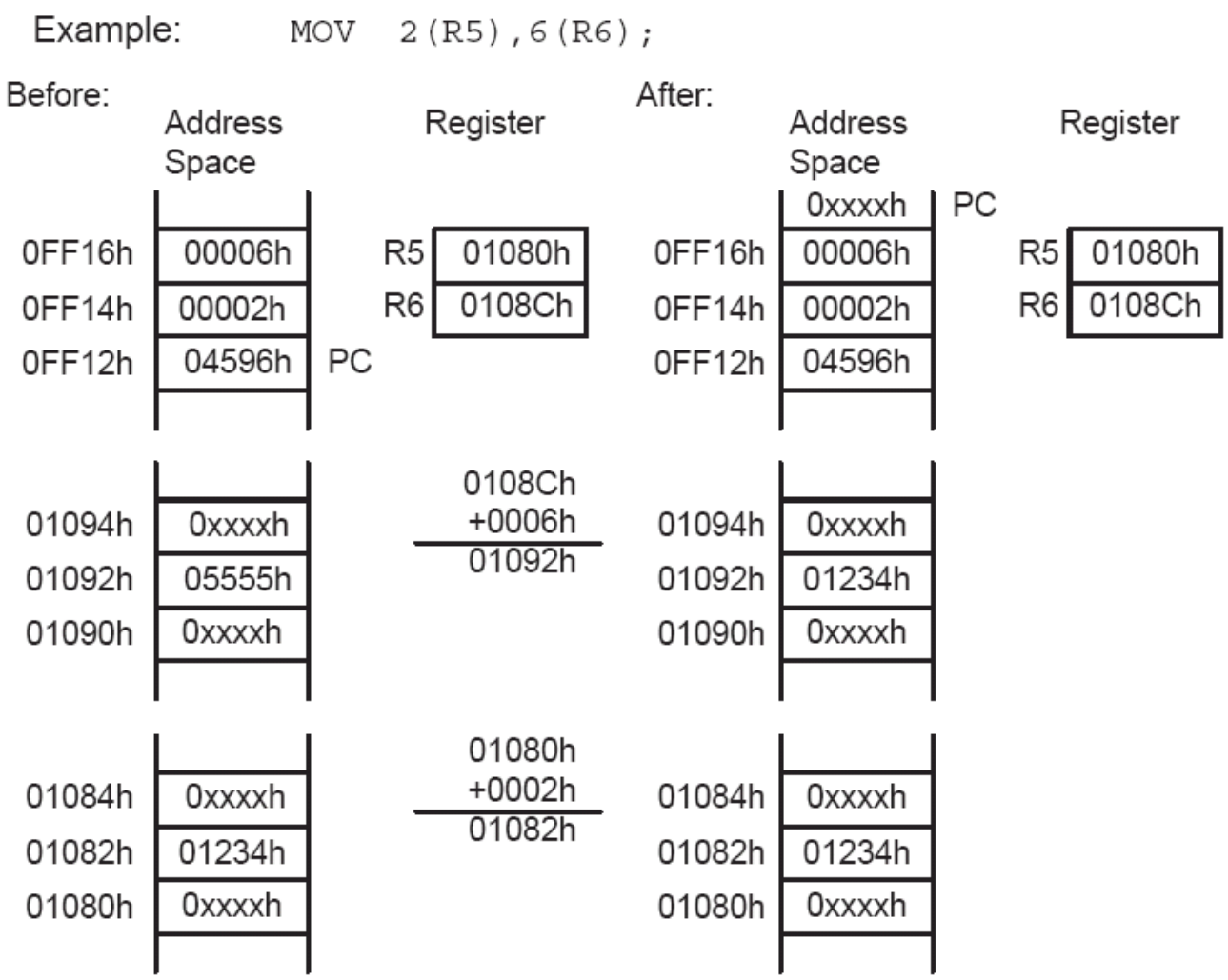
Op-Code	Source-Register	Ad	B/W	As	Destination-Register
0100	0100	1	0	01	0101

```
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.

Register-Indexed Addressing Mode (cont'd)



Symbolic Addressing Mode

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

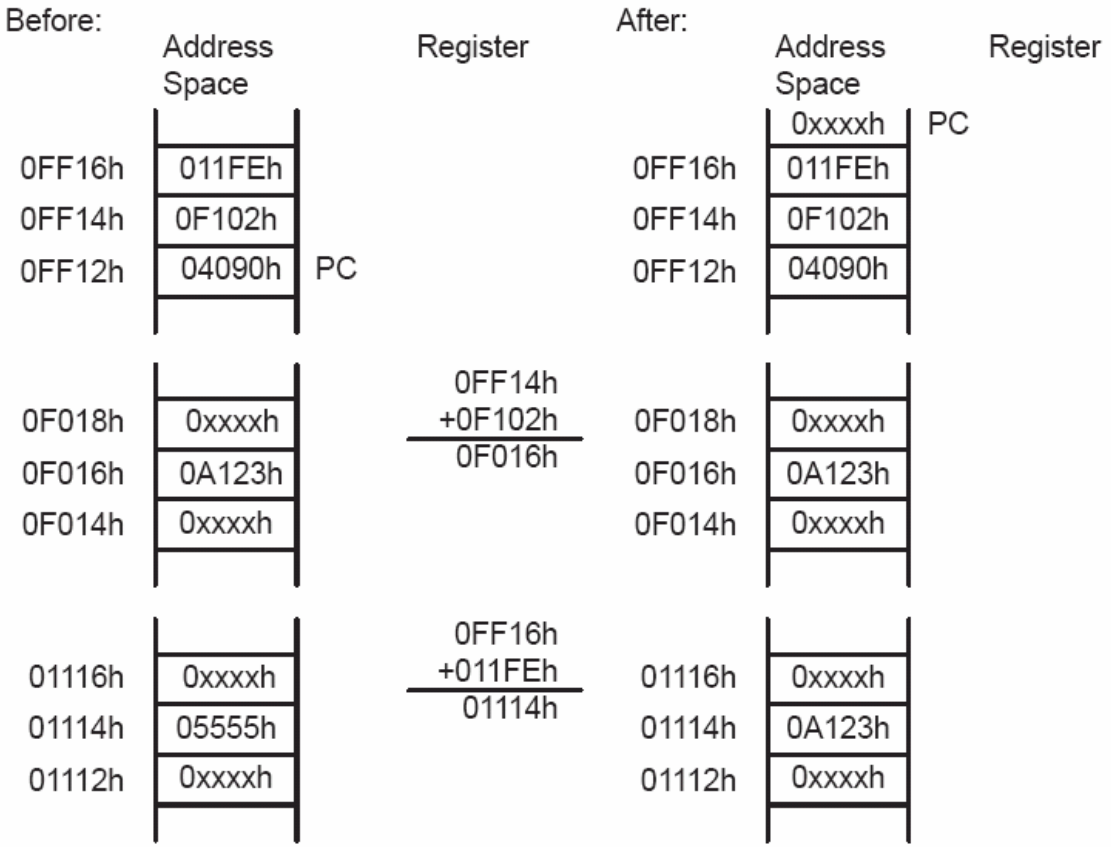
```
4090ffa80006   mov.w    EDE,TONI          ;

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.

Symbolic Addressing Mode (cont'd)

Example:

```
MOV  EDE,TONI ;Source address EDE = 0F016h
              ;Dest. address TONI=01114h
```



Absolute Addressing Mode

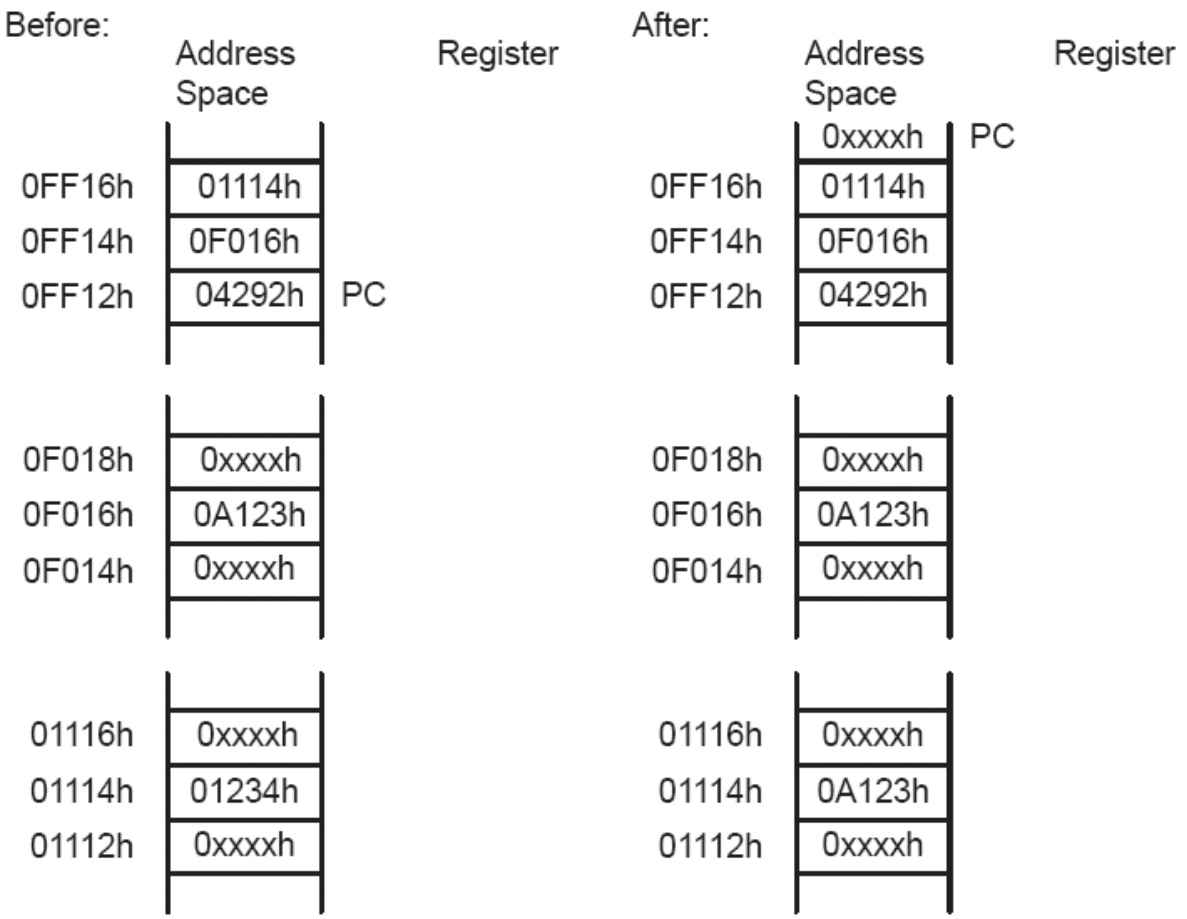
Op-Code	Source-Register	Ad	B/W	As	Destination-Register
0100	0010	1	0	01	0010

```
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.

Absolute Addressing Mode (cont'd)

```
Example:    MOV  &EDE,&TONI ;Source address EDE=0F016h,
                            ;dest. address TONI=01114h
```



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          ;
```

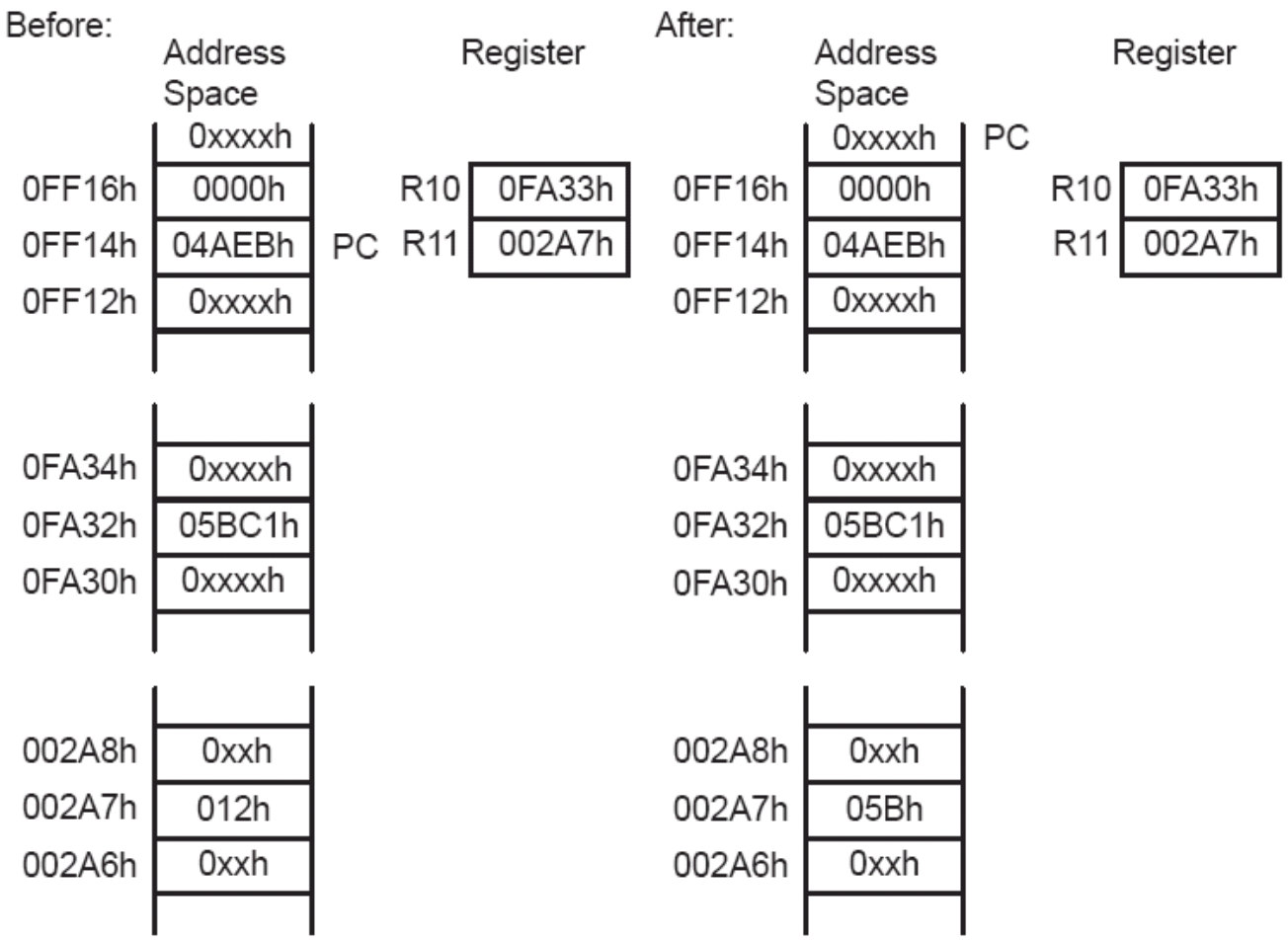
Source only As=10, Ad=n/a

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 Addressing Mode (cont'd)

Example: `MOV.B @R10,0(R11)`



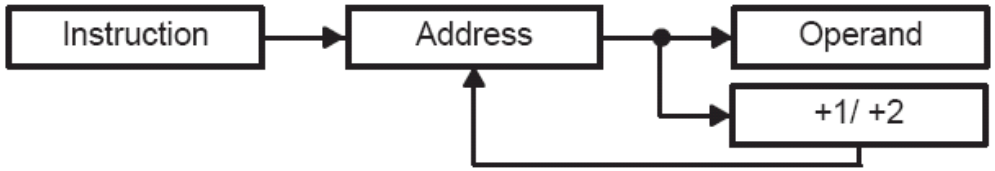
Register Indirect Autoincrement Addressing Mode

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

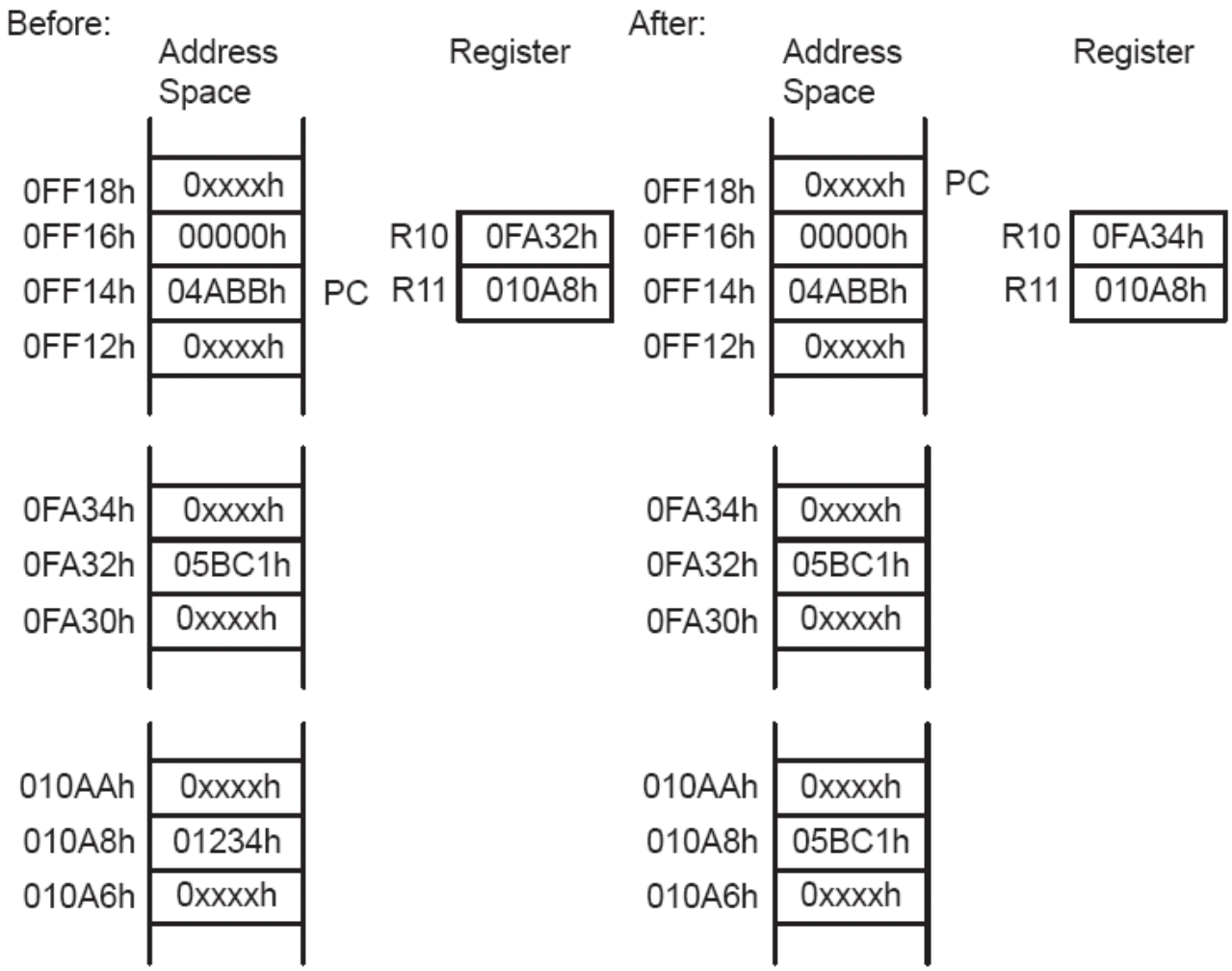
```
4435          mov.w    @R4+,R5              ;

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.



Register Indirect Autoincrement Addressing Mode (cont'd)



Immediate Addressing Mode

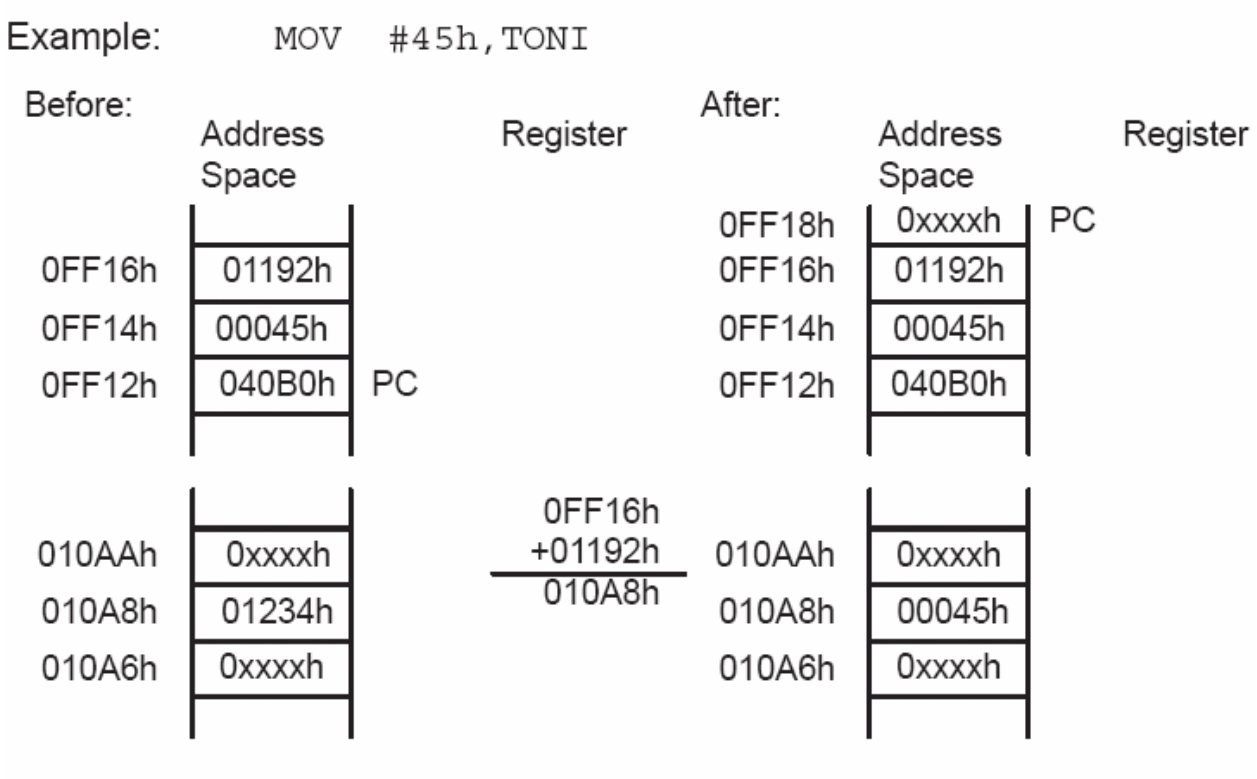
Op-Code	Source-Register	Ad	B/W	As	Destination-Register
0100	0000	0	0	11	0101

```
40351234        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.

Immediate Addressing Mode (cont'd)



Instruction Set

- 27 core instructions
  - Have unique op-codes decoded by the CPU
- 24 emulated instructions
  - Make code easier to write and read, but do not have op-codes; instead an equivalent core instruction is generated
  - No code or performance penalty for using emulated instructions
- 3 core instruction formats
  - Dual-operand
  - Single-operand
  - Jump
- All single- and dual-operand instructions can be byte or word instructions by using .B or .W (default) extensions
  - Byte instructions are used to access byte data or byte peripherals
  - Word instructions are used to access word data or word peripherals.

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

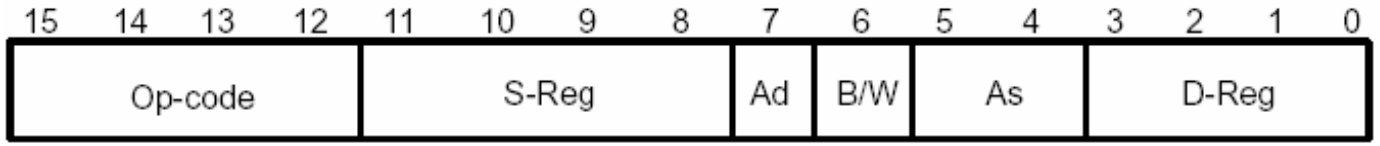
- Simply easier to understand with no code size or speed penalty
- Replaced by assembler with core instructions using CG, PC and SP

```
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

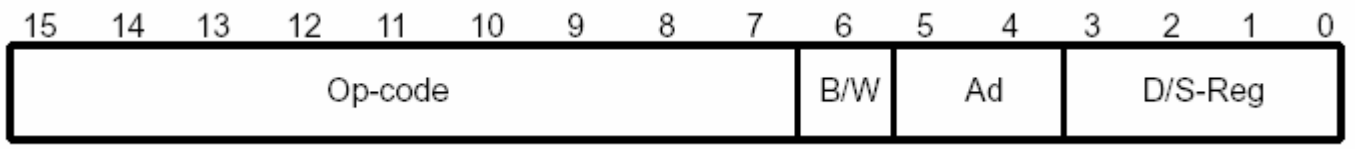
Format I Source,Destination	Format II Single Operand	Format III +/- 9bit Offset	Support
add(.b)	**br**	jmp	**clrc**
addc(.b)	call	jc	**setc**
and(.b)	swpb	jnc	**clrz**
bic(.b)	sxt	jeq	**setz**
bis(.b)	push(.b)	jne	**clrn**
bit(.b)	**pop.(b)**	jge	**setn**
cmp(.b)	rra(.b)	jl	**dint**
dadd(.b)	rrc(.b)	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)		

Double operand instructions



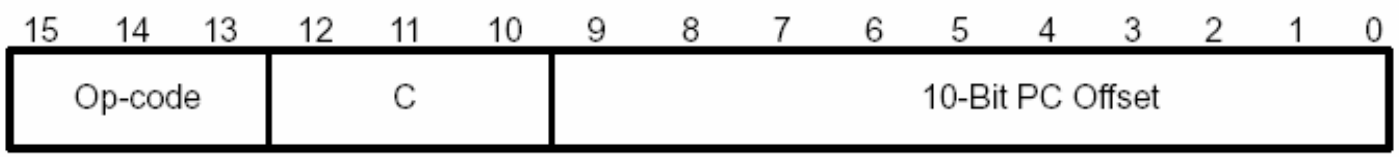
Mnemonic	S-Reg, D-Reg	Operation	Status Bits			
			V	N	Z	C
MOV(.B)	src,dst	src → dst	–	–	–	–
ADD(.B)	src,dst	src + dst → dst	*	*	*	*
ADDC(.B)	src,dst	src + dst + C → dst	*	*	*	*
SUB(.B)	src,dst	dst + .not.src + 1 → dst	*	*	*	*
SUBC(.B)	src,dst	dst + .not.src + C → dst	*	*	*	*
CMP(.B)	src,dst	dst – src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

Single Operand Instruction



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

Jump Instructions



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

3 Instruction Formats

```
; Format I Source and Destination
```

Op-Code	Source-Register	Ad	B/W	As	Destination-Register

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

; Format II Destination Only
```

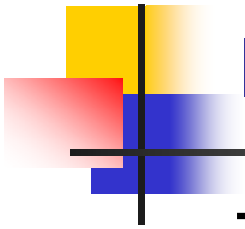
Op-Code	B/W	Ad	D/S- Register

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

; Format III There are 8(Un)conditional Jumps
```

Op-Code	Condition	10-bit PC offset

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

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



Instruction Cycles and Lengths

- The number of CPU clock cycles required for an instruction depends on the instruction format and the addressing modes used - not the instruction itself
- The number of clock cycles refers to the MCLK

Format I: Instruction Cycles and Length

Table 3−16. Format 1 Instruction Cycles and Lengths

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

Format II and Format III: Instruction Cycles and Length

Table 3−15. Format-II Instruction Cycles and Lengths

Addressing Mode	No. of Cycles: RRA, RRC SWPB, SXT	No. of Cycles: PUSH	No. of Cycles: CALL	Length of Instruction	Example
Rn	1	3	4	1	`SWPB R5`
@Rn	3	4	4	1	`RRC @R9`
@Rn+	3	5	5	1	`SWPB @R10+`
#N	(See note)	4	5	2	`CALL #0F000h`
X(Rn)	4	5	5	2	`CALL 2(R7)`
EDE	4	5	5	2	`PUSH EDE`
&EDE	4	5	5	2	`SXT &EDE`

- Format III: all jump instructions take 2 clock cycles to execute and are 1 word long
- Interrupt and reset cycles

Table 3−14. Interrupt and Reset Cycles

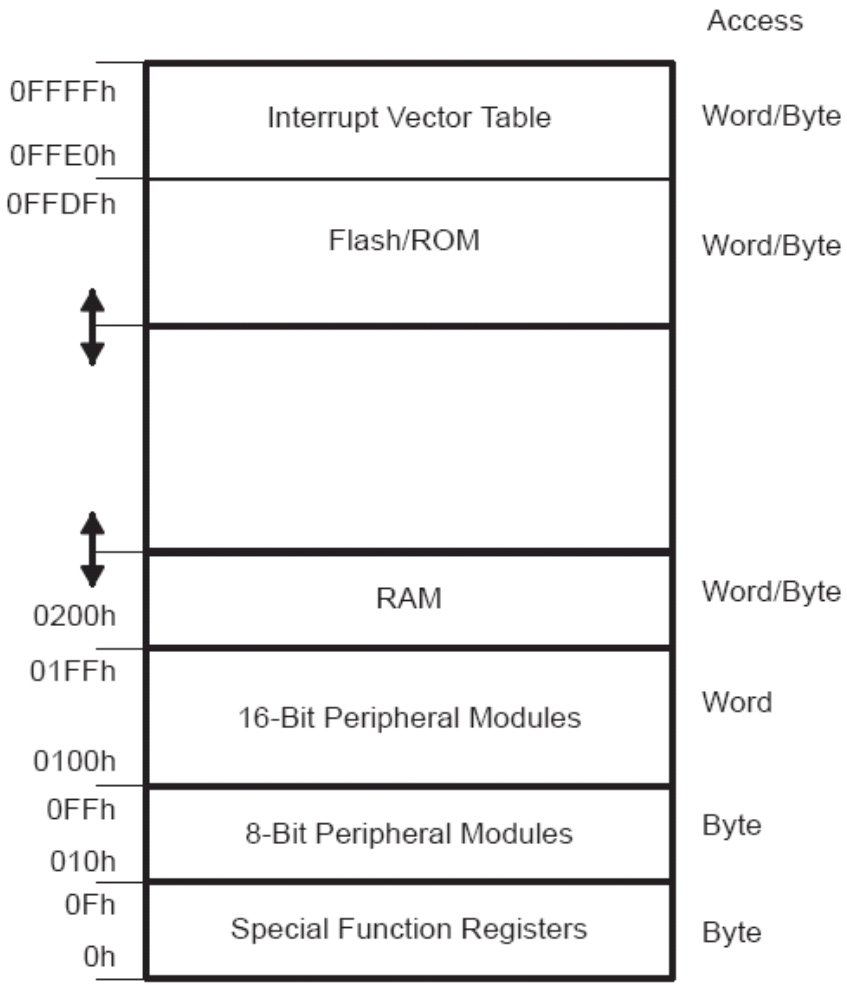
Action	No. of Cycles	Length of Instruction
Return from interrupt (`RETI`)	5	1
Interrupt accepted	6	–
WDT reset	4	–
Reset ($\overline{\text{RST}}$/NMI)	4	–

Instruction Encoding

	000	040	080	0C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0
0xxx																
4xxx																
8xxx																
Cxxx																
1xxx	RRC	RRC.B	SWPB		RRA	RRA.B	SXT		PUSH	PUSH.B	CALL		RETI			
14xx																
18xx																
1Cxx																
20xx	JNE/JNZ															
24xx	JEQ/JZ															
28xx	JNC															
2Cxx	JC															
30xx	JN															
34xx	JGE															
38xx	JL															
3Cxx	JMP															
4xxx	MOV, MOV.B															
5xxx	ADD, ADD.B															
6xxx	ADDC, ADDC.B															
7xxx	SUBC, SUBC.B															
8xxx	SUB, SUB.B															
9xxx	CMP, CMP.B															
Axxx	DADD, DADD.B															
Bxxx	BIT, BIT.B															
Cxxx	BIC, BIC.B															
Dxxx	BIS, BIS.B															
Exxx	XOR, XOR.B															
Fxxx	AND, AND.B															

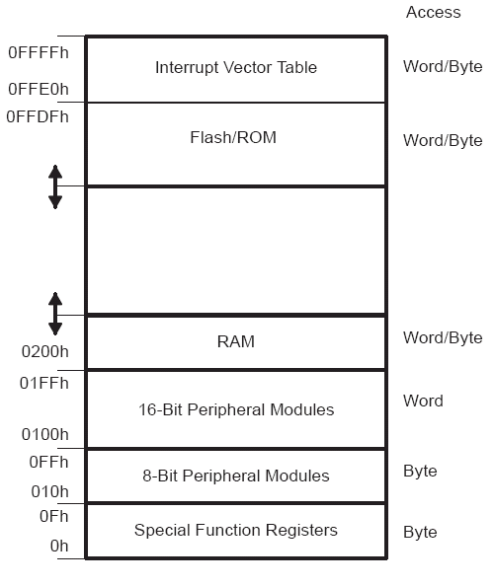
Address Space

- The MSP430x1xx von-Neumann architecture has one address space shared with special function registers (SFRs), peripherals, RAM, and Flash/ROM memory as shown
- Memory maps are device specific
- Code access are always performed on even addresses.
- Data can be accessed as bytes or words.
- The addressable memory space is 64 KB with future expansion planned.



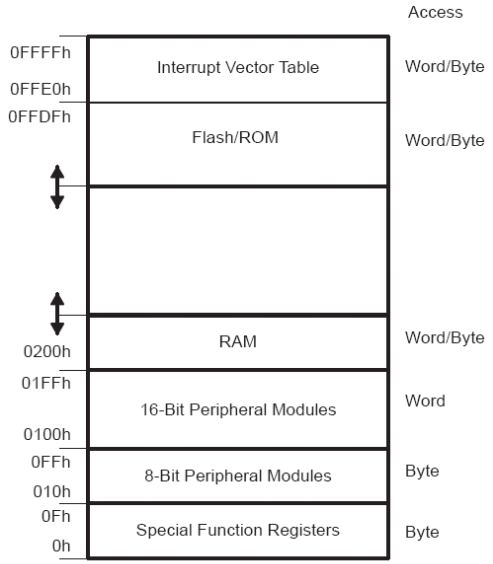
Address Space (cont'd)

- Special Function Registers (SFRs)
  - Some peripheral functions are configured in the SFRs
  - The SFRs are located in the lower 16 bytes of the address space, and are organized by byte
  - SFRs must be accessed using byte instructions only. See the device-specific data sheets for applicable SFF bits
- Peripheral modules (PM)
  - Peripheral modules are mapped into the address spac
  - Address space 0100-01FFh is reserved for 16-bit PMs
    - Should be accessed with word instructions.
    - If byte instructions are used, only even addresses are permissible, and the high byte of the result is always 0.
  - Address space 010h-0FFh is reserved for 8-bit PMs
    - Should be accessed with byte instructions.
    - Read access of byte modules using word instructions results in unpredictable data in the high byte.
    - If word data is written to a byte module only the low byte is written into the peripheral register, ignoring the high byte.



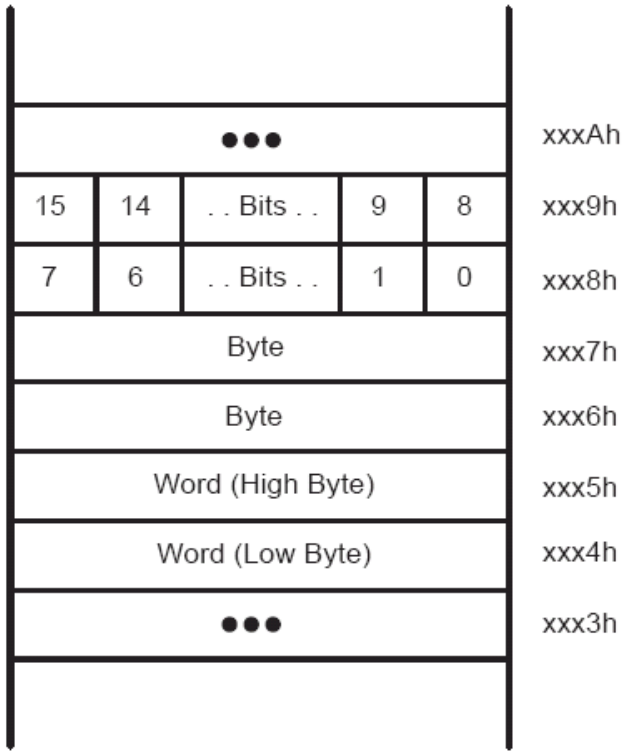
Address Space (cont'd)

- RAM
  - RAM starts at 0200h.
  - End address of RAM depends on the amount of RAM present and varies by device.
  - RAM can be used for both code and data
- Flash/ROM
  - Start address of Flash/ROM depends on the amount of Flash/ROM present and varies by device.
  - End address for Flash/ROM is 0FFFFh
  - Flash can be used for both code and data. Word or byte tables can be stored and used in Flash/ROM without the need to copy the tables to RAM before using them.
- Interrupt vector table
  - Is mapped into the upper 16 words of Flash/ROM address space, with the highest priority interrupt vector at the highest Flash/ROM word address (0FFFEh).

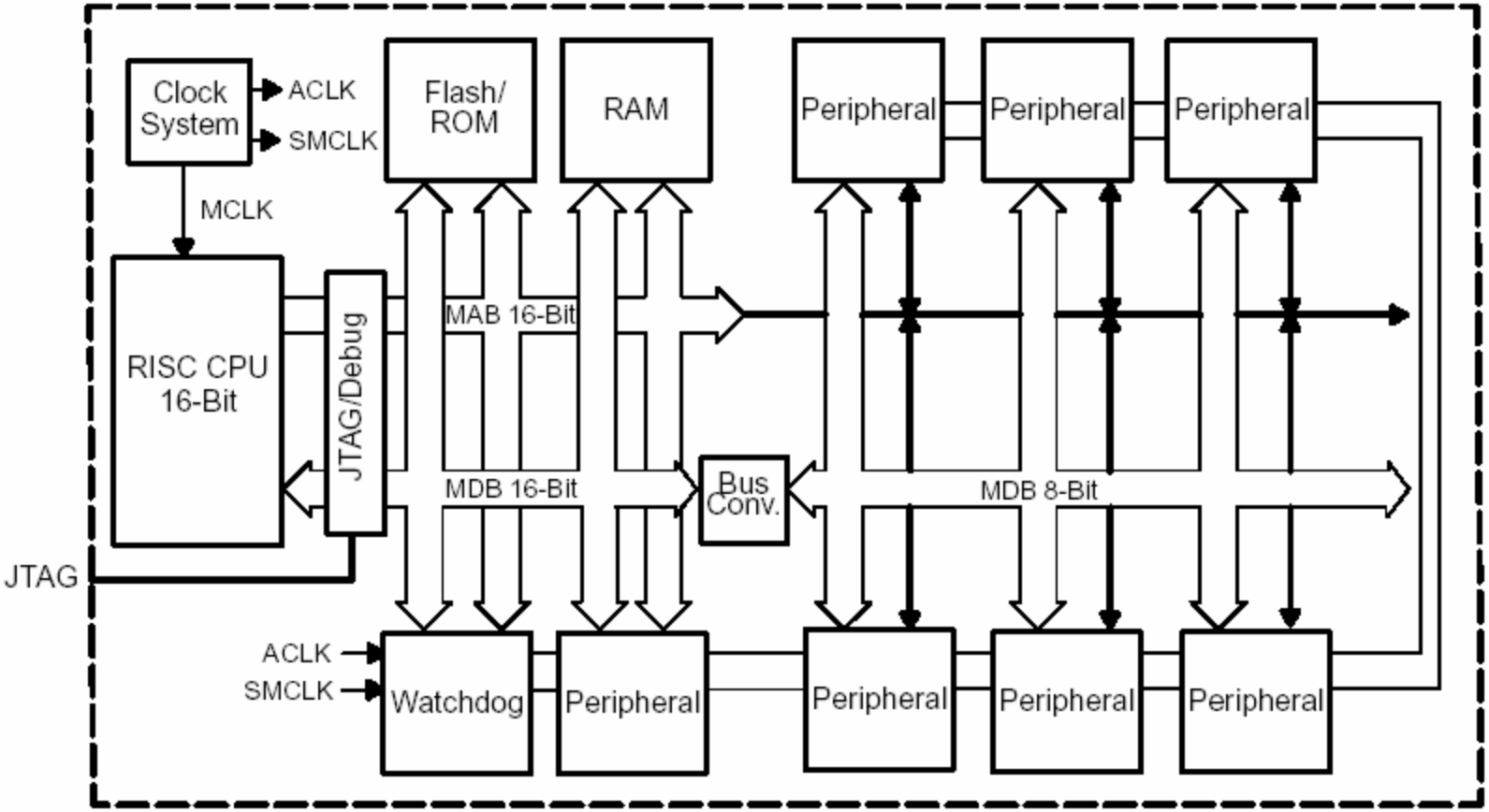


Memory Organization

- Word alignment
  - Bytes are located at even or odd addresses
  - Words are only located at even addresses
- Endianess (little-endian)
  - When using word instructions, only even addresses may be used. The low byte of a word is always an even address.
  - The high byte is at the next odd address.
  - For example, if a data word is located at address xxx4h, then the low byte of that data word is located at address xxx4h, and the high byte of that word is located at address xxx5h.

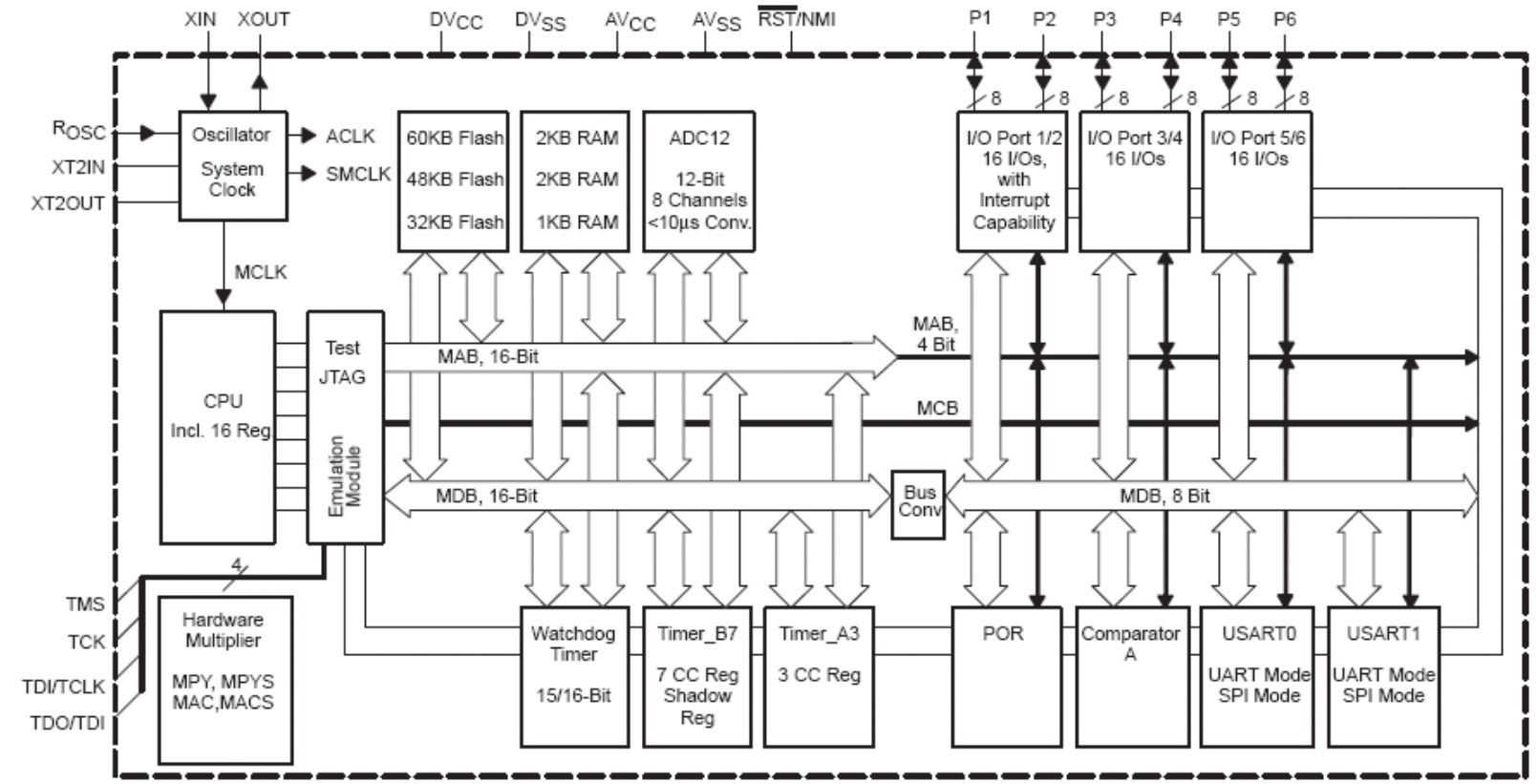


MSP 430 System Architecture: A Closer Look



MSPx430x14x Architecture

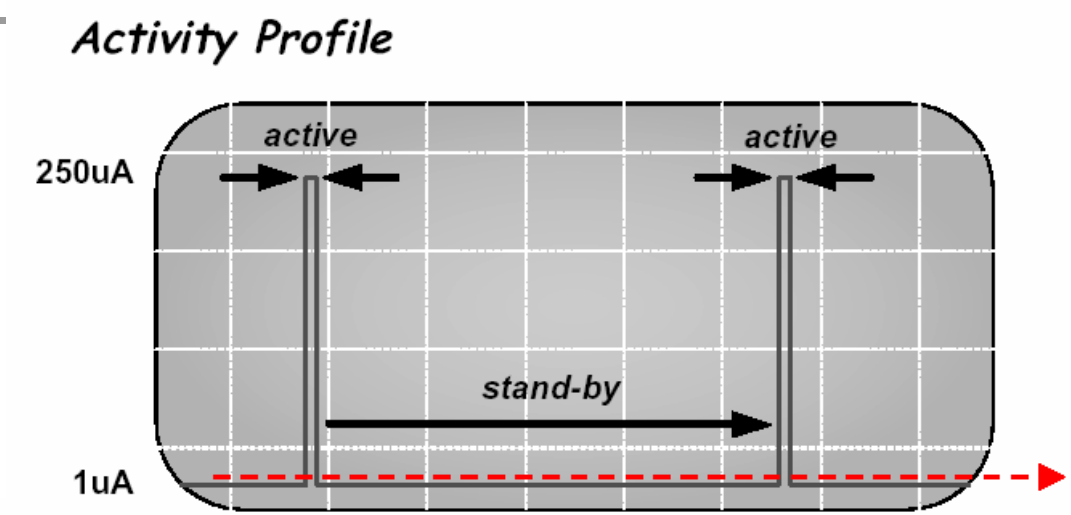
MSP430x14x

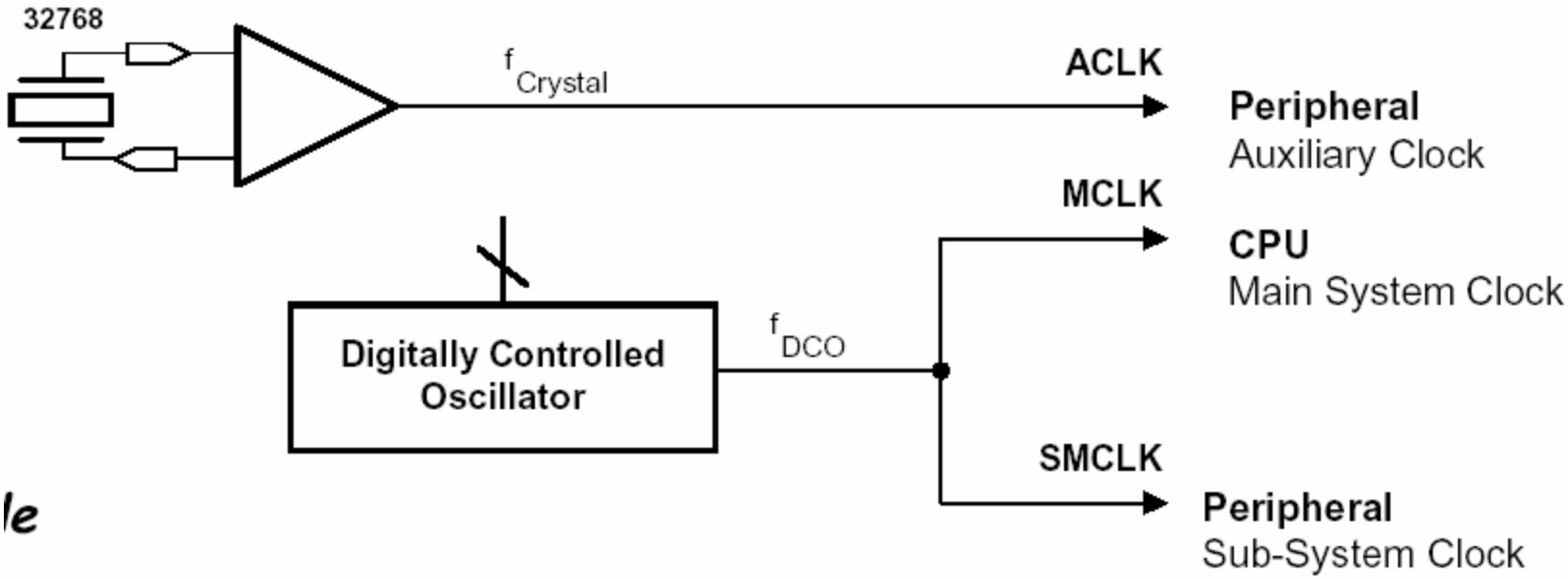


64 TQFP (The The Thin Quad Flat Pack package

Basic Clock System

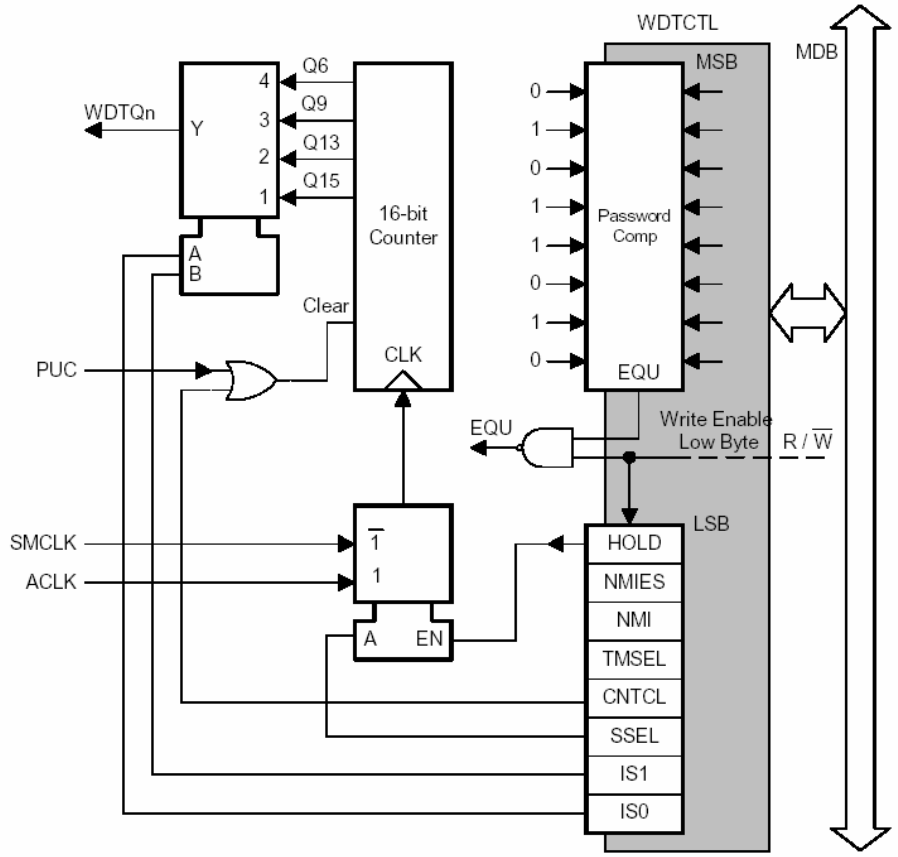
Basic Clock Module provides the clocks for the MSP430 processor and peripherals





Watchdog Timer

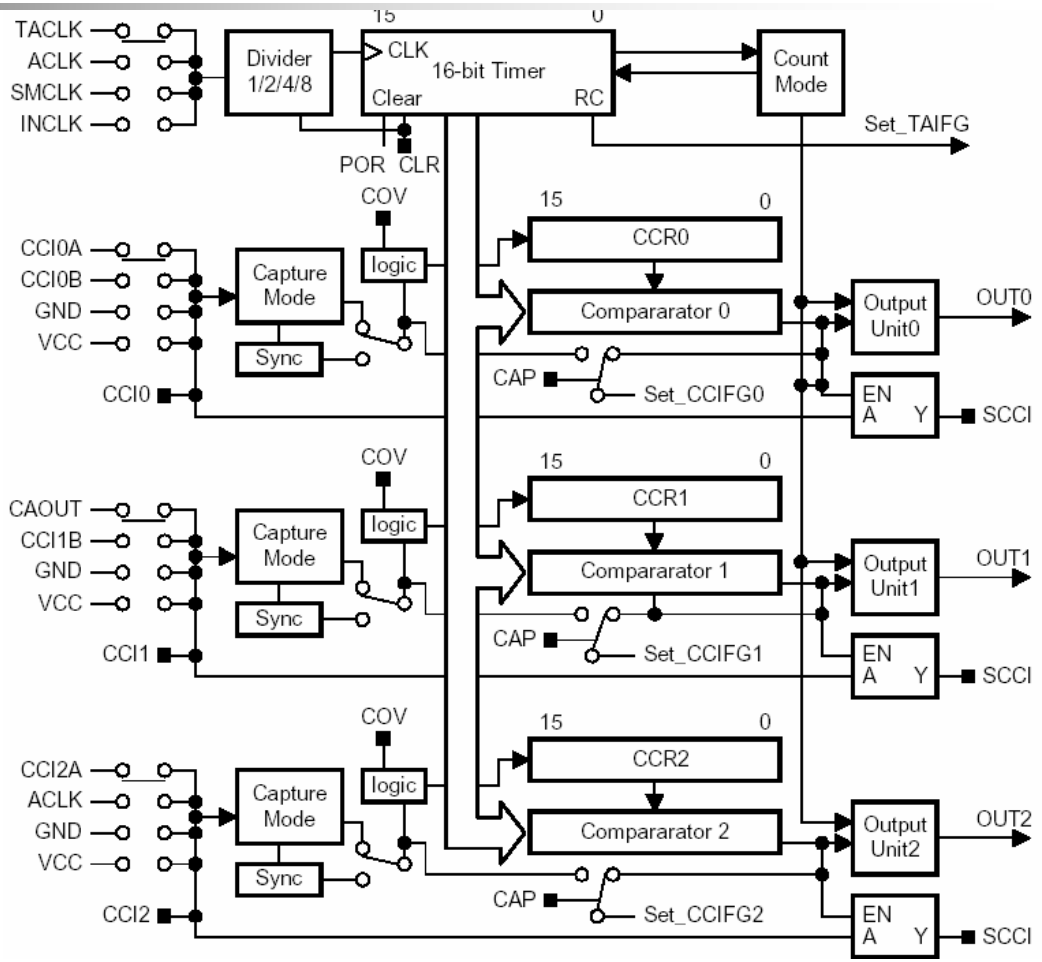
WDT module performs a controlled system restart after a software problem occurs



- Can serve as an interval timer (generates interrupts)
- WDT Control register is password protected
- Note: Powers-up active

Timer_A

Timer_A is a 16-bit timer/counter with three capture/compare registers

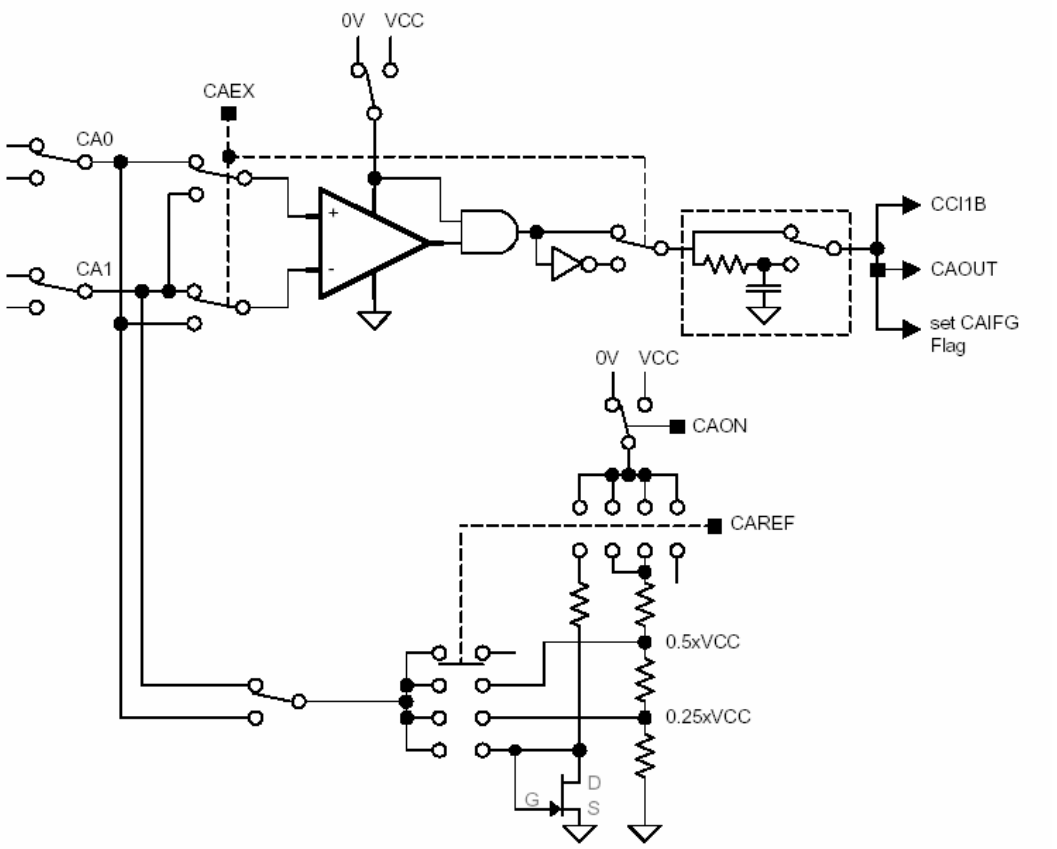


- Capture external signals
- Compare PWM mode
- SCCI latch for asynchronous communication

Comparator_A

Comparator_A is an analog voltage comparator

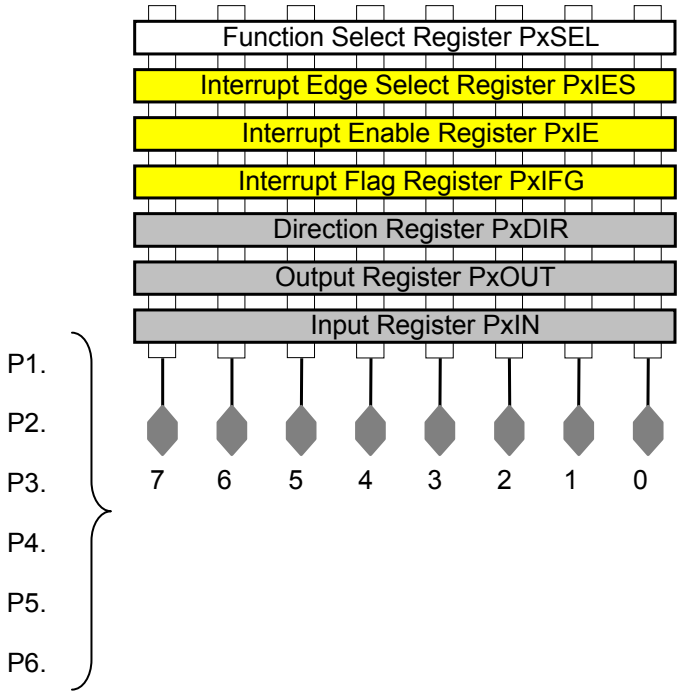
- Supports precision slope analog-to-digital conversions
- Supply voltage supervision, and
- Monitoring of external analog signals.



Digital I/O

Independently programmable individual I/Os

- Up to 6 ports (P1 – P6)
- Each has 8 I/O pins
- Each pin can be configured as input or output
- P1 and P2 pins can be configured to assert an interrupt request

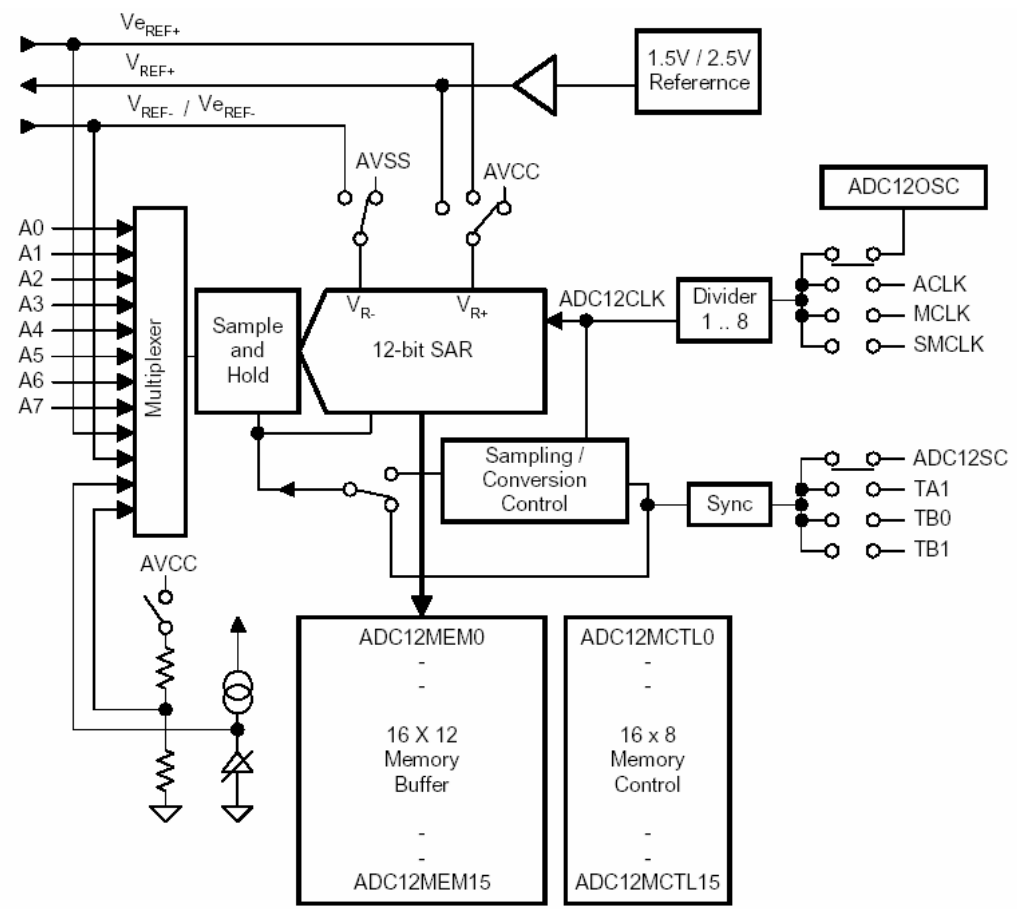


Port1 Port2	Port3 ... Port6
yes	yes
yes	no
yes	no
yes	no
yes	yes
yes	yes
yes	yes

ADC12

High-performance 12-bit analog-to-digital converter

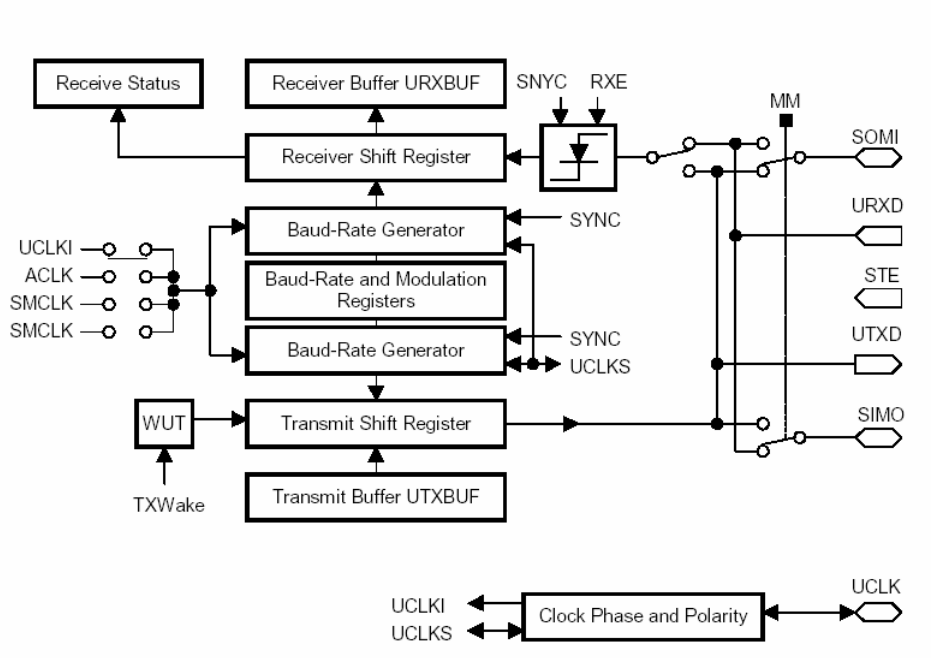
- More than 200 Ksamples/sec
- Programmable sample& hold
- 8 external input channels
- Internal storage



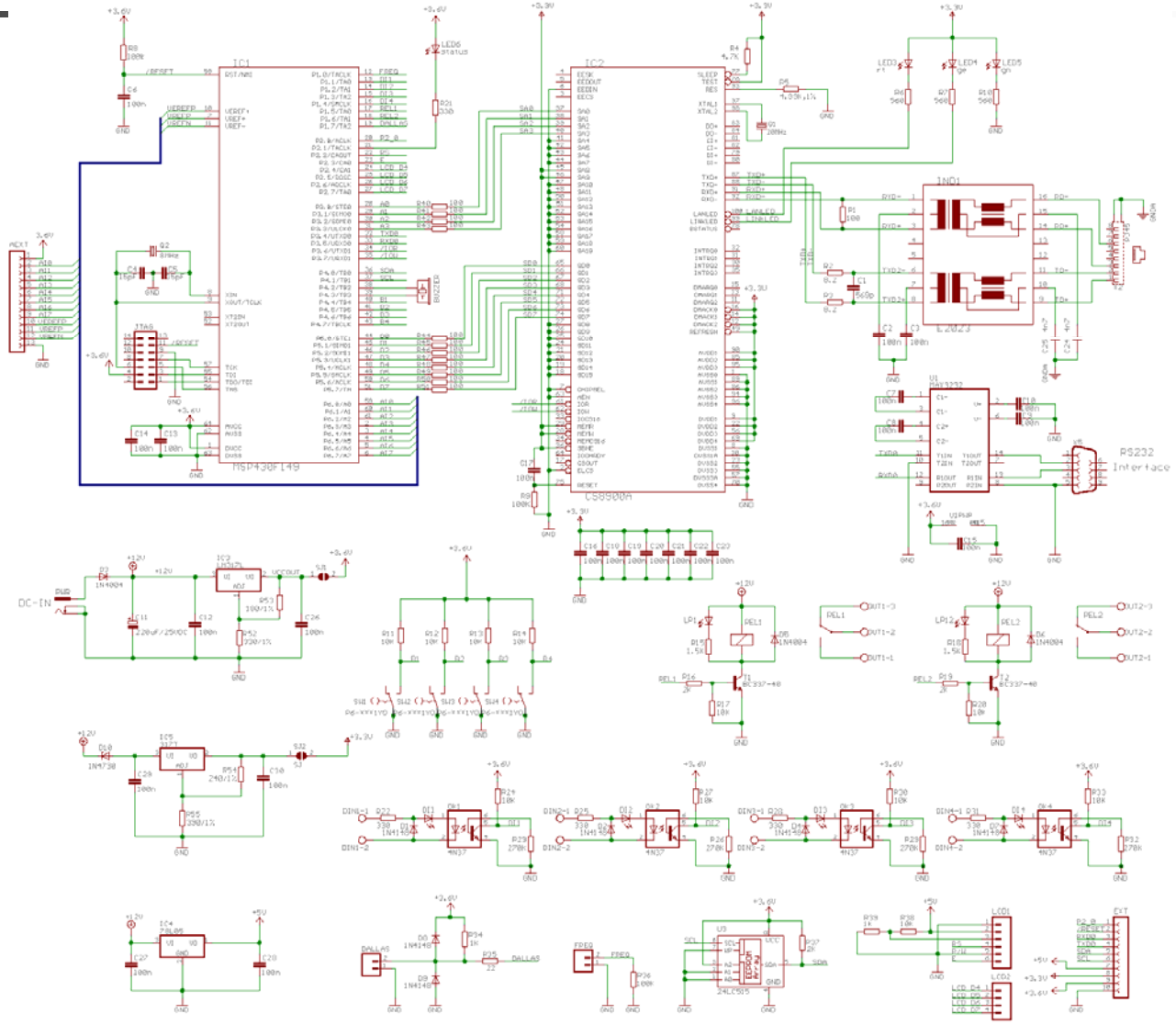
USART Serial Port

The universal synchronous/ asynchronous receive/transmit (USART) peripheral interface supports two serial modes with one hardware module

- UART or SPI (Synchronous Peripheral Interface) modes
- Double-buffered
- Baud-rate generator



Getting Started with EasyWeb2



Getting Started with EasyWeb2

```
//*************************************************
//  MSP-FET430P140 Demo - Software Toggle P2.1
//
//  Description; Toggle P2.1 by xor'ing P2.1
//  inside of a software loop.
//  ACLK = n/a, MCLK = SMCLK = default DCO ~
    800k
//
//                MSP430F149
//             -----------------
//         /|\|              XIN|-
//          | |                 |
//          --|RST          XOUT|-
//            |                 |
//            |             P2.1|-->LED
//
//  M. Buccini
//  Texas Instruments, Inc
//  January 2002
//  Built with IAR Embedded Workbench Version:
    1.25A
//
//  @Alex Milenkovich, milenkovic@computer.org
//  The University of Alabama in Huntsville
//  February 2005
//  Modified for easyWeb2 board to blink
//  the Status led (port P2.1)
//*************************************************
```

```
#include  <msp430x14x.h>

void main(void)
{
  // Stop watchdog timer
  WDTCTL = WDTPW + WDTHOLD;
  P2DIR |= 0x02;  // Set P2.1 to output directio
  for (;;)
  {
    unsigned int i;
    // Toggle P2.1 using exclusive-OR
    P2OUT ^= 0x02;
    i = 50000;     // Delay
    do (i--);
    while (i != 0);
  }
}
```