Computational hardware

- Digital logic (CSE370/351)
  - Gates and flip-flops: glue logic, simple FSMs, registers
  - Two-level PLDs: FSMs, muxes, decoders
- Programmable logic devices (CSE370/352, CSE467)
  - Field-programmable gate arrays: FSMs, basic data-paths
  - Mapping algorithms to hardware
- Microprocessors (CSE378/352)
  - General-purpose computer
  - Instructions can implement complex control structures
  - Supports computations/manipulations of data in memory

Microprocessors

- Arbitrary computations
  - Arbitrary control structures
  - Arbitrary data structures
  - Specify function at high-level and use compilers and debuggers
- Microprocessors can lower hardware costs
  - If function requires too much logic when implemented with gates/FFs
    - Operations are too complex, better broken down as instructions
    - Lots of data manipulation (memory)
  - If function does not require higher performance of customized logic
    - Ever-increasing performance of processors puts more and more applications in this category
    - Minimize the amount of external logic

Microprocessor basics

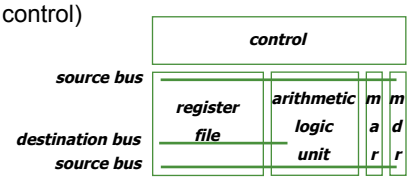
- Composed of three parts
  - Data-path: data manipulation and storage
  - Control: determines sequence of actions executed in data-path and interactions to be had with environment
  - Interface: signals seen by the environment of the processor
- Instruction execution engine: fetch/execute cycle
  - Flow of control determined by modifications to program counter
  - Instruction classes:
    - Data: move, arithmetic and logical operations
    - Control: branch, loop, subroutine call
    - Interface: load, store from external memory

Microprocessor basics (cont'd)

- Can implement arbitrary state machine with auxiliary data-path
  - Control instructions implement state diagram
  - Registers and ALUs act as data storage and manipulation
  - Interaction with the environment through memory interface
  - How are individual signal wires sensed and controlled?

Microprocessor organization

- Controller
  - Inputs: from ALU (conditions), instruction read from memory
  - Outputs: select inputs for registers, ALU operations, read/write to memory
- Data-path
  - Register file to hold data
  - Arithmetic logic unit to manipulate data
  - Program counter (to implement relative jumps and increments)
- Interface
  - Data to/from memory (address and data registers in data path)
  - Read/write signals to memory (from control)



General-purpose processor

- Programmed by user
- New applications are developed routinely
- General-purpose
  - Must handle a wide ranging variety of applications
- Interacts with environment through memory
  - All devices communicate through memory data
  - DMA operations between disk and I/O devices
  - Dual-ported memory (e.g., display screen)
  - Generally, oblivious to passage of time

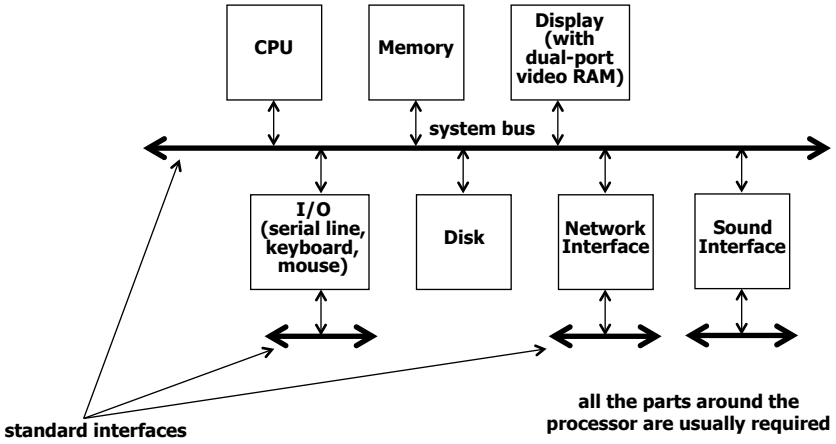
Embedded processor

- Typically programmed once by manufacturer of system
  - Many systems allow firmware updates
- Executes a single program (or a limited suite) with few parameters
- Task-specific
  - Can be optimized for a specific application
- Interacts with environment in many ways
  - Direct sensing and control of signal wires
  - Communication protocols to environment and other devices
  - Real-time interactions and constraints
  - Power-saving modes of operation to conserve battery power

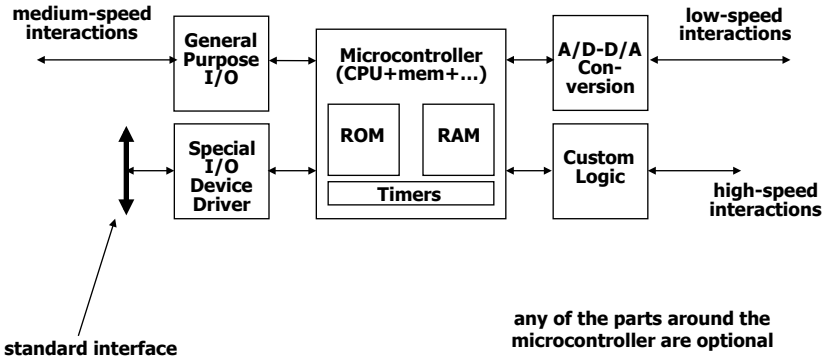
Why embedded processors?

- High overhead in building a general-purpose system
  - Storing/loading programs
  - Operating system manages running of programs and access to data
  - Shared system resources (e.g., system bus, large memory)
  - Many parts
    - Communication through shared memory/bus
    - Each I/O device often requires its own separate hardware unit
- Optimization opportunities
  - As much hardware as necessary for application
    - Cheaper, portable, lower-power systems
  - As much software as necessary for application
    - Doesn't require a complete OS, get a lot done with a smaller processor
  - Can integrate processor, memory, and I/O devices on to a single chip

Typical general-purpose architecture



Typical task-specific architecture



How does this change things?

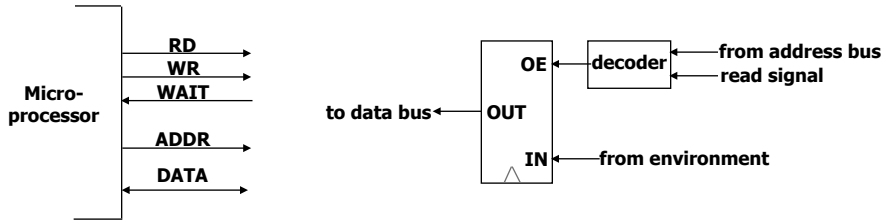
- Sense and control of environment
  - Processor must be able to “read” and “write” individual wires
  - Controls I/O interfaces directly
- Measurement of time
  - Many applications require precise spacing of events in time
  - Reaction times to external stimuli may be constrained
- Communication
  - Protocols must be implemented by processor
  - Integrate I/O device or emulate in software
  - Capability of using external device when necessary

Interactions with the environment

- Basic processor only has address and data busses to memory
- Inputs are read from memory
- Outputs are written to memory
- Thus, for a processor to sense/control signal wires in the environment they must be made to appear as memory bits
  - How do we make wires look like memory?

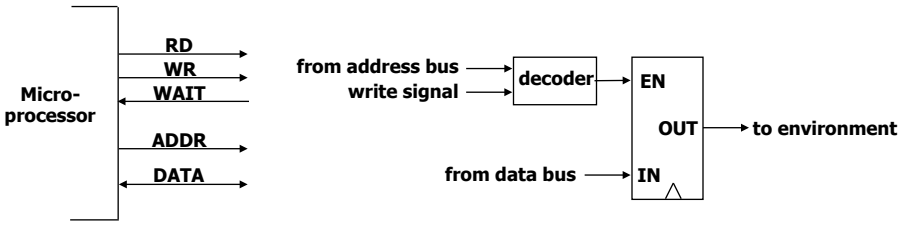
Sensing external signals

- Map external wire to a bit in the address space of the processor
- External register or latch buffers values coming from environment
  - Map register into address space
    - Decoder selects register for reading
  - Output enable (OE) to get value on to data bus
    - Lets many registers use the same data bus



Controlling external signals

- Map external wire to a bit in the address space of the processor
- Connect output of memory-mapped register to environment
  - Map register into address space
    - Decoder selects register for writing (holds value indefinitely)
  - Input enable (EN) to take value from data bus
    - Lets many registers use the same data bus



Time and instruction execution

- Keep track of detailed timing of each instruction's execution
  - Highly dependent on code
  - Hard to use compilers
  - Not enough control over code generation
  - Interactions with caches/instruction-buffers
- Loops to implement delays
  - Keep track of time in counters
  - Keeps processor busy counting and not doing other useful things
- Timer
  - Take differences between measurements at different points in code
  - Keeps running even if processor is idle to save power
  - An independent “co-processor” to main processor

Time measurement via parallel timers

- Separate and parallel counting unit(s)
  - Co-processor to microprocessor
  - Does not require microprocessor intervention
  - May be a simple counter or a more featured real-time clock
  - Alarms can be set to generate interrupts
- More interesting timer units
  - Self reloading timers for regular interrupts
  - Pre-scaling for measuring larger times
  - Started by external events

Input/output events

- Input capture
  - Record time when input event occured
  - Can be used in later handling of event
- Output compare
  - Set output event to happen at a point in the future
  - Reactive outputs
    - e.g., set output to happen a pre-defined time after some input
  - Processor can go on to do other things in the meantime

System bus based communication

- Extend address/data bus outside of chip
- Use specialized devices to implement communication protocol
- Map devices and their registers to memory locations
- Read/write data to receive/send buffers in shared memory or device
- Poll registers for status of communication
- Wait for interrupt from device on interesting events
  - Send completed
  - Receive occurred

Support for communication protocols

- Built-in device drivers
  - For common communication protocols
    - e.g., RS232, IrDA, USB, Bluetooth, etc.
  - Serial-line protocols most common as they require fewer pins
- Serial-line controller
  - Special registers in memory space for interaction
  - May use timer unit(s) to generate timing events
    - For spacing of bits on signal wire
    - For sampling rate
- Increase level of integration
  - No external devices
  - May further eliminate need for shared memory or system bus

Microcontrollers

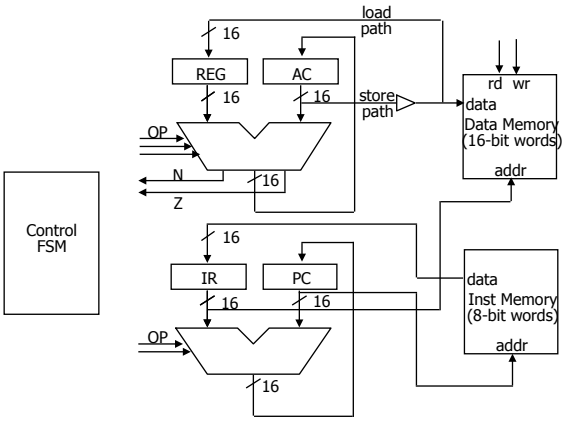
- Embedded processor with much more integrated on same chip
  - Processor core + co-processors + memory
  - ROM for program memory, RAM for data memory, special registers to interface to outside world
  - Parallel I/O ports to sense and control wires
  - Timer units to measure time in various ways
  - Communication subsystems to permit direct links to other devices

Microcontrollers (cont'd)

- Other features not usually found in general-purpose CPUs
  - Expanded interrupt handling capabilities
    - Multiple interrupts with priority and selective enable/disable
    - Automatic saving of context before handling interrupt
    - Interrupt vectoring to quickly jump to handlers
  - More instructions for bit manipulations
    - Support operations on bits (signal wires) rather than just words
- Integrated memory and support functions for cheaper system cost
  - Built-in EEPROM, Flash, and/or RAM
  - DRAM controller to handle refresh
  - Page-mode support for faster block transfers

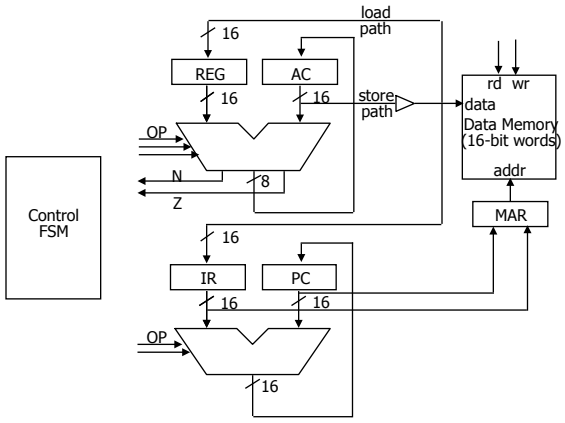
Block diagram of processor (Harvard)

- Register transfer view of Harvard architecture
  - Separate busses for instruction memory and data memory



Block diagram of processor (Princeton)

- Register transfer view of Princeton architecture
  - Single unified bus for instructions, data, and I/O



The MSP430: Introduction

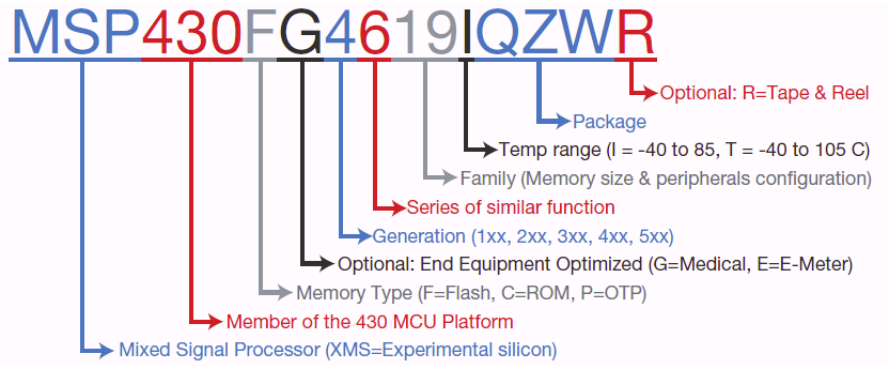
MSP430: An Introduction

- The MSP430 family
- Technology Roadmap
- Typical Applications
- The MSP430 Documentation
- MSP430 Architecture
- MSP430 Devices
- MSP430 RISC core

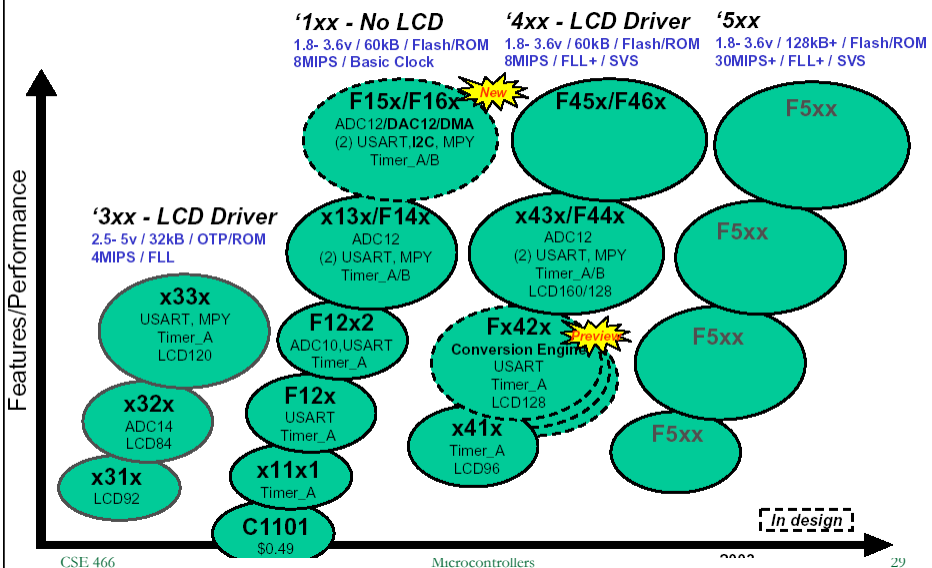
The Family

- Broad family of TI's 16-bit microcontrollers
  - from 1Kbytes ROM, 128 bytes RAM (approx. $1)
  - to 256Kbytes ROM, 16Kbytes RAM ($10)
- Many subfamilies
  - MSP430x1xx: Flash/ ROM based MCUs offer 1.8V to 3.6V operation, up to 60kB, 8MIPs with Basic Clock.
  - **MSP430F2xx:** 16 MHz. integrated on-chip oscillator, internal pullup/pull-down resistors
  - MSP430x4xx: 120kB/ Flash/ ROM 8MIPS with FLL + SVS, integrated LCD controller
  - MSP430x5xx: 25 MIPS, 1.8 to 3.6V, Power Management Module for optimizing power consumption, 2x memory

Part numbering convention



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)

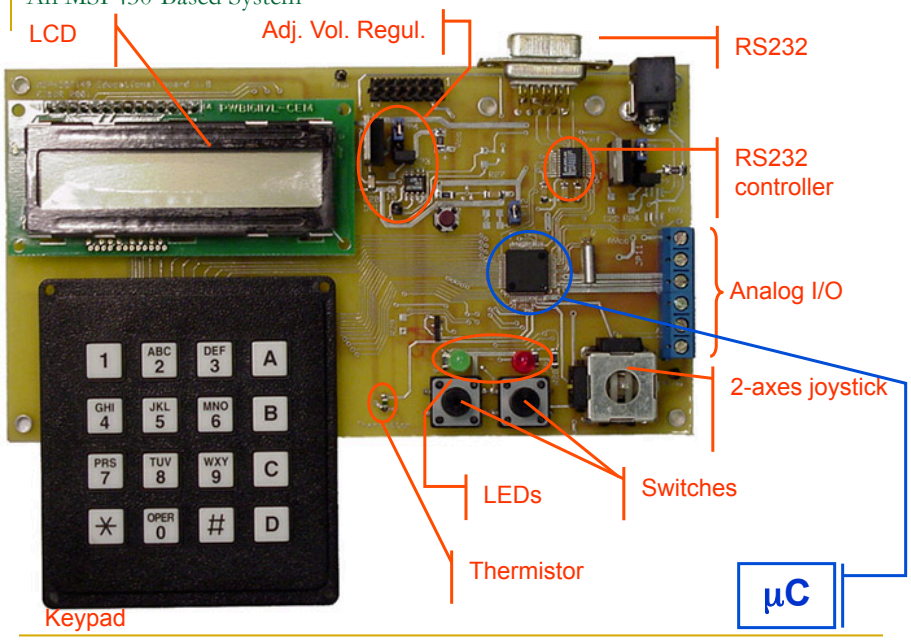
Sports equipment
- Altimeter
- Bike computer
- Diving watches

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,..)

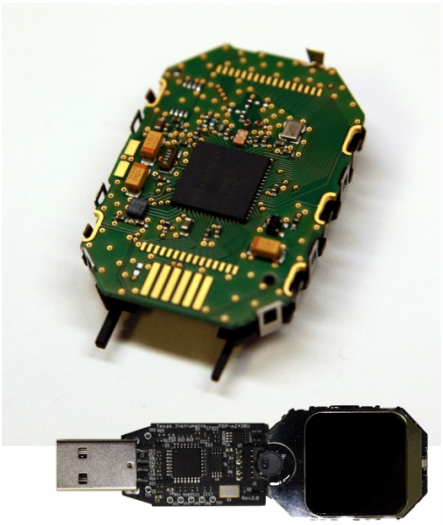
Misc
- Smart card reader
- Taxi meter
- Smart Batteries

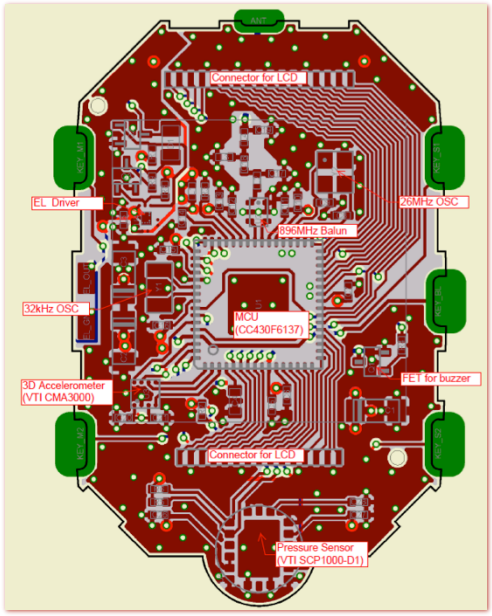
An MSP430-Based System



Another MSP430-Based System

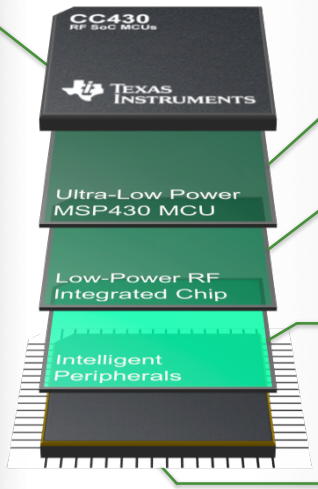
Chronos | Teardown





10/4/10

CC430 | Low-Power RF + Ultra-Low Power MCU



MSP430™ Microcontroller
- Industry's lowest power MCU
- 16-bit RISC architecture
- 27 MHz processor
- High-performance analog
- Sensor interface

CC1101 RF Transceiver SoC
- High sensitivity
- Low current consumption
- Excellent blocking performance
- Flexible data rate & modulation format

Intelligent Peripherals
- 100 nA comparator
- 8ch 12-bit ADC offering 200-ksps
- 96 segment LCD controller
- 128-bit AES security encryption/ decryption coprocessor

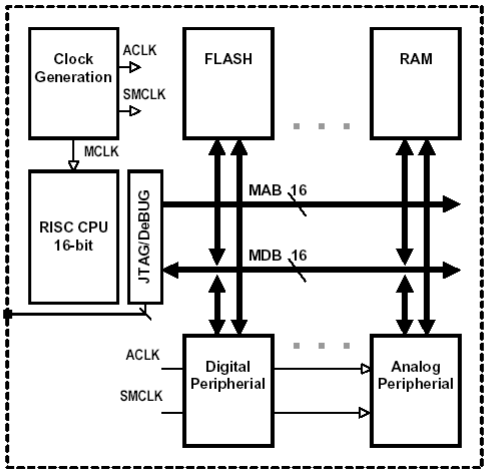
64QFN Pin Package
- 9.1 mm x 9.1 mm area

MSP430 Documentation

- MSP430 home page (TI)
  - www.ti.com/msp430
- User's manual (MSP430x2xx Family)
  - http://www.ti.com/litv/pdf/slau144e
- Datasheet
  - http://www.ti.com/lit/gpn/msp430f2013
- Chronos:
  - http://www.ti.com/lit/pdf/slau292

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

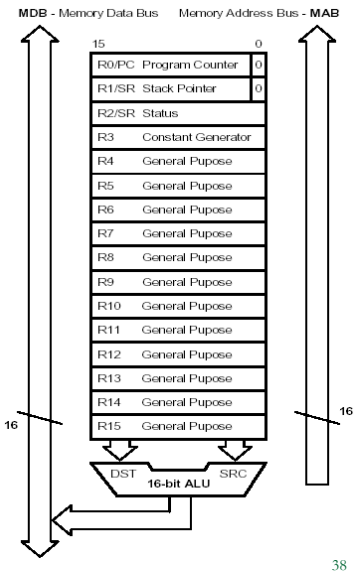
16-bit bus handles wide-width data much more effectively

CPU Introduction

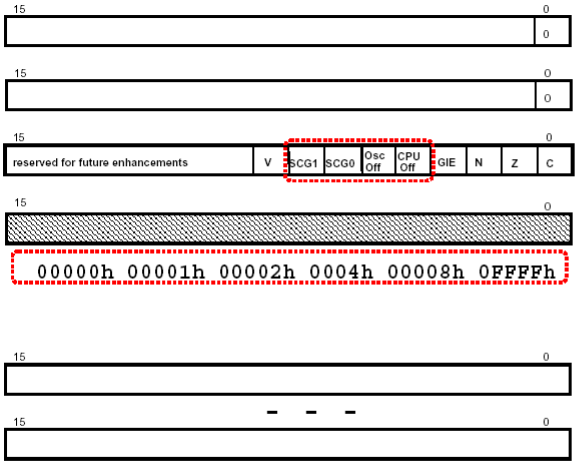
- RISC architecture with 27 instructions and 7 addressing modes.
- Orthogonal architecture with every instruction usable with every
- addressing mode.
- Full register access including program counter, status registers, and stack pointer.
- Single-cycle register operations.
- Large 16-bit register file reduces fetches to memory.
- 16-bit address bus allows direct access and branching throughout entire memory range.
- 16-bit data bus allows direct manipulation of word-wide arguments.
- Constant generator provides six most used immediate values and reduces code size.
- Direct memory-to-memory transfers without intermediate register holding.
- Word and byte addressing and instruction formats.

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



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.

Registers: PC (R0)

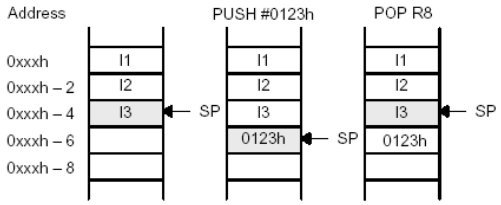
- Each instruction uses an even number of bytes (2, 4, or 6)
- PC is word aligned (the LSB is 0)

```
MOV #LABEL,PC ; Branch to address LABEL
MOV LABEL,PC ; Branch to address contained in LABEL
MOV @R14,PC ; Branch indirect, indirect R14
```

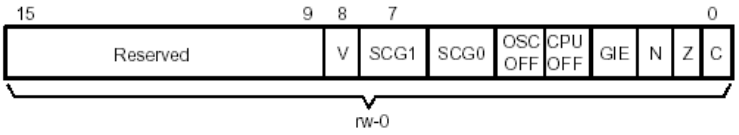
Registers: SP (R1)

- Stack pointer for return addresses of subroutines and interrupts
- SP is word aligned (the LSB is 0)
- Pre-decrement/post-increment scheme

```
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
```



Registers: SR (R2)



- C: SR(0)
- Z: SR(1)
- N: SR(2)
- GIE (Global interrupt enable): SR(3)
- CPUOff: SR(4)
- OSCOff: SR(5)
- SCG1, SCG0: SR(7), SR(6)
- V: SR(8)

Status bits

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 Generators

- As – source register addressing mode in the instruction word

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

CISC / RISC Instruction Set

- **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

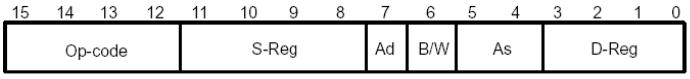
- Simply easier to understand with no code size or speed penalty
- Replaced by assembler with core instructions using CG, PC and SP

```
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

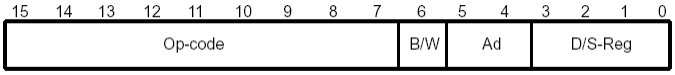
Format I Source,Destination	Format II Single Operand	Format III +/- 9bit Offset	Support
add(.b)	br	jmp	clrc
addc(.b)	call	jc	setc
and(.b)	swpb	jnc	clrz
bic(.b)	sxt	jeq	setz
bis(.b)	push(.b)	jne	clrn
bit(.b)	pop.(b)	jge	setn
cmp(.b)	rra(.b)	jl	dint
dadd(.b)	rrc(.b)	jn	eint
mov(.b)	inv(.b)		nop
sub(.b)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

Double operand instructions



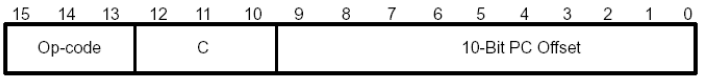
Mnemonic	S-Reg, D-Reg	Operation	Status Bits			
			V	N	Z	C
MOV(.B)	src,dst	src → dst	–	–	–	–
ADD(.B)	src,dst	src + dst → dst	*	*	*	*
ADDC(.B)	src,dst	src + dst + C → dst	*	*	*	*
SUB(.B)	src,dst	dst + .not.src + 1 → dst	*	*	*	*
SUBC(.B)	src,dst	dst + .not.src + C → dst	*	*	*	*
CMP(.B)	src,dst	dst – src	*	*	*	*
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	*	*	*

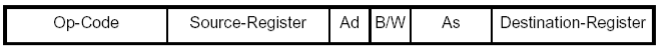
Jump Instructions



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

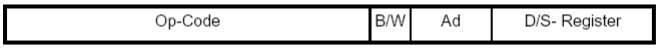
3 Instruction Formats

; Format I Source and Destination



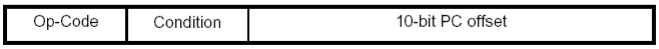

```
5405          add.w   R4,R5              ; R4+R5=R5 xxxx
5445          add.b   R4,R5              ; R4+R5=R5 00xx
```

; 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
```

Addressing Modes

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-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

```
4090 ffa80006   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

```
4292201720174   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

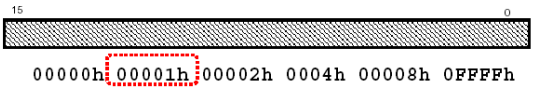
```
40351234      mov.w   #1234h,R5        ; Any 16-bit value
```

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.

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/III Instructions

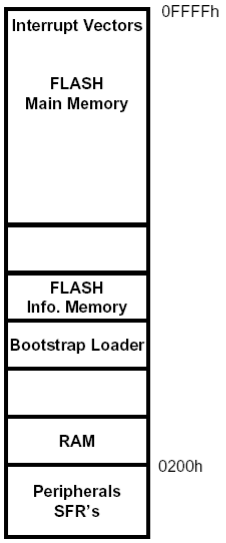
Address Mode As/Ad	#-of-Cycles: RRA RRC SWPB SXT	#-of-Cycles: PUSH/ CALL	Length of Instruction [words]	Examples
00, Rn	1	3/4	1	SWPB R5
01, x(Rn) 01, EDE	4	5/5	2	CALL Table(R7) PUSH EDE
10, @Rn	3	4/4	1	RRC @R9
11, @Rn+ 11, #N	3	4/5	1 2	SWPB @R10+ CALL 2(R7)

Machine Cycles for Format III Instructions

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

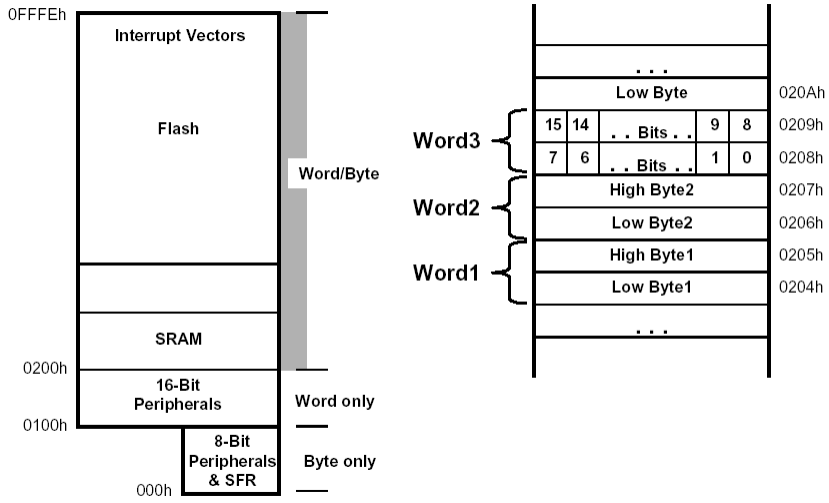
- Clock Cycles: 2
- Length of Instruction: 1 word

MSP430 Memory Model

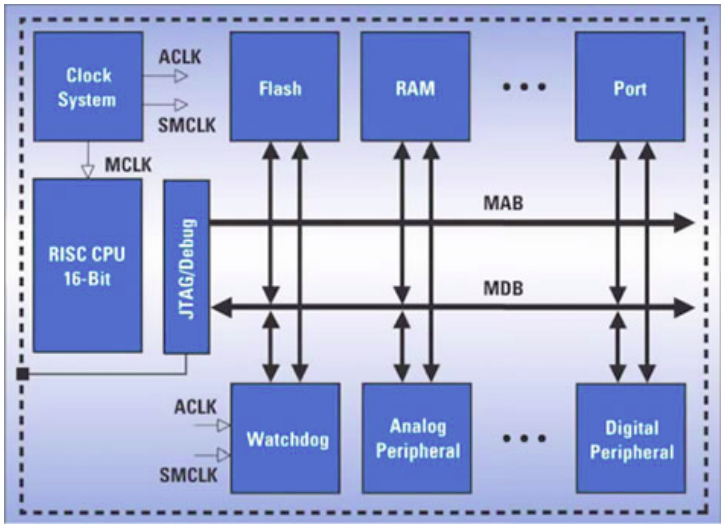


- Unified 64kB continuous memory map
- Same instructions for data and peripherals
- Program and data in Flash or RAM with no restrictions
- Easy to understand with no paging
- Designed for modern programming techniques such as pointers and fast look-up tables

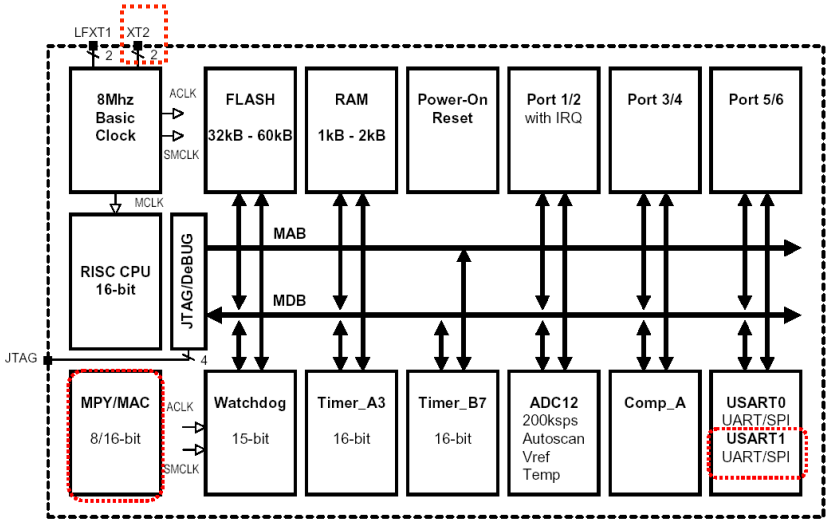
Memory Organization



MSP 430 Architecture: A Closer Look



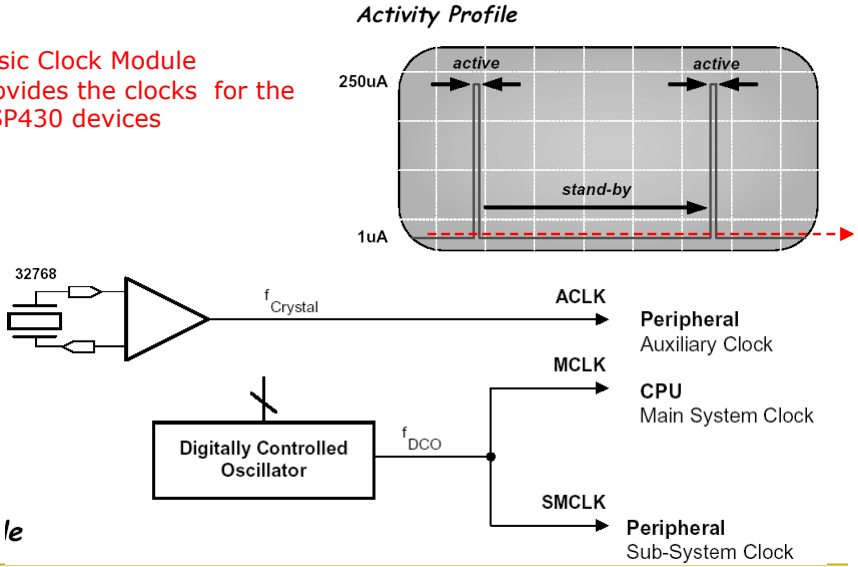
MSPx430x14x Architecture



64 TQFP (The The Thin Quad Flat Pack package

Basic Clock System

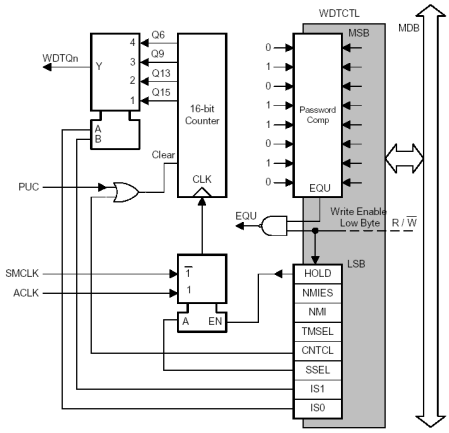
Basic Clock Module provides the clocks for the MSP430 devices



Watchdog Timer

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

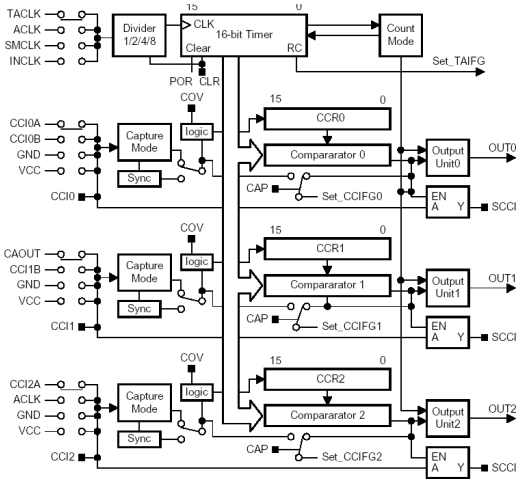
- Can serve as an interval timer (generates interrupts)
- WDT Control register is password protected
- Note: Powers-up active



Timer_A

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

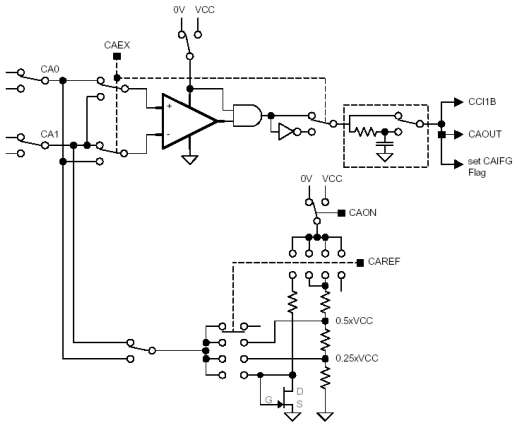
- Capture external signals
- Compare PWM mode
- SCCI latch for asynchronous communication



Comparator_A

Comparator_A is an analog voltage comparator

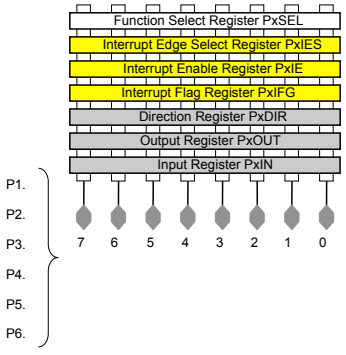
- Supports precision slope analog-to-digital conversions
- Supply voltage supervision, and
- Monitoring of external analog signals.



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

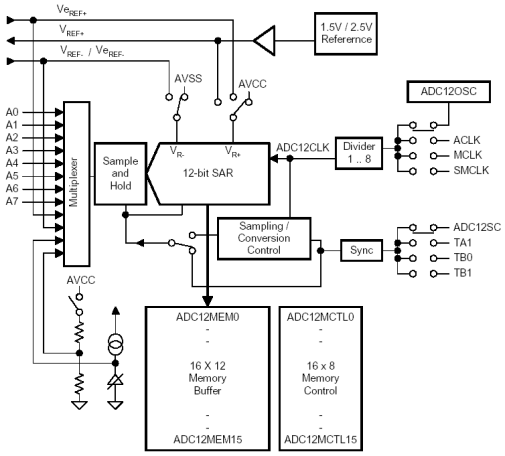


Port1 Port2	Port3 … Port6
yes	yes
yes	no
yes	no
yes	no
yes	yes
yes	yes
yes	yes

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