MSP430 Family
Architecture Guide and Module Library

Contents

Topic **Page**

Figures

Fig. Title Page

Fig. Title Page

Fig. Title Page

Fig. Title Page

Tables

List of Notes

Purpose of guide, and conventions used

The MSP430 User's Guide is intended to assist the development of MSP430 family products by sssemling together and presenting hardware and software information in a manner which will be easy to use by engineers and programmers.

There follows a short description of the nomenclature conventions used for signals and processor states:

- ADC Analog-to-Digital converter
- CPUOff mode Low power mode with RAM contents and I/O signals unchanged
  Modules using auxiliary clock (32 768 Hz crystal) are active
- DCO Digital controlled oscillator
- LCD Liquid crystal display
- FF Flip-Flop
- MAB Memory address bus. This is the address bus between the individual modules. It can be any width from 16 bits to 4 bits. Together with the MS signal it defines the physical address.
- MDB Memory data bus. This is the data bus between the individual modules. It can be 8 bits or 16 bits wide.
- MS Module select. This is the pre-decoded address space. Together with the MAB it defines the physical address.

• MSFR	Module special function register. This is the pre-decoded address space (0h to 0Fh) of the special function registers.
• OSCOff mode	Lowest power mode. RAM contents and I/O signals are unchanged. The crystal oscillator has stopped
• OTP	One-time programmable
• POR	Power-on reset
• PUC	Power-up clea, "1" sets processor's start condition
• SAR	Successive approximation register
• SCI	Serial communication interface to handle synchronous and asynchronous protocols
• SCG	System clock generator
• SFR	Special function register
• SPI	Serial peripheral interface (widely used synchronous serial communication protocol)
• TBD	To be defined
• TOS	Top of stack
• UART	Universal asynchronous receive transmit (most commonly-used serial communication protocol)
• USART	Universal synchronous asynchronous receive transmit
• WD,WDT	Watchdog, Watchdog Timer

Bit Type Convention for Register Bit

- rw: read/write
- r: read only
- r0: read as '0'
- w: write only
- (w): no register bit implemented; writing a '1' will result in a pulse.
  The register bit is always read as '0'.
- -0,-1: condition after PUC
- -(0),-(1): condition after POR
- h0: cleared by hardware

Symbols

Operations

@	Register indirect addressing
&	Absolute address
-->	Data transfer direction
+	Addition

-	Subtraction
x	Multiplication
/	Division
.AND.	logical AND
.OR.	logical OR
.XOR.	logical Exclusive-OR
.NOT.	logical NOT

Register Symbols

R0 or PC	Register 0 or Program Counter
R1 or SP	Register 1 or Stack Pointer
R2 or SR/CG1	Register 2 or Status Register/Constant Generator 1
R3 or CG2	Register 3 or Constant Generator 2
R4 to R15	Working Register, general-purpose

Contents of Status Register

C	Carry or borrow
Z	Zero
N	Negative
CPUOff	CPU Off Bit
OscOff	System Oscillator Off Bit

GIE	General Interrupt Enable
SCG0	System Clock Generator, Control Bit 0
SCG1	System Clock Generator, Control Bit 1
V	Overflow

Others

=	Equal Sign
‡	Not Equal Sign
>, <,≥,≤	Comparison Signs
" "	ASCII Character inside
h	Hexadecimal Data
b	Binary Data
#	Immediate Data
E	Exponent
&	Absolute Address Mode Indicator

Assembler Directives

.equ	Equate command
.sect	section directive
.word	word data
.byte	byte data
;	comment indicator

1 MSP430 Family

This section discusses the features of the MSP430 family of controllers, having special capabilities for analog processing control. All family members are software compatible, allowing easy migration within the MSP430 family by maintaining a common software base, and common design expertise and development tools.

The concept of a CPU designed for various applications with a 16-bit structure is presented. It uses a "von-Neumann Architecture" and hence has RAM, ROM and all peripherals in one address space.

Topic **Page**

1.1 Features and Capabilities

- Up to 64K byte addressing space as needed, for allocation of ROM, RAM, EERAM and peripherals as needed. Future expansion to 1M byte is planned.
- No limitation of interrupt and subroutine levels due to stack processing
- Only 3 instruction formats. Strong orthogonality without any exception
- 1word/instruction is used, as far as possible
- Seven address modes in the source
- Four address modes in the destination
- External interrupt pins: extended use of Input/Output pins for interrupt capability
- Prioritized interrupts: simultaneously occurring interrupts are handled prioritized)
- Nested interrupt structure: interrupt routines may be interrupted by higher priority interrupts
- Memory mapped peripherals: all registers are in the modules - no RAM space is used
- USART on chip - see device configuration: separate interrupts for transmit and receive
- Timer with interrupt for event counter, timing generation, PWM, ...
- Watchdog
- ADC (10 bits or more) with 8 inputs and current source
- EPROM version (OTP)
- LCD-driver
- Stable processor frequency using a FLL and a clock crystal of 32,768 Hz

- Easy program development because of the orthogonal structure: all instructions with all addressing modes
- C-compiler development has started
- Modular design concept: modules are strictly memory mapped

1.2 System Key Features

- Ultra-low current consumption: CPUOff and OscOff modes
- Full operation down to 2.5 V
- System building blocks: LCD-Drive, A/D-Converter, I/O-Ports, UART, Watchdog Timer, EEPROM .... all on chip
- Only microcomputer mode; there is no microprocessor mode
- Ease of use
  The powerful and convenient instruction set allows fast software development.
- Software may run in RAM
  Programs loaded into the RAM via the UART or test paths..., can execute jobs under real-time conditions. This reduces test costs and calibration expenses.
- Every ROM/RAM mix is possible in the common address range of 64k byte
- High level language (HLL) programming capabilities
  Large register file (12 general purpose registers)

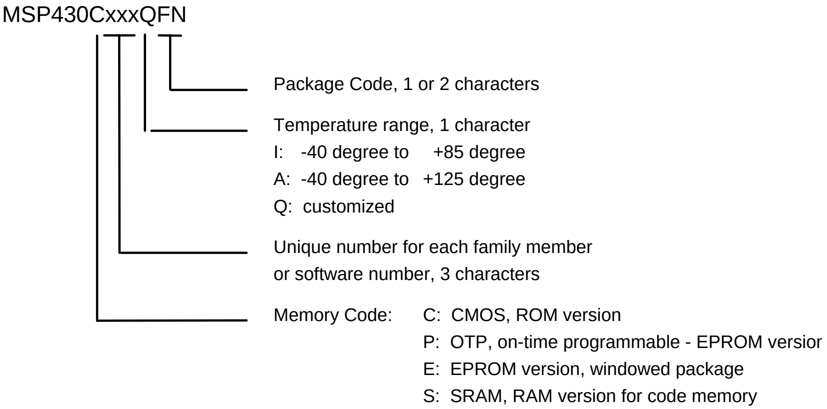
Stack orientation
Large ROM and RAM spaces
Orthogonal instruction set, without any exceptions
Table processing orientation, due to addressing modes

- Fast hexadecimal-to-decimal conversion with special instruction DADD
- Instructions are commonly used for ROM references, RAM access, data handling, I/Os and other peripherals: there are no special instructions!
- Potential of CPU far exceeds the requirements of intelligent sensor signal systems. The real-time capability opens fields in other low power systems, including the usage of other peripherals e.g. DTM transceiver for wired telecom

1.3 MSP430 Family Devices

The MSP430 family of devices can be summarized as follows:

- Nomenclature used:



MSP430CxxxQFN

Package Code, 1 or 2 characters

Temperature range, 1 character
I: -40 degree to +85 degree
A: -40 degree to +125 degree
Q: customized

Unique number for each family member
or software number, 3 characters

Memory Code: C: CMOS, ROM version
P: OTP, on-time programmable - EPROM versior
E: EPROM version, windowed package
S: SRAM, RAM version for code memory

- Development tools include the software simulator **DT430**, assembler and linker **ASM430/LNK430**, C-compiler (under development) **CS430/CW430**, and hardware in-circuit emulator **ICE430**. All development tools are PC-based using integrated desktop features compatible with the windows SAA standard.

The minimum requirements for the PC are:
IBM compatible
DOS 5.0 or later
Windows 3.1, 3.11 or ‘95
Personal computer with a 486 or higher processor running
8 MB of available memory
One 3.5" high-density disk drive
A hard disk with 5 MB available

	MSP430x310	MSP430x320	MSP430x330
Max. internal clock rate Frequency of crystal	1.1 MHz @3V 3.3 MHz @5V 32.768 kHz	1.1 MHz @3V 2.2 MHz @5V 32.768 kHz	1.1 MHz @3V 2.2 MHz @5V 32.768 kHz
Operating Temperature	-40^{o}C to +85^{o}C	-40^{o}C to +85^{o}C	-40^{o}C to +85^{o}C
Program memory MSP430Cxxx: MSP430Pxxx: MSP430Exxx: Memory expansion	 4/8/12k byte ROM 8K byte OTP 8K byte wind. EPROM NO	 8K byte ROM 16K byte OTP 16K byte wind. EPROM NO	 24K byte ROM 32K byte OTP 32K byte wind. EPROM NO
Internal RAM	256/512 Bytes	256 Bytes	1024 Bytes
Data EEPROM	No	No	No

Modules			
HW Multiply	No	No	Yes
Port0, 8-bit, all interrupt	Yes	Yes	Yes
Port1, 8-bit, all interrupt			Yes
Port2, 8-bit, all interrupt			Yes
Port3			Yes
Port4			Yes
Watchdog timer	Yes	Yes	Yes
Basic Timer1/Real time	Yes	Yes	Yes
clock	Yes	Yes	Yes
8-bit Timer/Counter	Yes	Yes	Yes
Timer/Port ,1x8-bit	Yes	Yes	Yes
Timer_A,16-bit	No	No	Yes
SPI	No	No	USART, SPI mode
UART	(8b Tim./Cnt. + SW)	(8b Tim./Cnt. + SW)	USART, UART mode or (8b Tim./Cnt. + SW)
LCD	Max. 23x4 segments	Max. 21x4 segments	Max. 30x4 segments
ADC/Current source	Yes/Yes	see Timer/Port	see Timer/Port
DAC	No	No	No
I/O lines	9	9	40
Input lines	1	7	1
Output lines	27	25	34

Interrupts/Reset			
External	11	11	1 + 24
Vectors total	16	16	16
Sources total			
Package Type	64 QFP	56 SSOP	100 QFP

Table 1.1: MSP430 Family Feature Summary

2 Architectural Overview

This section describes the basic functions of a MSP430 based system.

Topic **Page**

The MSP430 devices contain the following main functions:

- Central Processing Unit (CPU)
- Program Memory (ROM or EPROM)
- Data Memory (RAM or EEPROM)
- Control of operation
- Peripheral Modules
- Oscillator + Frequency Multiplier.

The architecture of the MSP430 family is based on a memory-to-memory architecture, a common address space for all functional blocks, and a reduced instruction set applicable for all functional blocks.

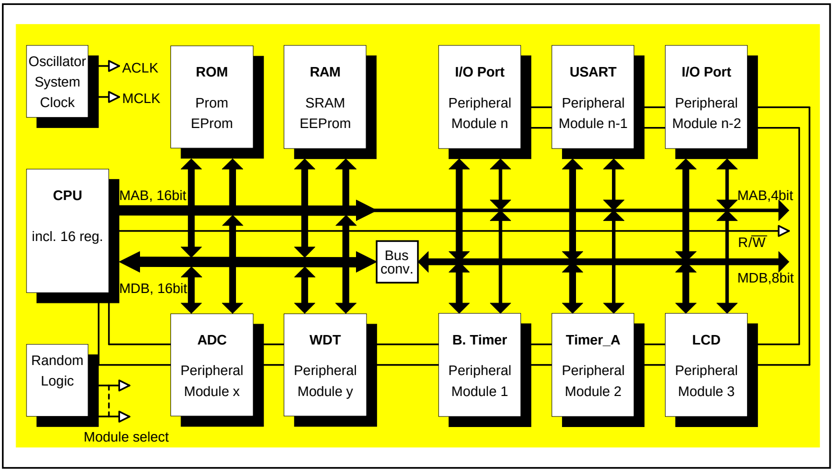


Figure 2.1: MSP430 system configuration

2.1 CPU

The central processing unit incorporates the following reduced, highly transparent instruction set, and a highly orthogonal design. It consists of a sixteen bit ALU, sixteen registers and an instruction control logic. Four of these registers are used for special purposes, these are the Program Counter PC, Stack Pointer SP, Status Register SR and Constant Generator CG2. All registers - except R3/CG2 and part of R2/CG1- can be used as general-purpose registers applying the complete instruction set to registers. The constant generator supplies constants for performing the instruction not for storing any data. The addressing mode used on CG1 separates the data of the constants.

The complete control over the Program Counter, the processor's Status Register and the Stack Pointer with the reduced instruction set, allows the development of applications with complex addressing modes or SW algorithms.

2.2 Code Memory

Access to the Code Memory is always word organized for fetching code, data can be read with word or byte access. Any access uses the 16-bit Memory Data Bus and as many of the least significant address lines as are needed to access the memory locations. Blocks of memory are automatically selected via Module Enable signals, this being a technique to reduce overall current consumption. Program memory can be integrated as programmable (EPROM) or mask programmable (ROM) memory. Standard members of the MSP430 family support OTP and mask programmed versions. Support of external memory will be the subject of future enhancements.

Sixteen words of memory are reserved for reset and interrupt vectors at the top of the lowest 64K byte address space from 0FFFFh down to 0FFE0h.

Access to Program Memory via software program is fully supported for read operation (MOV &0FFA0h,R5), but not for write (→ ROM).

Future enhancements:

The address space will be enhanced using segmented memory areas. The expanded addressable space is supported mainly using three extensions: branch and call long instructions, code segment pointer CSP and data pointer DPP. The code segment pointer is located within the status register SR. This enhanced address space is used for instruction codes (CSP + PC) and for data memory ([DPPi] + operand address) as follows:

MAB = CSP * 10000h + PC	during any access to code memory
MAB = DDPi * 4000h + Rs/d	during any access to stack or data memory

For basic devices using up to 64K byte addressing space, the content of CSP and DPP is unused by the Memory Address Bus.

2.3 Data Memory (RAM)

The Data Memory is connected to the CPU via two busses: the Memory Address Bus (MAB), and the Memory Data Bus (MDB). The Data Memory can be integrated into the specific family member either with full (word) data width or with reduced (byte) data width.

The entire instruction set operates fully on byte and word data. All operations on stack and PC are word operations, and should use only even aligned addresses.

2.4 Control of operation

The operations of the different MSP430 members are controlled mainly with the information stored in special function registers, SFRs. The different bits in the SFRs enable interrupts, support the software on the status of the interrupt flags and define the operating modes of the peripherals. Peripherals that are disabled stop their functional operation to reduce current consumption. All data stored in the module's register are retained. Peripherals that have their operating mode controlled can be identified in the specific sections.

2.5 Peripherals

Peripheral modules are connected to the CPU via Memory Address Bus MAB, Memory Data Bus MDB and interrupt service and request lines. The MAB is usually a 5-bit bus for most of the peripherals. The MDB is an 8-bit or 16-bit bus. Modules with an 8-bit data bus are connected via bus conversion circuitry to the 16-bit CPU. The data exchange with these modules should be handled with byte instructions, without exception. Instruction execution on word-oriented peripherals operates without any restrictions. Most of the peripherals are operating in byte format. The SFRs are handled within an 8-bit data range without any exception. The operation to 8-bit peripherals follows the orders described.

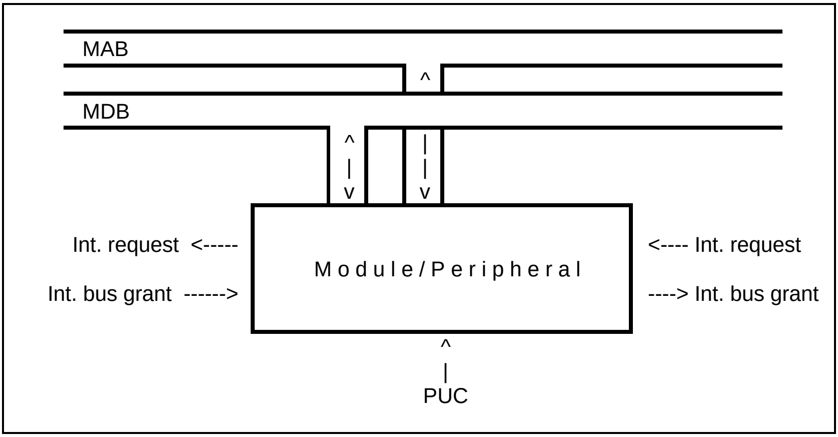


Figure 2.2: Bus connection of modules/peripherals

2.6 Oscillator, Frequency Multiplier and Clock Generator

The oscillator is specially designed for the commonly-used clock crystal of 32,768 Hz with low current consumption. All analog components are integrated; only the crystal has to be connected.

This oscillator is the direct source for some modules with low-frequency requirements. For the CPU and other modules, the crystal's frequency is multiplied by a first order frequency lock loop circuitry FLL. The FLL starts after power-up with its lowest possible frequency, and is regulated to the proper frequency by controlling a digital controlled oscillator DCO.

The long-term deviation is limited by the stability of the crystal and oscillator.

The frequency of the clock generator for the processor's operation is a fixed multiple of the crystal, and supports the clock MCLK.

3 System Reset, Interrupts and Operating Modes

Topic **Page**

3.1 System Reset & Initialization

The MSP430 has four possible reset sources: applying supply voltage to V_{CC} pin, a low input to the , RST/NMI pin, a programmable watchdog timer time-out and a security key violation during write access to WDTCTL register.

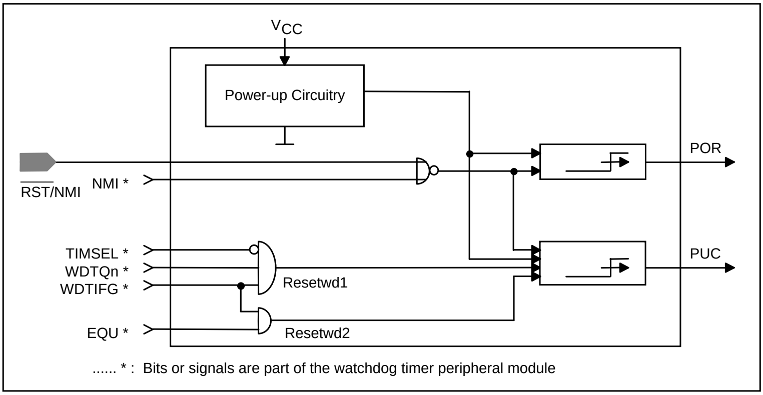


Figure 3.1: System Reset Functions

After the occurrence of a reset, the program can interrogate flags according to the reset source. The program can determine the source of reset in order to take appropriate action.

The MSP430 starts hardware initialization after applying V_{CC}:

- All I/O-pins are switched to the input direction
- The I/O-flags are cleared as described in the appropriate peripheral descriptions
- The address contained in the reset vector at word address 0FFFEh is placed into the Program Counter
  The CPU starts at the address contained in the power-up clear (PUC) vector.
- The status register SR is reset.
- All registers have to be initialized by the user's program (e.g., the Stack Pointer, the RAM, ...), except for PC and SR.
- Registers located in the peripherals are handled as described in the appropriate section.
- The frequency controlled system clock starts with the lowest frequency of the digital controlled oscillator. After the start of the crystal clock, the frequency is regulated to the target value.
  The , RST/NMI pin is configured with the reset function after applying V_{CC}. It remains reset as long as the reset function is selected. When the pin is configured with the reset function, the MSP430 starts operation after the , RST/NMI pin is pulled down to Gnd and released as follows:

- The address contained in the reset vector at word address 0FFFEh is placed into the Program Counter
- The CPU starts at the address contained in the reset vector after the release of the , RST/NMI pin.
- The status register SR is reset.
- All registers have to be initialized by the user's program (e.g., the Stack Pointer, the RAM, ...), except for PC and SR.
- Registers located in the peripherals are handled as described in the appropriate section.
- The frequency controlled system clock starts with the lowest frequency of the DCO. After the start of the crystal clock the frequency is regulated to the target value.

3.2 Global Interrupt Structure

There are three types of interrupts:

- System reset
- Non-maskable interrupts
- Maskable interrupts

Sources causing a system reset are:

• Applying supply voltage	@ POR, PUC
• 'low' on , RST/NMI (if reset mode selected)	@ POR, PUC
• Watchdog timer overflow (if watchdog mode selected)	@ PUC
• Watchdog timer security key violation	@ PUC
• (writing to WDTCTL with incorrect password)	

A non-maskable interrupt can be generated by:

- Edge on , RST/NMI-pin (if NMI mode selected)
- Oscillator fault

> **Note: Oscillator fault**
>
> The oscillator fault is maskable by an individual enable bit OFIE. It is not disabled during a general interrupt enable (GIE) reset.

Sources for maskable interrupts are:

- Watchdog timer overflow (if timer mode is selected)
- other modules having interrupt capability

MSP430 Interrupt Priority Scheme

The interrupt priority of the modules is defined by the arrangement of the modules in the connection chain: the nearer a module in the chain is towards the CPU/NMIRS, the higher is the priority.

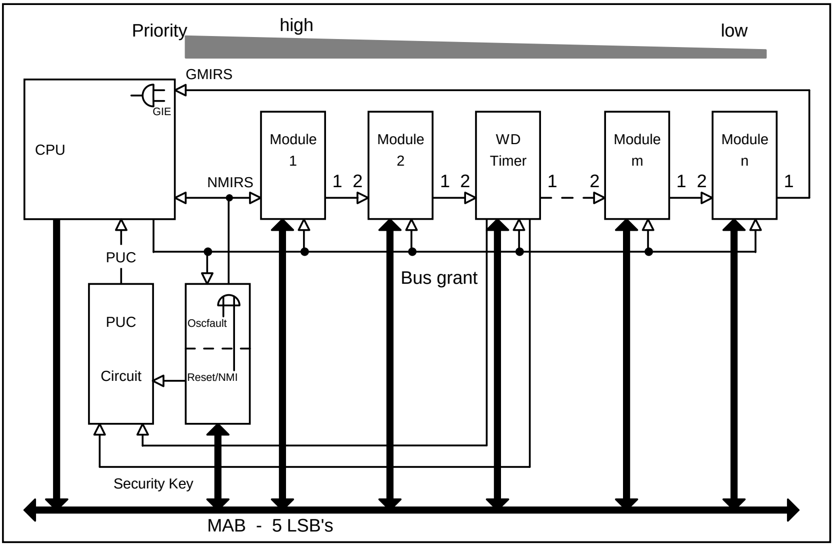


Figure 3.2: Interrupt Priority Scheme

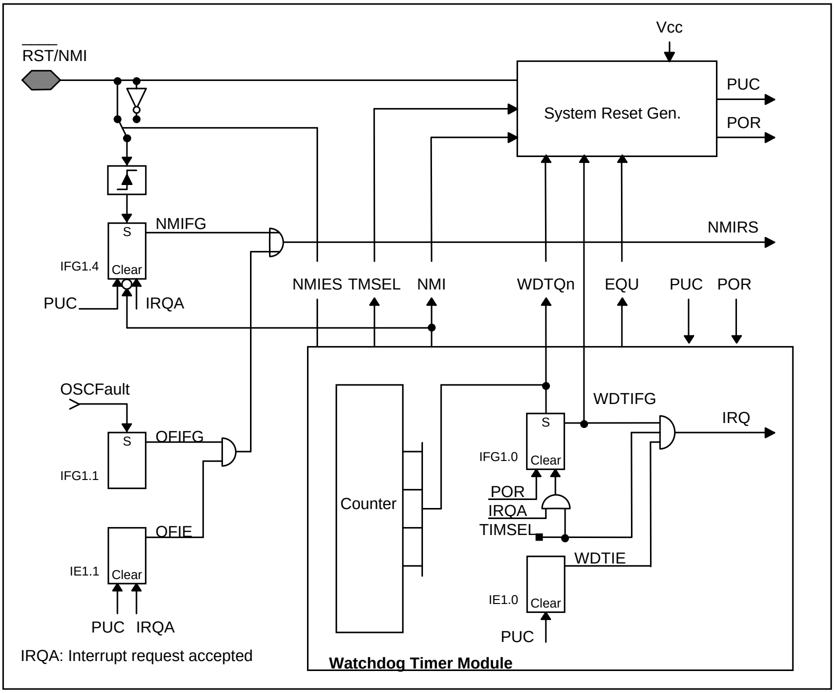


Figure 3.3: Reset/NMI-mode selection

Reset and NMI can be used only as alternatives, because they make use of the same input pin. The associated control bits are located in the Watchdog Timer Control register, and are also password protected.

WDTCTL 0120h	7							0
	HOLD	**NMIES**	**NMI**	TMSEL	CNTCL	SSEL	IS1	IS0
	rw-0	rw-0	rw-0	rw-0	(w)-0	rw-0	rw-0	rw-0

BIT 5: The NMI-Bit selects the function of the , RST/NMI-input pin. It is cleared after PUC.

NMI = 0: The , RST/NMI input works as Reset input.
As long as the , RST/NMI-pin is held 'low', the internal PUC-signal is active (level sensitive).

NMI = 1: The , RST/NMI input works as an edge-sensitive non-maskable interrupt input.

BIT 6: This bit selects the activating edge of the , RST/NMI input if NMI function is selected. It is cleared after PUC.

NMIES = 0: A rising edge triggers a NMI-interrupt.
NMIES = 1: A falling edge triggers a NMI-interrupt.

Operation of global interrupt - Reset/NMI

If the Reset function is selected, the CPU is held in the reset state as long as the , RST/NMI-pin is held 'low'. After the input has changed to high, the CPU starts program execution at the word address which is stored in word location 0FFFEh (Reset vector).

If the NMI function is selected, an edge according to the NMIES-bit generates an unconditional interrupt, and program execution is resumed at the address which is stored in location 0FFFCh. The , RST/NMI flag in the SFR (IFG1.4) is also set. It is automatically reset during interrupt request service. The , RST/NMI pin should never be held permanently 'low'. When a situation happens that activates the PUC, the consecutive reset of the bits in WDTCTL register forces the reset function on , RST/NMI pin. An continuous 'low' at , RST/NMI pin results in a permanent reset and system hold.

Note: NMI edge select

When NMI mode is selected and the NMI edge select bit is changed, an NMI can be generated, depending on the actual level at , RST/NMI pin.

When the NMI edge select bit is changed before selecting the NMI mode no NMI is generated.

Operation of global interrupt - Oscillator fault control

As described in the oscillator section, the FLL oscillator will continue to work even if the crystal is defective, but it will then run at the lowest possible frequency. The second limit is the highest possible frequency. Both cases are usually error conditions and must be detectable by the CPU. Therefore the oscillator fault signal can be enabled by SFR bit IE1.1 to generate an NMI interrupt. By testing the interrupt flag IFG1.1 in the SFR, the CPU can determine if the interrupt was caused by an oscillator fault.

Operation of global interrupt - Power-up-clear (PUC)

Three sources or events can initiate system reset:

- Power-up logic
- ,RST/NMI input

- Watchdog overflow.

Resets caused by , RST/NMI and the watchdog can be evaluated by software through testing the associated interrupt flag in SFR bit IFG1.0.

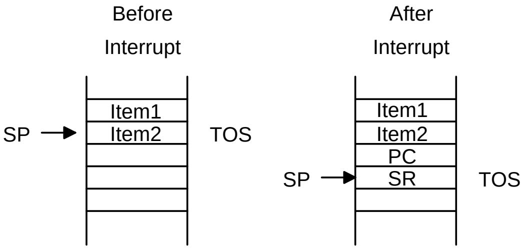
3.3 Interrupt Processing

The MSP430 programmable interrupt structure allows flexible on-chip and external interrupt configurations to meet real-time interrupt-driven system requirements. Interrupts may be initiated by the processor's operating conditions, such as watchdog overflow, peripheral modules or external events. Each interrupt source can be disabled individually by an interrupt enable bit or all interrupts are disabled by general interrupt enable bit GIE in the status register.

Whenever an interrupt is requested and the appropriate interrupt enable bit and the General Interrupt Enable Bit (GIE) is set, the interrupt service routine becomes active as follows:

- CPU active: The currently executed instruction is completed.
  CPU stopped: The low power modes are terminated.
- The Program Counter pointing to the next instruction is pushed onto the stack.
- The Status Register is pushed onto the stack.
- The interrupt with the highest priority is selected if multiple interrupts occurred during the last instruction and are pending for service.
- The appropriate interrupt requesting flag is reset automatically on single source flags. Multiple source flags remain set for servicing by software.
- The general interrupt enable bit GIE is reset;
  the CPUOff bit, the OscOff bit and the SCG1*) bit are cleared;
  the status bits V, N, Z and C are reset.
- The content of the appropriate interrupt vector is loaded into the Program Counter: The program continues with the interrupt handling routine at that address.

*) SCG0 is left unchanged, and FLL loop control remains in previous operating condition.



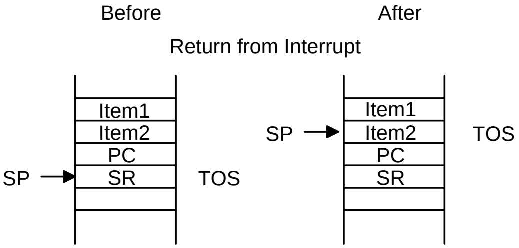
The interrupt latency is six cycles, starting with the acceptance of an interrupt request, and lasting until the start of execution of the first instruction of the appropriate interrupt service routine.

The interrupt handling routine terminates with the instruction:

```
RETI
```

which performs the following actions:

- The Status Register is popped from the stack.
  The interrupted software continues with exactly the same status as before the interrupt
  including OscOff, CPUOff and GIE bits.
  The GIE bit in the Status Register replaces the logical state present during interrupt service with the pushed state from TOS. It is set in any case, because it was set prior to accepting the interrupt.
- The Program Counter is popped from the stack.



The return from an interrupt service routine with the RETI instruction takes five cycles. Interrupt nesting is activated if the GIE-bit is set inside the interrupt handling routine.
The general interrupt enable bit GIE is located in the Status Register SR/R2 which is included in the CPU as follows:

15	8	7							0
reserved for future enhancements	V	SCG1	SCG0	Osc Off	CPU Off	GIE	N	Z	C
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Figure 3.4: Status Register SR

Apart from the GIE bit, other sources of interrupt requests can be enabled/disabled individually or in groups. The interrupt enable flags are located together within two addresses of the special function register SFR. The program flow conditions on interrupt requests can be easily adjusted by extensive use of the interrupt enable masks. The hardware serves the highest priority within the empowered interrupt source.

3.3.1 Interrupt Control Bits in Special Function Registers SFRs

Most of the interrupt control bits, interrupt flags and interrupt enable bits are collected in SFRs under a few addresses. The Special Function Registers are located in the lower address range and are implemented in byte format. SFRs should be only accessed with byte instructions.

Address	7 0
000Fh	Not yet defined or implemented yet
000Eh	:
000Dh	:
000Ch	:
000Bh	:
000Ah	:
0009h	:
0008h	:
0007h	:
0006h	:
0005h	Module enable 2; ME2.x
0004h	Module enable 1; ME1.x
0003h	Interrupt flag reg. 2; IFG2.x
0002h	Interrupt flag reg. 1; IFG1.x
0001h	Interrupt enable 2; IE2.x
0000h	Interrupt enable 1; IE1.x

The various devices of the MSP430 Family support the SFRs with the correct logic and function within the individual modules. Each module interrupt source, except the non-maskable sources, can be individually enabled to access the interrupt function and the operation. Full software control of these configuration bits allows the application software to react to system requirements on interrupt enable mask.

Interrupt Enable 1 and 2

Bit position	Short form	Initial state*	Comment
IE1.0	WDTIE	reset	Watchdog Timer enable signal. Inactive if watchdog mode is selected.
IE1.1	OFIE	reset	Oscillator fault enable
IE1.2	P0IE.0	reset	Dedicated I/O P0.0
IE1.3	P0IE.1	reset	Dedicated I/O P0.1 or 8-bit Timer/Counter
IE1.4		reset	reserved, not defined yet
IE1.5		reset	reserved, not defined yet
IE1.6		reset	reserved, not defined yet
IE1.7		reset	reserved, not defined yet
IE2.0	URXIE	reset	USART receive enable
IE2.1	UTXRIE	reset	USART transmit enable
IE2.2	ADIE / TPIE	reset	ADC or Timer/Port enable signal (‘310 config.)

* Initial state is the logical state after PUC. For the WDTIFG see the appropriate comment.

Bit position	Short form	Initial state	Comment
IE2.3	TPIE	reset	Timer/Port (‘320,’330 config.)
IE2.4		reset	reserved, not defined yet
IE2.5		reset	reserved, not defined yet
IE2.6		reset	reserved, not defined yet
IE2.7	BTIE	reset	Basic Timer enable signal

Interrupt Flag Register 1 and 2

Bit position	Short form	Initial state	Comment
IFG1.0	WDTIFG	unchanged	Set on overflow or security key violation;
		or reset	Reset on VCC power-on or reset condition at ````, RST/NMI-pin
IFG1.1	OFIFG	set	Flag set on oscillator fault
IFG1.2	P0IFG.0	reset	Dedicated I/O P0.0
IFG1.3	P0IFG.1	reset	Dedicated I/O P0.1 or 8-bit Timer/Counter
IFG1.4	NMIIFG	reset	Signal at````, RST/NMI-pin
IFG1.5			reserved, not defined yet
IFG1.6			reserved, not defined yet
IFG1.7			reserved, not defined yet
IFG2.0	URXIFG		USART receive flag
IFG2.1	UTXIFG		USART transmitter ready
IFG2.2	ADIFG	reset	ADC, set on end-of-conversion
IFG2.3			reserved, not defined yet
IFG2.4			reserved, not defined yet
IFG2.5			reserved, not defined yet
IFG2.6			reserved, not defined yet
IFG2.7	BTIFG	unchanged	Basic Timer flag

Module enable 1and 2

Bit position	Short form	Initial state	Comment
ME1.0			reserved, not defined yet
ME1.1			reserved, not defined yet
ME1.2			reserved, not defined yet
ME1.3			reserved, not defined yet
ME1.4			reserved, not defined yet
ME1.5			reserved, not defined yet
ME1.6			reserved, not defined yet
ME1.7			reserved, not defined yet
ME2.0	URXE		USART receiver enable
ME2.1	UTXE		USART transmit enable
ME2.2			reserved, not defined yet
ME2.3			reserved, not defined yet
ME2.4			reserved, not defined yet
ME2.5			reserved, not defined yet
ME2.6			reserved, not defined yet
ME2.7			reserved, not defined yet

Interrupt Vector Addresses

The interrupt vectors and the power-up starting address are located in the ROM, using address range 0FFFFh - 0FFE0h. The vector contains the 16-bit address of the appropriate interrupt handler instruction sequence. The interrupt vectors are shown in decreasing priority order of priority:

Interrupt source	Interrupt flag	System Interrupt	Word Address	Priority
Power-up ext. Reset Watchdog	WDTIFG	Reset	0FFFEh	15, highest
NMI OSC. fault	NMIIFG OFIFG *	non-maskable (non-)maskable	0FFFCh	14
Dedicated I/O	P0IFG.0	maskable	0FFFAh	13
Dedicated I/O	P0IFG.1	maskable	0FFF8h	12
		maskable	0FFF6h	11
Watchdog timer	WDTIFG	maskable	0FFF4h	10
Timer_A	CCIFG0	maskable	0FFF2h	9
Timer_A	TAIFG **	maskable	0FFF0h	8
USART Receive	URXIFG	maskable	0FFEEh	7
USART Transmit	UTXIFG	maskable	0FFECh	6
ADC, Timer/Port [2]	ADCIFG	maskable	0FFEAh	5
Timer/Port [1]		maskable	0FFE8h	4
Port P2	P2IFG.07 *, **	maskable	0FFE6h	3
Port P1	P1IFG.07 *, **	maskable	0FFE4h	2
Basic Timer	BTIFG	maskable	0FFE2h	1
Port P0	P0IFG.27 *, **	maskable	0FFE0h	0, lowest

*) multiple source flags
**) Preliminary definition
[1] Timer/Port vector in ‘320 and ‘330 configuration
[2] Timer/Port vector in ‘310 configuration

Table 3.1: Interrupt sources, flags and vectors

3.3.2 External Interrupts

All eight bits of the entire ports P0, P1 and P2 are implemented for interrupt processing of external events. All individual I/O bits are programmable independently.

Any combinations of inputs, outputs and interrupt conditions are possible. This allows an easy adaptation to different I/O configurations.

> **Note: Minimum pulse width of external interrupt signals**
>
> All external interrupt signals should have a minimum pulse width of 1.5 MCLK to ensure stable interrupt acknowledgement, but shorter signals may also request an interrupt service

Port P0
Three separate vectors are allocated to the port P0 module. The signals on P0.0, P0.1 and the remaining port signals P0.2 to P0.7 are used as the three interrupt vector sources. The vector contained in the corresponding memory location is loaded into the Program Counter by an interrupt even.

The port P0 has 6 registers used for the control of the I/O-pins

- Input Register
- Output Register
- Direction Register
- Interrupt Flags: This register contains six flags, which contain information if the I/O-pins are used as interrupt inputs:
  Bit = 0: No interrupt is pending
  Bit = 1: An interrupt is pending, due to a transition at the I/O-pin.
  Writing a zero to an Interrupt Flag resets it.
  Writing a one to an Interrupt Flag sets it. Device operation continues just the same way as if an interrupt event had occurred.
- Interrupt Edge Select: This register contains a bit for each I/O-pin that selects which transition triggers the interrupt flag.
  Bit = 0: The interrupt flag is set with LO/HI transition
  Bit = 1: The interrupt flag is set with HI/LO transition
- Interrupt Enable: This register contains six bits for the I/O-pins P0.2 to P0.7, to enable interrupt request on an interrupt event.
  Bit = 0: The interrupt request is disabled
  Bit = 1: The interrupt request is enabled

I/O-PIN interrupt handler for P0.2 to P0.7: Programming Example

```
; The I/O-PIN interrupt handler for P0.2 to P0.7starts here
;
IOINTR          PUSH            R5                  ; Save R5
                MOV.B           &P0IFG,R5           ; Read interrupt flags
                BIC.B           R5,&P0IFG           ; Clear status flags with the read
                                                    ; data
                                                    ; Additional set bits are not cleared!
                EINT                                ; Allow interrupt nesting
;
; R5 contains information which I/O-pin(s) caused interrupt:
; the processing starts here.
;
                .........
                .........
                POP             R5                  ; JOB done: restore R5
                RETI                                ; Return from interrupt
                .........
                .........
; Definition of interrupt vector table
                .sect           "IO27_vec",0FFE0h
                .WORD           IOINTR              ; I/O-Pin (2 To 7) Vector In ROM
                ;
                .sect           "RST_vec",0FFFEh    ; Interrupt Vectors
                .WORD           RESET
```

Port P1, Port P2

The ports P1 and P2 are identical. A separate vector is allocated to the port P1 and port P2 module. The pins P1.0 to P0.7 and P2.0 to P2.7 are used as the interrupt sources. The vector contained in the corresponding memory location is loaded into the Program Counter by an interrupt event.

Each port P1 and P2 has 7 registers used for the control of the I/O-pins

- Input Register
- Output Register
- Direction Register
- Interrupt Flags: This register contains eight flags that contain information if the I/O-pins are used as interrupt inputs:
  Bit = 0: No interrupt is pending
  Bit = 1: An interrupt is pending due to a transition at the I/O-pin.
  Writing a zero to an Interrupt Flag resets it.
  Writing a one to an Interrupt Flag sets it. Device operation continues just the same way as if an interrupt event had occurred.

- Interrupt Edge Select: This register contains a bit for each I/O-pin that selects which transition triggers the interrupt flag.
  Bit = 0: The interrupt flag is set with LO/HI transition
  Bit = 1: The interrupt flag is set with HI/LO transition
- Interrupt Enable: This register contains eight bits for the I/O-pins P0.2 to P0.7 to enable interrupt request on an interrupt event.
  Bit = 0: The interrupt request is disabled
  Bit = 1: The interrupt request is enabled
- Function Select Register.

Note: How the interrupts on digital ports P0, P1 and P2 are handled

Only transitions (not static levels) cause interrupts.

The interrupt routine must reset the multiple used Interrupt Flags. Multiple interrupt flags are P0IFG.2 to P0IFG.7, P1IFG.0 to P0IFG.7 and P2IFG.0 to P0IFG.7. The single source flags P0IFG.0 and P0IFG.1 are reset when they are serviced.

If an Interrupt Flag is still set (because the transition occurred during the interrupt routine) when the *RETI* instruction is executed, an interrupt occurs again after the *RETI* is completed. This ensures that each transition is seen by the software.

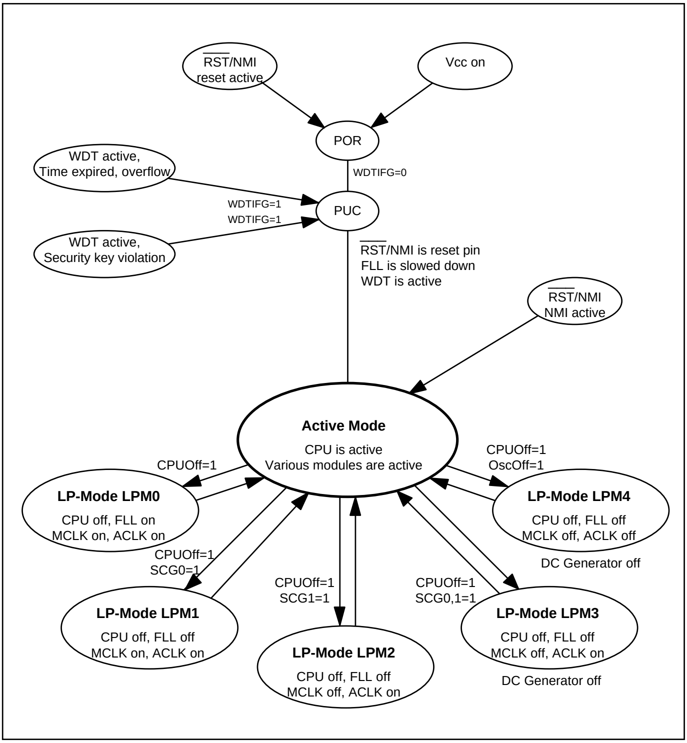
3.4 Operating Modes

The MSP430 operating modes support various requirements for ultra-low power and ultra-low energy consumption in an advanced manner. This is combined with an intelligent management of operations during the different module and CPU states. An interrupt event awakes the system from each of the various operating modes and the RETI instruction returns operation to the mode that was selected before the interrupt event.

The MSP430 Family has been developed for ultra-low power applications and uses different levels of operating modes.

Ultra-low power system design in CMOS technology takes account of three primary intentions:

- the desire for speed and data throughput conflicts with a design for ultra-low power
- minimize individual current consumption
- limit activity state to the minimum required.



There are five operating modes which the software can configure:

- **Active Mode AM,**
with different combinations of active peripheral modules

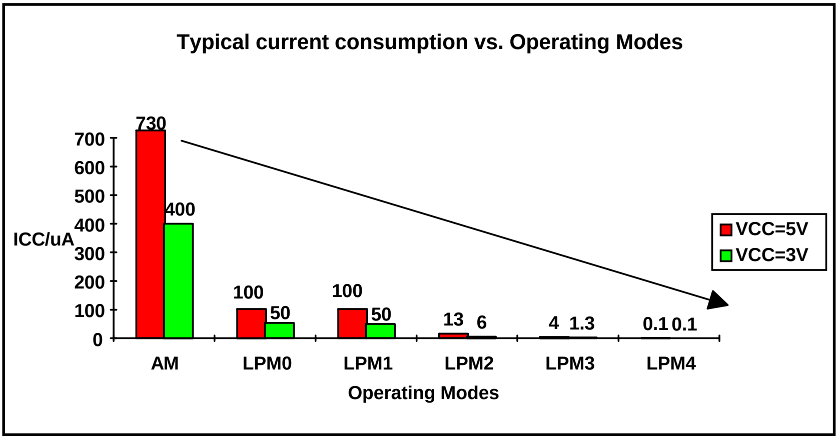
- **Low Power Mode 0 LPM0,**
  with CPUOff bit set, the CPU is disabled,
  peripheral's operation is not halted by CPUOff,
  ACLK and MCLK signal are active. Loop control for MCLK is active.
  @ SCG1=0, SCG0=0, OSCOff=0, CPUOff=1

- **Low Power Mode 1 LPM1,**
  with CPUOff bit set, the CPU is disabled,
  peripheral's operation is not halted by CPUOff,
  loop control (frequency-lock-loop) for MCLK is inactive,
  ACLK and MCLK signal are active.
  @ SCG1=0, SCG0=1, OSCOff=0, CPUOff=1

- **Low Power Mode 2 LPM2,**
  with CPUOff bit set, the CPU is disabled,
  peripheral's operation is not halted by CPUOff,

loop control for MCLK signal is inactive,
ACLK signal is active.
@ SCG1=1, SCG0=0, OSCOff=0, CPUOff=1

- **Low Power Mode 3 LPM3,**
  with CPUOff bit set, the CPU is disabled,
  peripheral's operation is not halted by CPUOff,
  Loop control for MCLK and MCLK signal are inactive,
  DC generator of the DCO (-> MCLK generator) is switched off.
  ACLK signal is active.
  @ SCG1=1, SCG0=1, OSCOff=0, CPUOff=1
- **Low Power Mode 4 LPM4,**
  with CPUOff bit set, the CPU is disabled,
  peripheral's operation is not halted by CPUOff,
  loop control for MCLK signal is inactive,
  DC generator of the DCO (-> MCLK generator) is switched off,
  ACLK signal is inactive; the crystal oscillator is stopped.
  @ SCG1=X, SCG0=X, OSCOff=1, CPUOff=1


Typical current consumption vs. Operating Modes

Source: TI Data sheet SLASE07, January 1996 (MSP430C312/314)

The activity state of individual peripheral modules and the CPU can be controlled using the appropriate low power mode, and various options to stop operation of parts of peripheral modules, or to stop them completely. There are different ways to configurethe lowest potential current consumption, using the software on an application-specific basis. The special function registers include module enable bits that stop or enable the operational function of the specific peripheral module. All registers of the peripherals may be accessed even during disable mode. Other current saving functions can be implemented into peripherals that are accessed via the state of the register bits. An example is the enables/disable of the analog voltage generator in the LCD peripheral: this is turned on or off via one register bit. The most general bits that influence the current consumption and support fast turn-on from low power operating modes are located in the status register SR. There are four bits that control the CPU and the system clock generator.

These four bits are very useful to support the request for discontinuous active mode AM, and to limit the time period of the full operating mode. The four bits are CPUOff, OscOff, SCG0 and SCG1. The major advantage of including the operating mode bits into the status register is that the present state of the operating condition is saved onto stack during an interrupt request service. As long as the stored status register information is not altered, the processor continues (after *RETI*) with the same operating mode as before the interrupt event. Another program flow may be selected by manipulation of the data stored on the stack or the stack pointer. The easy access of the stack and stack pointer with the instruction set allows individually optimized program structures.

3.5 Low Power Modes

The module enable bits in the SFRs enable the configuration of individual power consuming controller operation states. The users program defines the state of the peripheral modules to be active or inactive. The current consumption of disabled modules is decreased by the leakage current of all parts that can be disabled. The only active parts of a module are those which are mandatory to get it to the enable state or to pass interrupt requests to the CPU (e.g. external hardware interrupt).

In addition to the individual enable options, there are five more current saving modes possible: the CPU off mode (LPM0), and four operating configurations of the system clock generator. They are entered if one or more of the bits CPUOff, SCG1, SCG0, OscOff - located in the Status Register - are set. The reaction of the system clock generator module on the status of the bits SCG1, SCG0 and OscOff with its four low power modes are described in detail in the system clock generation section.

Enter interrupt routine
The interrupt routine is entered and processed if an enabled interrupt wakes-up the MSP430:

- The SR and PC are stored onto the stack, with the content present at the interrupt event.
- Subsequently the operation mode control bits OscOff, SCG1 and CPUOff are cleared automatically in the Status Register.

Return from interrupt
Two different ways back from interrupt service routine to continue flow of operation are practicable:

- Return with set low power mode bits
  When returning from the interrupt, the program counter points to the next instruction. The instruction pointed to is not executed, since the restored low power mode stops CPU activity.
- Return with reset low power mode bits
  When returning from the interrupt, the program continues at the address following the instruction which set the OscOff or CPUOff-bit in the Status Register.

3.5.1 Low Power Mode 0 and 1, LPM0 and LPM1

Low power mode 0 or mode 1 is selected if the appropriate bit CPUOff in the status register is set. Immediately after the bit is set the CPU stops operation, and the normal operation of the system core is stopped. The operation of the CPU is halted until any interrupt request or reset is effective. All internal bus activities are stopped. The system clock generator continues operation, and the clock signals MCLK and ACLK are active depending on the state of the other three bits, SCG0, SCG1 and OscOff in the status register. The SCG1 bit defines if the MCLK is controlled to be N*ACLK, or to run with the latest DCO control signals.

Those peripherals are active which are enabled and clocked with the MCLK or ACLK signal. All pins of I/O ports and the RAM/registers are unchanged. Wake-up is possible by all enabled interrupts.

```
; === Main program flow with switch to CPUOff Mode ========================
;
            BIS    #18h,SR     ; Enter LPM0 + enable general interrupt GIE.
                               ; The PC is incremented during execution of this in-
                               ; struction and points to the consecutive program step.
            ..........         ; The program continues here if CPUOff bit is reset
                               ; during the interrupt service routine

; === Interrupt service routine ==========================================
            ..........
            ..........
            RETI               ; RETI restores the same state of CPU before
            interrupt.
                               ; This is possible because control registers GIE,
                               ; CPUOff, OscOff, SGC1 and SCG0 are located in the
                               ; status register which is restored during execution of
                               ; return-from-interrupt.
```

3.5.2 Low Power Mode 2 and 3, LPM2 and LPM3

Low power mode 2 or mode 3 is selected if the appropriate bit CPUOff and SCG1 bit in the status register are set. Immediately after the bits are set, the CPU and MCLK are halted. The CPU and MCLK are halted until any interrupt request or reset is effective. All internal bus activities are stopped. The SCG1 bit defines if the MCLK is controlled to be N*ACLK or to run with the latest DCO control signals when the sytem returns to active mode.

Those peripherals are active that are enabled and clocked with the ACLK signal. Peripherals that are operating with the MCLK signal are inactive, because the MCLK signal is inactive. All pins of I/O ports and the RAM/registers are unchanged. Wake-up is possible by those enabled interrupts coming from system clock (MCLK) independent sources.

3.5.3 Low Power Mode 4, LPM4

All activities cease; only the RAM contents, Port and registers are maintained. Wake-up is only possible by enabled external interrupts.

Before activating LPM4, the software flow should consider the conditions that are applied to the system during the period of this low power mode. The two and most important figures that should be looked at are the environmental situation, with the influence at the DCO and the clocked operation conditions. The environmental situation defines whether the actual value of the frequency integrator should be held or corrected. A correction can be intended when ambient conditions would increase the system frequency drastically. When clocked operation is applied, it should be considered that the loop can lose control over the frequency if there remaining time slot is insufficient to hold the closed loop in the correct operating range.

The following example shows the entering of the low power mode 4 (OscOff):

```
BIS   #B8h,SR    ; Enter LPM4 + enable general interrupt GIE.
                 ; The CPU must be switched of with LPMs.
                 ; Additionally the DCO operation is enabled.
                 ; When during the interrupt routine the LPM4 is going
                 ; to be disrupted, DCO operation is prepared.
..........       ; The program continues here if OscOff bit is reset
..........       ; during the interrupt service routine.
..........       ; Otherwise it retains in OscOff mode
```

3.6 Basic Hints for Low Power Applications

There are some general basics principles which should be considered when the current consumption is a critical part of a system application:

- Tie unused FETI input to V_{SS}
- Switch-off the Analog Generator in the LCD+ module or an external one if convenient
- Do not tie the JTAG inputs TMS, TCK and TDI to V_{SS}
- Any CMOS input should have no floating node: tie all inputs to an appropriate voltage level
- Select the lowest possible operating frequency - for the core and for the individual peripheral module
- Select the weakest drive capability if an LCD is used, or switch it off
- Utilize the feature of interrupt driven software - the program starts execution rapidly.

4 Memory Organization

Topic **Page**

The MSP430 family's memory space is configured in a "von-Neumann Architecture" and has code memory (ROM, EPROM, RAM) and data memory (RAM, EEPROM, ROM) in one address space using a unique address and data bus.

All the physically separated memory areas, the internal areas for ROM, RAM, SFRs and peripheral modules, and the external memory, are mapped into the common address space. The total addressable memory space provided is 64KB in the small memory model and 1MB in the large memory model. The small memory model uses a linear address space, while in the large memory model the address space is arranged in sixteen segments of 64KB at code access, and 16 pages of 64KB at data access.

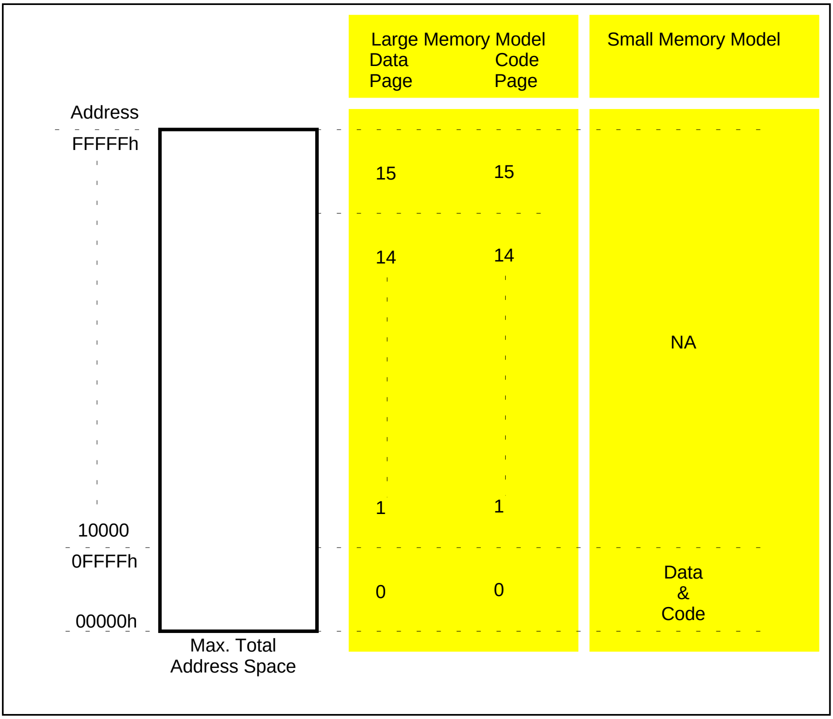


Figure 4.1: Total Memory Address Space

Devices with a memory configuration of 64KB or less use the small memory model with basic address range of the lowest 64KB, and do not care about code segments and data pages.

The configuration according to the small memory model and data bus width is shown below:

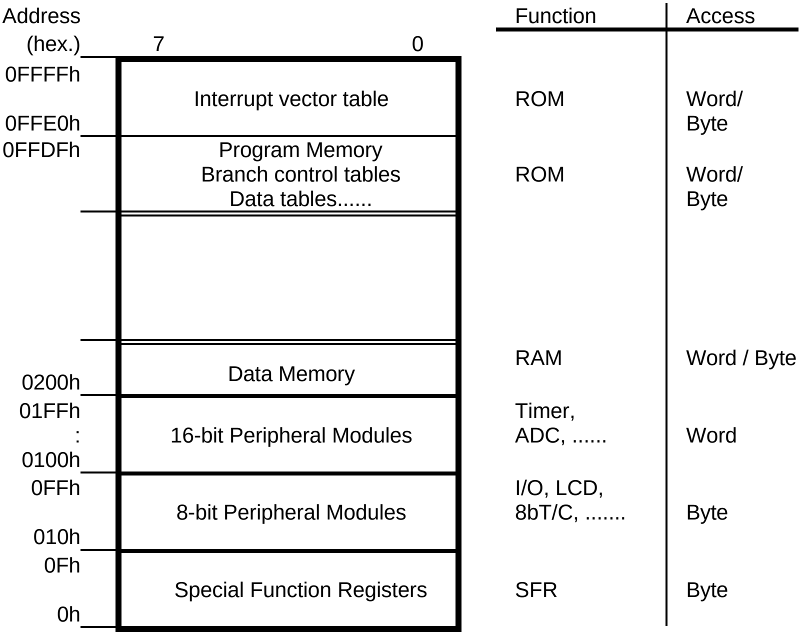
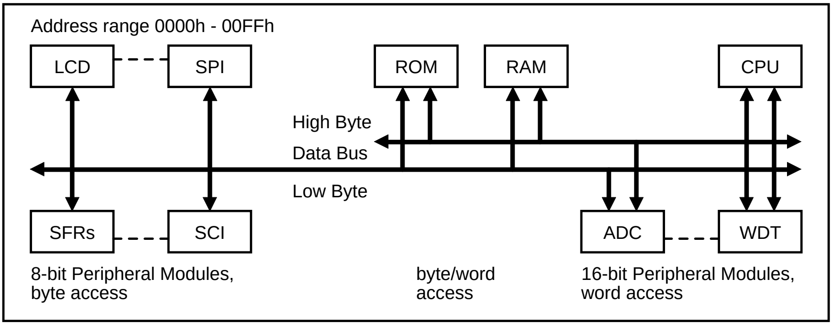


Figure 4.2: Memory Map of Basic Address Space

The Data Bus is 16-bit or 8-bit wide. For those modules that can be accessed with word data, the width is always 16 bits, and for the other modules 8 bits; they should only be accessed with byte instructions. The Program Memory (ROM) and the Data Memory (RAM) can be accessed with byte or word instructions. Parts of peripheral modules are realized as 16-bit wide or 8-bit wide modules. The access should use the proper instructions, either byte or word.

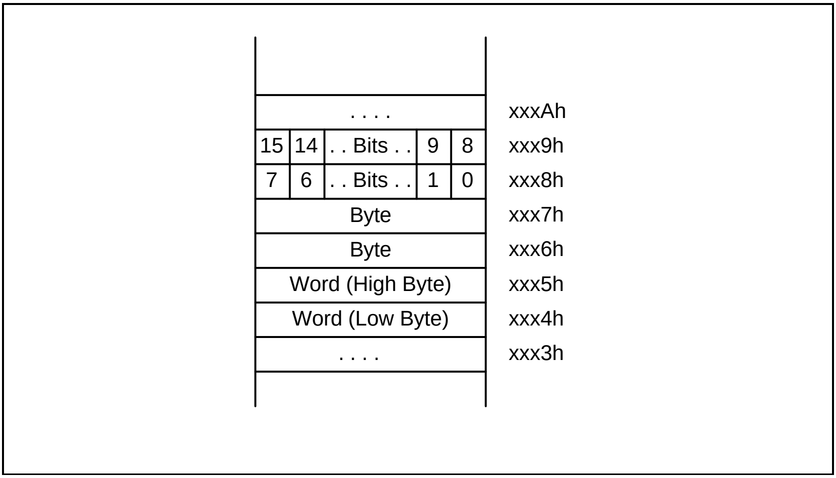
Many peripheral modules are connected to the CPU with an 8-bit Memory Data Bus (MDB), with the 5 least significant bits of the Memory Address Bus (MAB) plus two Module Enable signals (ME), two interrupt control/request lines, and a power-up signal.

The access to these modules should be always performed using byte instruction formats. Other 16-bit peripheral modules are connected to the 16-bit MDB with full supporting word processing, and should use word instruction format for any access.


Address range 0000h - 00FFh
LCD
SPI
ROM
RAM
CPU
High Byte
Data Bus
Low Byte
SFRs
SCI
ADC
WDT
8-bit Peripheral Modules, byte access
byte/word access
16-bit Peripheral Modules, word access

4.1 Data in the Memory

Bytes are located at even or odd addresses. Words are located in the ascending memory locations aligned to even addresses: the low byte is at the even address, followed by the high byte at the next odd address.



. . . .					xxxAh
15	14	. . Bits . .	9	8	xxx9h
7	6	. . Bits . .	1	0	xxx8h
Byte					xxx7h
Byte					xxx6h
Word (High Byte)					xxx5h
Word (Low Byte)					xxx4h
. . . .					xxx3h

Figure 4.3: Bit, Byte and Word in a byte organized Memory

4.2 Internal ROM Organization

Various sizes of ROM up to 64K bytes are possible. The common address space is shared with special function registers, peripheral module registers, data and code memory. The special function registers and peripheral modules are mapped into the address range, starting with 0 and up to 01FFh. The remaining address space 0200h to 0FFFFh is shared by data and code memory.

The start address for all different sizes of ROM is at the same address 0FFFEh. The interrupt vector table also starts with highest priority at this highest ROM word address. The program counter, and hence the flow of instructions, is in the opposite direction - from lower addresses towards higher addresses. The program counter is increased by two, four or six according to the address mode used - program flow control instruction Jumps, branches and calls excluded.

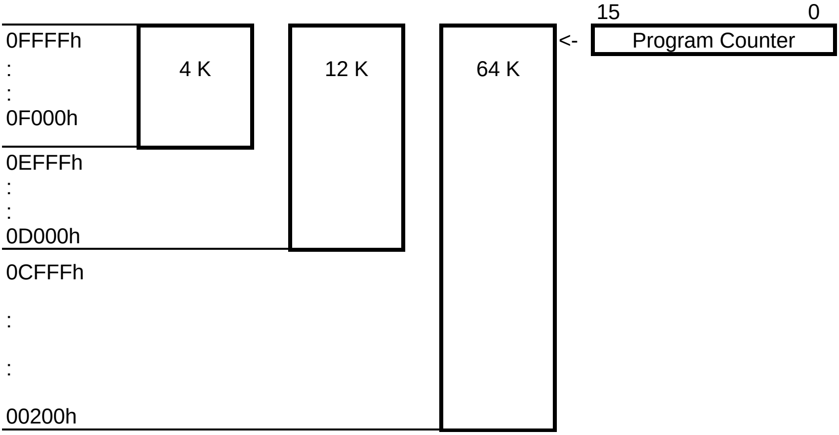


Figure 4.4: ROM Organization

The interrupt vectors and the power-up vector are located in the ROM, starting at address 0FFFEh. The vectors contain the 16-bit addresses of the appropriate interrupt handler instruction sequence.

4.2.1 Processing of ROM Tables

The MSP430 architecture allows the storage of large tables in the ROM. To access these tables, all word and byte instructions can be used. This offers various advantages with regard to flexible and ROM saving programming:

- Storage of an Output-PLA for display character conversion inside the ROM
- As many OPLA-terms as needed (no restriction on n terms)

- OTP version automatically includes OPLA programmability
- Computed table accesses (e.g. for a bar graph display)
- Table supported program flows.

The processing of tables is a very important feature, which allows very fast and clear programming. Especially for sensor applications, it is advantageous to have the sensor data in tables e.g. for linearization, compensation etc.

4.2.2 Computed Branches and Calls

Computed branches and subroutine calls are possible using standard instructions. The CALL and BR instructions use the same addressing modes as the other instructions (see programming examples).

The addressing modes allow indirect-indirect addressing, ideally suited for computed branches and calls. The full use of this programming technique permits a program structure different to conventional 8- and 16-bit controllers. A lot of routines can be handled easily using software status handling, instead of 'Flag' type program flow control.

The computed branches and subroutine calls are valid within a 64KB code segment.

4.3 RAM and Peripheral Organization

The entire RAM can be accessed in byte or word data, using the appropriate instruction suffix. The peripheral modules are located in two different address spaces:

- the special function registers are byte oriented by hardware and mapped into the address space from 0h up to 0Fh
- the peripheral modules that are byte oriented by hardware are mapped into the address space from 010h up to 0FFh
- and peripheral modules that are word oriented by hardware are mapped into the address space from 100h up to 01FFh

4.3.1 RAM

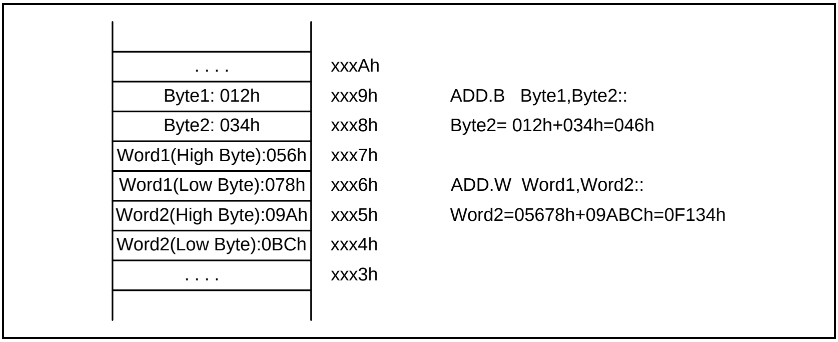
The RAM can be used for both code and data memory. Code accesses are always made on even byte addresses.

The suffix at the instruction memonic defines the access of the data as being word or byte data.

Example:

```
        ADD.B     &TCDATA,TCSUM_L                          ;Byte acess
        ADDC.B    TCSUM_H                                   Byte acess
        ADD       R5,SUM_A     ≡     ADD.W     R5,SUM_A;   ;Word acess
        ADDC      SUM_B        ≡     ADDC.W    SUM_A       ;Word acess
```

A Word consists of two bytes, a Highbyte (bit 15 to bit 8) and a Lowbyte (bit 7 to bit 0) and should always be aligned to even addresses.



Memory	Address	
. . . .	xxxAh	
Byte1: 012h	xxx9h	ADD.B Byte1,Byte2::
Byte2: 034h	xxx8h	Byte2= 012h+034h=046h
Word1(High Byte):056h	xxx7h	
Word1(Low Byte):078h	xxx6h	ADD.W Word1,Word2::
Word2(High Byte):09Ah	xxx5h	Word2=05678h+09ABCh=0F134h
Word2(Low Byte):0BCh	xxx4h	
. . . .	xxx3h	

Figure 4.5: Byte and Word Operation

All operations on Stack and PC are word operations, and use even aligned memory addresses.

Word-to-word and byte-to-byte operations are performed completely correctly, both the results of the operation and the status bit information.

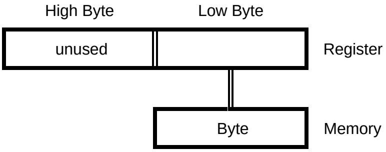
Word-word operation:

```
R5 = 0F28Eh
EDE  .EQU   0212h
Mem(0F28Eh) = 0FFFEh
Mem(0212h) = 00112h

ADD          @R5,&EDE

Mem(0212h) = 00110h
C = 1, Z = 0, N = 0
```

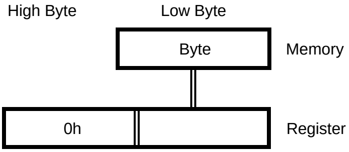
Byte-byte operation

```
R5 = 0223h
EDE  .EQU   0202h
Mem(0223h) = 05Fh
Mem(0202h) = 043h

ADD.B              @R5,&EDE

Mem(0202h) = 0A2h
C = 0, Z = 0, N = 1
```

Register-Byte operation:



Byte-Register operation:



Example Register-Byte operation	Example Byte-Register operation
R5 = 0A28Fh	R5 = 01202h
R6 = 0203h	R6 = 0223h
Mem(0203h) = 012h	Mem(0223h) = 05Fh
ADD.B R5,0(R6)	**ADD.B @R6,R5**
08Fh	05Fh
+ 012h	+ 002h ;Lowbyte of R5
0A1h	061h ;-> store into R5 -
Highbyte is 0	
Mem(0203h) = 0A1h	R5 = 061h
C = 0, Z = 0, N = 1	C = 0, Z = 0, N = 0
(Lowbyte of register)	(addressed byte)
+ (addressed byte)	+ (Lowbyte of register)
->(addressed byte)	->(Lowbyte of register, zero to Highbyte)

Note: Word-Byte operation

Word-Byte or Byte-Word operations on memory data are not supported.
Each register-byte and byte-register operation is performed as a byte operation.

4.3.2 Peripheral Modules - Address Allocation

All peripheral modules are accessed and controlled by the software. All instructions are approved for the data interchange operation. Since there are modules physically using the MDB with its word construction, and modules that use only the eight least significant bits, the address space from 0100 to 01FFh is reserved for word modules and the address space from 00h to 0FFh is reserved for byte modules.

Peripheral modules mapped into the word address space should be accessed with word instructions (e.g. MOV R5,&WDTCTL). Peripheral modules mapped into the word address space should be accessed with byte instructions (MOV.B #1,&TCCTL).

The addressing of both is made via the absolute addressing mode, or via the 16-bit working registers, using the indexed, indirect or indirect autoincrement addressing mode.

Address (hex.)	7 0	Function	Access
01FFh : 0100h	16-bit Peripheral Modules	Timer, ADC, ...	Word
0FFh 010h	8-bit Peripheral Modules	I/O, LCD, 8b T/C, ...	Byte
0Fh 0h	Special Function Registers	SFR	Byte

Figure 4.6: Example of RAM/peripheral organization

Word modules

Word modules are peripherals that are connected to the complete 16-bit MDB.

Access to word modules is always in word format, and byte access is not supported since the hardware is constructed for word operation only.

The peripheral file address space is organized in sixteen frames, and each frame represents eight words.

Address	Description
1F0h - 1FFh	reserved
1E0h - 1EFh	reserved
1D0h - 1DFh	reserved
1C0h - 1CFh	reserved
1B0h - 1BFh	reserved
1A0h - 1aFh	reserved
190h - 19Fh	reserved
180h - 18Fh	reserved
170h - 17Fh	**Timer_A**
160h - 16Fh	**Timer_A**
150h - 15Fh	reserved
140h - 14Fh	reserved
130h - 13Fh	**Multiplier**
120h - 12Fh	**Watchdog Timer**
110h - 11Fh	**Analog-to-Digital Converter**
100h - 10Fh	reserved

Figure 4.7: Peripheral File Address Map - Word Modules

Byte modules

Byte modules are peripherals that are connected to the reduced (eight LSB) MDB. The access to byte modules is always a byte access. The hardware in the peripheral byte modules takes the LowByte - the least significant bits - along with a write operation.

Byte instructions operate on byte modules without any restriction. Read access to the data of a peripheral byte module with word instructions results in unpredictable data on the Highbyte. Word data are written into a byte module by writing the LowByte to the appropriate peripheral register, and ignoring the HighByte.

The peripheral file address space is organized in sixteen frames.

Address	**Description**
00F0h - 00FFh	reserved
00E0h - 00EFh	reserved
00D0h - 00DFh	reserved
00C0h - 00CFh	reserved
00B0h - 00BFh	reserved
00A0h - 00AFh	reserved
0090h - 009Fh	reserved
0080h - 008Fh	reserved
0070h - 007Fh	**USART registers**
0060h - 006Fh	reserved
0050h - 005Fh	**System Clock Generator registers**
0040h - 004Fh	**Basic Timer, 8-bit Timer/Counter, Timer/Port registers**
0030h - 003Fh	**LCD registers**
0020h - 002Fh	**Digital I/O Port P3 and P4 control registers**
0010h - 001Fh	**Digital I/O Port P0, P1 and P2 control registers**
0000h - 000Fh	**Special Function Registers**

Figure 4.8: Peripheral File Address Map - Byte Modules

4.3.3 Peripheral Modules - Special Function Registers SFRs

The system configuration and the individual reaction of the peripheral modules to processor operation modes are mainly defined in Special Function Registers. The Special Function Registers are located in the lower address range, and are realized in **byte** manner. SFRs should be only accessed with byte instructions. Even if specific SFR bits share the same address space, they can be implemented physically within the associated module.

Address	Data Bus 7 … 0
000Fh	Not defined / implemented yet
000Eh	:
000Dh	:
000Ch	:
000Bh	:
000Ah	:
0009h	:
0008h	:
0007h	:
0006h	:
0005h	Module enable 2; ME2.2
0004h	Module enable 1; ME1.1
0003h	Interrupt flag reg. 2; IFG2.x
0002h	Interrupt flag reg. 1; IFG1.x
0001h	Interrupt enable 2; IE2.x
0000h	Interrupt enable 1; IE1.x

Figure 4.9: Special Function Register Address Map

The different devices of the MSP430 Family support SFRs with the correct logic and function within the individual modules. Each module can be enabled individually, to access the interrupt function and the operation. Full software control of these configuration bits enables the application software to react to system requirements on interrupt enable mask.

The power consumption of the system is influenced by the number of the enabled modules, and their function. Disabling a module from the actual operation mode reduces power consumption while other parts of the controller remain fully active. Two parts can not be disabled: ROM and RAM. The processor core can be switched to disabled mode - CPUOff Mode - with all internal functions disabled: CPU and bus activities are stopped.

5 CPU, 16bit

Topic **Page**

The equal width of the PC register, and also of the working registers, allows new features: for example, seven addressing modes.
The "von-Neumann-Architecture" used in the MSP430 has RAM and ROM in one adress space, using a single address and data bus.

5.1 CPU Registers

Fourteen 16-bit registers (R0, R1, R4 to R15) are used for data and addresses. These registers are implemented in the CPU. They are able to address up to 64KBytes (ROM, RAM, EERAM, Peripherals,...) without any segmentation. The complete CPU register set is shown below. The registers which are used for special purposes are marked. The registers R0, R1, R2 and R3 are restricted in their common use due to their special functions, described later.

Program Counter PC	R0
Stack Pointer SP	R1
Status Register SR	R2
Constant Generator CG1	
Constant Generator CG2	R3
Working Register R4	R4
Working Register R5	R5
: :	: :
Working Register R13	R13
Working Register R14	R14
Working Register R15	R15

Table 5.1: Register by functions

5.1.1 The Program Counter PC

The 16-bit Program Counter PC defines which instruction will be executed next. Each instruction uses an even number of bytes: two, four or six bytes. The instruction accesses are performed on word boundaries, and so the program counter is aligned to even addresses. The PC is double-incremented during the fetch cycle of an instruction: it points to the word following the currently executed instruction. This makes two additional addressing modes possible (Immediate Mode and Symbolic Mode), which use the word following the instruction for information.

15	1	0
Program counter bits 15 to 1		0

Figure 5.1: Program Counter PC

5.1.2 The System Stack Pointer SP

The system Stack Pointer SP should always be aligned to even addresses, since the stack is accessed with word data during interrupt request service. The system Stack Pointer SP is used by the CPU for the storage of the return addresses of subroutine calls and interrupts. It uses a pre-decrement, post-increment scheme. This scheme has the advantage that the item on the top of the stack (TOS) is available. The SP may be used by the user's software (PUSH and POP instructions), but it should be remembered that the CPU uses the Stack Pointer too.

15	1	0
System Stack Pointer bits 15 to 1		0

Figure 5.2: System Stack Pointer SP

Note: Software stack pointer using general purpose registers

The general purpose registers R4 to R15 can be used as SW-stackpointers.

Pushing item onto a **word SW-stack** controlled by Rn:

```
DECD      Rn              ; Double-decrement SW-SP Rn
MOV       item,0(Rn)      ; PUSH item on SW-stack
```

Popping item off a SW-stack is made by:

```
MOV       @Rn+,item       ; POP ITEM off SW-stack
```

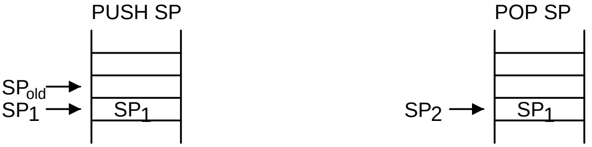
Pushing item onto a **byte SW-stack** controlled by Rm:

```
DEC       Rm              ; Decrement SW-SP Rm
MOV.B     item,0(Rn)      ; PUSH item on SW-stack
```

Popping item off a byte SW-stack is made by:

```
MOV.B     @Rn+,item       ; POP ITEM off SW-stack
```

Special condition on **PUSH** and **POP** of the System Stack Pointer.



The Stack Pointer is not changed after PUSH SP instruction

The Stack Pointer SP is loaded with the data of the memory pointed to by SP before executing POP SP instruction

After the sequence

```
PUSH SP             ; SP1 is stack pointer after 1. inst.
     |
     |
POP  SP             ; SP2 is stack pointer after 2. inst.
```

the Stack Pointer is two bytes lower than before this sequence.

Examples for System Stack Pointer addressing (refer to figure Stack Usage):

```
MOV    SP,R4      ; #0xxxh - 4  -> R4
MOV    @SP,R5     ; Item I3 (TOS) -> R5
MOV    2(SP),R6   ; Item I2 -> R6
MOV    R7,0(SP)   ; overwrite TOS with R7
MOV    R8,4(SP)   ; modify item I1
PUSH   R12        ; store R12 in address 0xxxh - 6; SP points to same address
POP    R12        ; restore R12 from address 0xxxh - 6; SP points to 0xxxh - 4
MOV    @SP+,R5    ; item I3 -> R5 (popped from Stack); same as POP instruction
PUSH   #1
POP    R8
```

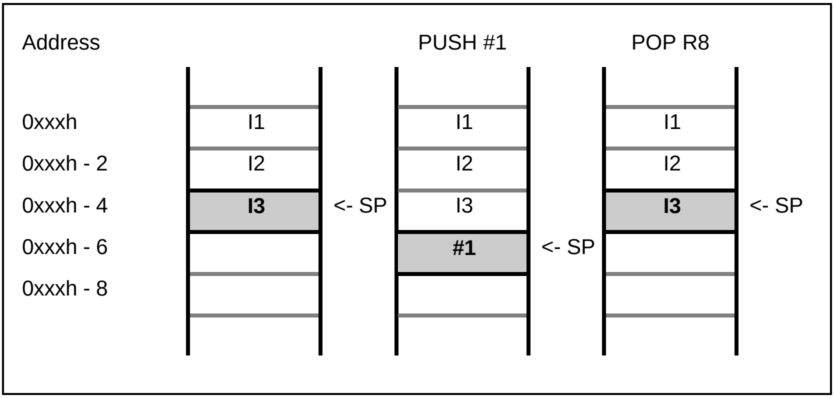


Figure 5.3: Stack Usage

5.1.3 The Status Register SR

The Status Register SR contains the CPU status bits:

• **V**	Overflow Bit
• **SCG1**	System Clock Generator Control Bit 1
• **SCG0**	System Clock Generator Control Bit 0
• **OscOff**	Crystal Oscillator Off Bit
• **CPUOff**	CPU Off Bit
• **GIE**	General Interrupt Enable Bit
• **N**	Negative Bit
• **Z**	Zero Bit
• **C**	Carry Bit

15 ... 9	8	7							0
reserved for future enhancements	V	SCG1	SCG0	OscOff	CPUOff	GIE	N	Z	C
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Figure 5.4: Status Register SR

Description of status bits

- **Overflow Bit (V):** Set if the result of an arithmetic operation overflows the signed variable range. It is valid for both data formats, byte and word:

  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, SCG0:** These bits control four activity states of the system clock generator and therefore influence the operation of the processor system.
- **Oscillator Off:** If set, the Crystal Oscillator enters the Off Mode: all activities cease, but the RAM contents, Port and registers are maintained. Wake-up is possible only by enabled external interrupts when GIE is set and from NMI. This bit should **not** be set without simultaneously setting CPUOff bit.
- **CPU Off:** If set, the CPU enters the Off Mode: all activities ceases, but the RAM, Port and registers and specially enabled peripherals e.g. Basic Timer, UART ... stay active. Wake-up is possible by all enabled interrupts.
- **GIE Bit (GIE):** If set, all enabled interrupts are handled. If reset, all interrupts are disabled. The GIE Bit is cleared by interrupts and restored by the RETI instruction. It can be also changed by appropriate instructions.
- **Negative Bit (N):** Set if the result of an operation is negative.
  Word operations: Negative bit is set to the value of bit 15 of the result.
  Byte operations: Negative bit at is set to the value of bit 7 of the result.
- **Zero Bit (Z):** Set if the result of an operation is 0, cleared if the result is not 0.
- **Carry Bit(C):** Set if the result of an operation produced a carry, cleared if no carry occurred.
  Word operation: The carry is as the result of the word operation.
  Byte operation: The carry is as the result of the byte operation.
  Some instructions have the carry bit modified with the inverted zero bits.

Note: Status bits V, N, Z and C

The status bits V, N, Z and C are modified only with the appropriate instruction. Please see the detailed description of the instruction set, MSP430 Software User's Guide.

5.1.4 The Constant Generator Registers CG1 and CG2

The most often used constants can be generated with the constant registers R2 and R3, without occupying an additional 16-bit-word. The used constant for immediate values is defined by the addressing bits As:

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

Table 5.2: Values of constant generators CG1, CG2

The major advantages are allied with the use of this type of constant generation:

- No special instructions
- No additional word for the seven most used constants
- Shorter instruction cycles time: direct access without use of MDB

The assembler uses the R2 or R3 modes automatically, if one of the six constants is used in immediate mode as a source operand. The Status Register SR/R2 - used as source or destination register - can be used in register mode only. The remaining combinations of address bits As are used to support absolute address mode and bit processing without adding additional code. Registers R2 and R3 used in the 'constant mode' cannot be addressed explicitly; they act just like a source only register.

The Constant Generator Registers allow the emulation of several instructions by other ones. The CPU is much simpler this way. Only 27 instructions are needed for the complete instruction set. For example the Single Operand Instruction:

```
CLR     dst
```

is emulated by the Double Operand Instruction with the same length:

```
MOV     R3,dst
or equivalent
MOV     #0,dst
```

where #0 is replaced by the assembler, with R3 used with As = 00:

- one word instruction
- no additional control operation or hardware within CPU
- register addressing mode for source: no extra fetch cycle for constants (#0).

5.2 Addressing modes

All seven addressing modes for the source operand and all four addressing modes for the destination operand can address the complete address space. The bit numbers show the contents of the As and Ad 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.
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
11/-	Immediate Mode	#N	The word following the instruction contains the immediate constant N. Indirect Autoincrement Mode @PC+ is used

Note: Addressing modes

The addressing modes using the PC as the working register use the normal effects of the addressing modes. The special addressing modes are caused by the pointing of the PC to the ROM word following the currently executed instruction.

The seven addressing modes are explained in detail by examples. Most of the examples show the same addressing modes for source and destination, but any valid combination of source and destination addressing modes is possible with an instruction.

5.2.1 Register mode

Assembler Code	Content of ROM
MOV R10,R11	MOV R10,R11

Length: 1 or 2 word

Operation: Move the content of R10 to R11. R10 is not affected.

Comment: Valid for source and destination

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**
>
> Since the data in the registers are word data, any operation register-register should be a word operation, and word instructions should be used.

5.2.2 Indexed mode

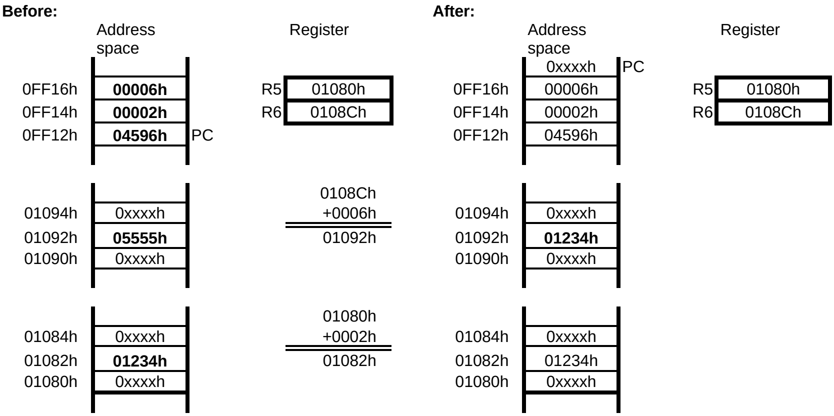
Assembler Code	Content of ROM
MOV 2(R5),6(R6)	MOV X(R5),Y(R6)
	X = 2
	Y = 6

Length: 2 or 3 words

Operation: Move the contents of the source address (contents of R5 + 2) to the destination address (contents of R6 + 6). The source and destination registers (R5 and R6) are not affected. With Indexed mode, the PC is incremented automatically so that program execution continues with the next instruction.

Comment: Valid for source and destination

Example: MOV 2(R5),6(R6):



5.2.3 Symbolic mode

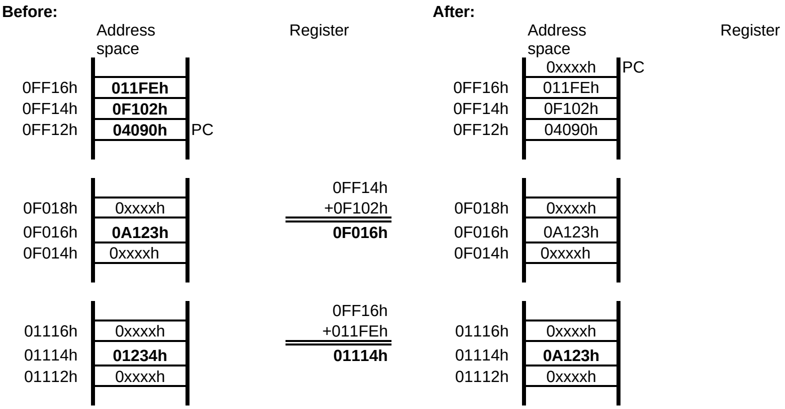
Assembler Code	Content of ROM
MOV EDE,TONI	MOV X(PC),Y(PC)
	X = EDE - PC
	Y = TONI - PC

Length: 2 or 3 words

Operation: Move the contents of the source address EDE (contents of PC + X) to the destination address TONI (contents of PC + Y). The words after the instruction contain the differences of the PC and the source or destination addresses. The assembler computes and inserts the offsets X and Y automatically. With Symbolic mode, the PC is incremented automatically so that program execution continues with the next instruction.

Comment: Valid for source and destination

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



5.2.4 Absolute mode

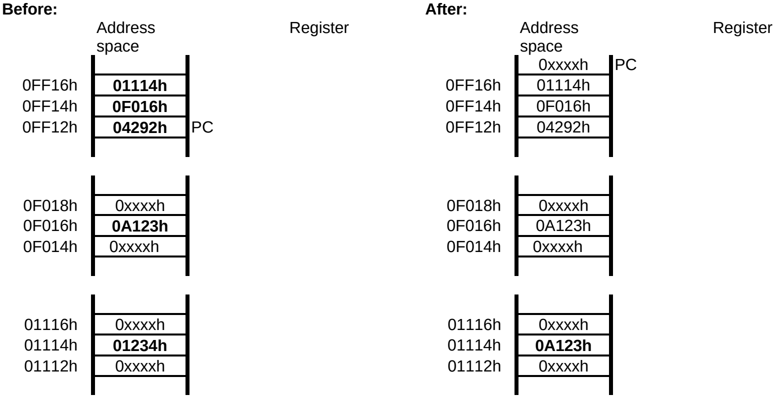
Assembler Code	Content of ROM
MOV &EDE,&TONI	MOV X(0),Y(0)
	X = EDE
	Y = TONI

Length: 2 or 3 words

Operation: Move the contents of the source address EDE to the destination address TONI . The words after the instruction contain the absolute address of the source and of the. destination addresses. With absolute mode, the PC is incremented automatically, so that program execution continues with the next instruction.

Comment: Valid for source and destination

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



The main use of this address mode is for hardware peripheral modules that are located at an absolute, fixed address. These should be addressed with absolute mode to ensure software transportability e.g. position independent code (PIC) programming techniques. Absolute mode always uses code segment 0.

5.2.5 Indirect mode

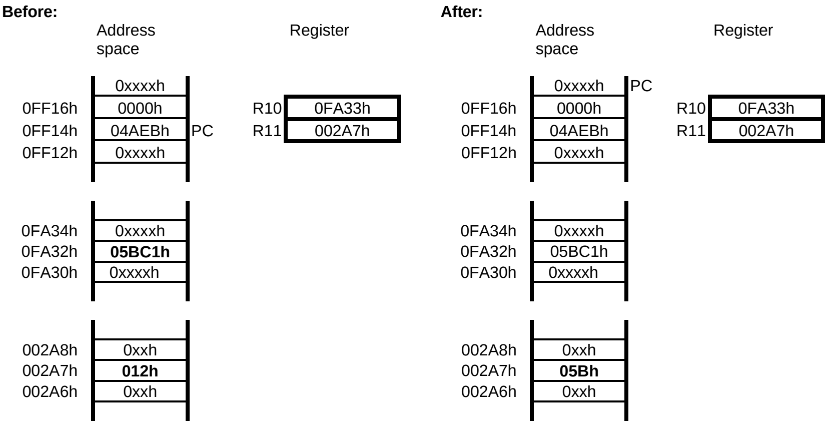
Assembler Code	Content of ROM
MOV @R10,0(R11)	MOV @R10,0(R11)

Length: 1 or 2 word(s)

Operation: Move the contents of the source address (contents of R10) to the destination address (contents of R11). The registers are not modified.

Comment: Valid only for source operand. Substitute for destination operand is 0(Rd).

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



5.2.6 Indirect autoincrement mode

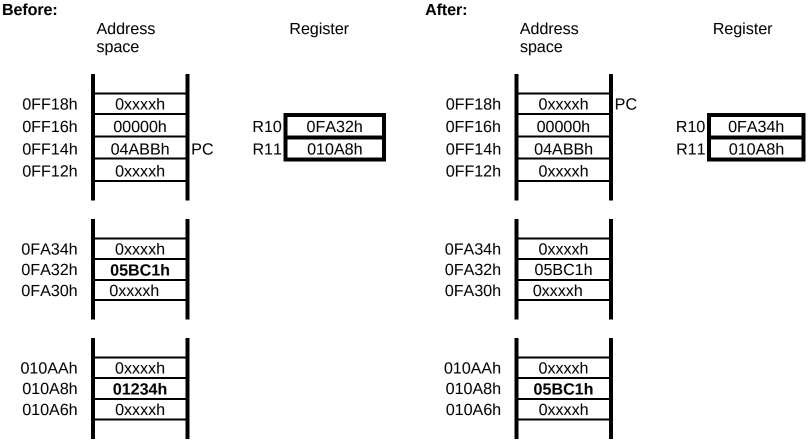
Assembler Code	Content of ROM
MOV @R10+,0(R11)	MOV @R10+,0(R11)

Length: 1 or 2 word(s)

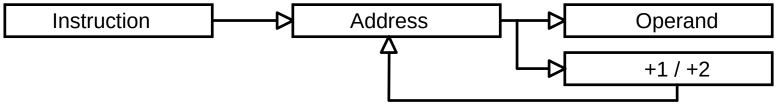
Operation: Move the contents of the source address (contents of R10) to the destination address (contents of R11). The register R10 is incremented by one (byte operation) or two (word operation) after the fetch: it points to the next address now without any overhead. This is very useful for table processing.

Comment: Valid only for source operand. Substitute for destination operand is 0(Rd) plus second instruction INCD Rd.

Example: MOV @R10+,0(R11)



The autoincrement of the registers' content is done after the operand is fetched for performing the operation.



5.2.7 Immediate mode

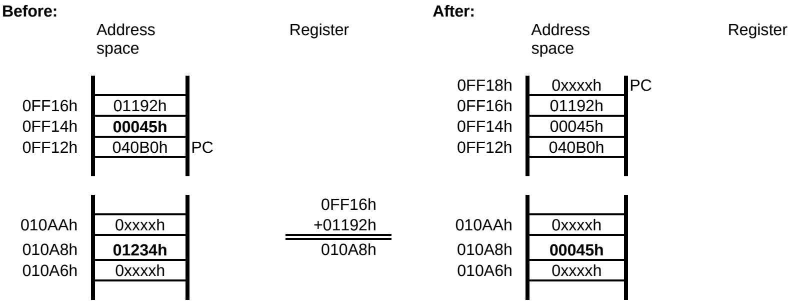
Assembler Code	Content of ROM
MOV #45,TONI	MOV @PC+,X(PC)
	45
	X = TONI - PC

Length: 2 or 3 words
It is 1 word less if a constant of CG1 or CG2 can be used.

Operation: Move the immediate constant 45 which is contained in the word following the instruction to the destination address TONI. When fetching the source, the PC points to the word after the instruction and moves the contents to the destination.

Comment: Valid only for source operand.

Example: MOV #45,TONI



5.2.8 Clock cycles, Length of Instruction

The operating speed of the CPU is independent of individual instructions. It depends on the instruction format and the addressing modes. The number of clock cycles refer to the internal oscillator frequency.

Format I Instructions

Address Mode		# of cycles	Length of instruction	Example
As	Ad			
00, Rn	0, Rm	1	1	MOV R5,R8
	0,PC	2	1	BR R9
00, Rn	1, x(Rm)	4	2	ADD R5,3(R6)
	1, EDE		2	XOR R8,EDE
	1, &EDE		2	MOV R5,&EDE
01, x(Rn)	0, Rm	3	2	MOV 2(R5),R7
01, EDE			2	AND EDE,R6
01, &EDE				MOV &EDE,R8
01, x(Rn)	1, x(Rm)	6	3	ADD 3(R4),6(R9)
01, EDE	1, TONI		3	CMP EDE,TONI
01, &EDE	1, &TONI		3	MOV 2(R5),&TONI
				ADD EDE,&TONI
10, @Rn	0, Rm	2	1	AND @R4,R5
10, @Rn	1, x(Rm)	5	2	XOR @R5,8(R6)
	1, EDE		2	MOV @R5,EDE
	1, &EDE		2	XOR @R5,&EDE
11, @Rn+	0, Rm	2	1	ADD @R5+,R6
	0, PC	3	1	BR @R9+
11, #N	0, Rm	2	2	MOV #20,R9
	0, PC	3	2	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

Format II Instructions

Address Mode $A_{(s/d)}$	# of cycles		Length of instruction [words]	Example
	RRA RRC SWPB SXT	PUSH/ CALL		
00, Rn	1	3/4	1	SWPB R5
01, x(Rn) 01, EDE 01,&EDE	4 4	5 5	2 2	CALL 2(R7) PUSH EDE SXT &EDE
10, @Rn	3	4	1	RRC @R9
11, @Rn+ see Note 11, #N	3	4/5	1 2	SWPB @R10+ CALL #81h

Note: Instruction Format II immediate mode

Instructions RRA, RRC, SWPB and SXT should not be used with the immediate mode in the destination field. This would result in unpredictable program operation.

Format III Instructions

Jxx - instructions need all the same #-of-cycles independent of a successful Jump or not.

Clock Cycle:	2 Cycle
Length of Instruction:	1 word

Miscellanous Instructions or Operations

RETI	Clock Cycle:	5 Cycle
	Length of instruction:	1 word
Interrupt	Clock Cycle:	6 Cycle
WDTreset	Clock Cycle:	4 Cycle
Reset (RST/NMI)	Clock Cycle:	4 Cycle

5.3 Instruction set overview

The following gives a short overview of the instruction set.

The effects of an instruction on the Status Register Bits are shown below:

*	The Status Bit is affected
-	The Status Bit is not affected
0	The Status Bit is cleared
1	The Status Bit is set

The source and destination parts of an instruction are defined by two fields each (the addressing modes are described above):

src The source operand defined by As and S-reg
dst The destination operand defined by Ad and D-reg
As The addressing bits responsible for the addressing mode used for the source src
S-reg The used Working Register for the source src
Ad The addressing bits responsible for the addressing mode used for the destination dst
D-reg The used Working Register for the destination dst
B/W Byte or word operation: 0: word operation
1: byte operation

Note: Destination Address

The destination can be anywhere in the 64kByte address range. Operations that write data back should use address ranges into those data can be written, otherwise the data is lost.

5.3.1 Double operand instructions

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="4">Opcode</td><td colspan="4">S-Reg</td><td>Ad</td><td>B/W</td><td colspan="2">As</td><td colspan="4">D-Reg</td></tr>
</table>

Figure 5.5: Double Operand Instruction Format

			Status Bits			
			V	N	Z	C
MOV	src,dst	src -> dst	-	-	-	-
ADD	src,dst	src + dst -> dst	*	*	*	*
ADDC	src,dst	src + dst + C -> dst	*	*	*	*
SUB	src,dst	dst + .not.src + 1 -> dst	*	*	*	*
SUBC	src,dst	dst + .not.src + C -> dst	*	*	*	*
CMP	src,dst	dst - src	*	*	*	*
DADD	src,dst	src + dst + C -> dst (dec)	*	*	*	*
AND	src,dst	src .and. dst -> dst	0	*	*	*
BIT	src,dst	src .and. dst	0	*	*	*
BIC	src,dst	.not.src .and. dst -> dst	-	-	-	-
BIS	src,dst	src .or. dst -> dst	-	-	-	-
XOR	src,dst	src .xor. dst -> dst	*	*	*	*

Note: Instructions CMP and SUB

The instructions CMP and SUB are identical except the storage of the result. The same is true for the BIT and the AND instruction.

5.3.2 Single operand instructions

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode									B/W	Ad		D/S-Reg			

Figure 5.6: Single Operand Instruction Format

			Status Bits			
			V	N	Z	C
RRC	dst	C -> MSB -> ....LSB -> C	*	*	*	*
RRA	dst	MSB -> MSB ->...LSB -> C	0	*	*	*
PUSH	src	SP - 2 -> SP, src -> @SP	-	-	-	-
SWPB	dst	swap bytes	-	-	-	-
CALL	dst	SP - 2 -> SP PC+2 -> stack, dst -> PC	-	-	-	-
RETI		TOS -> SR, SP <- SP + 2 TOS -> PC, SP <- SP + 2	x	x	x	x
SXT	dst	Bit7 -> Bit8 .... Bit15	0	*	*	*

All addressing modes are possible for the CALL instruction. If the Symbolic Mode (ADDRESS), the Immediate Mode (#N), the Absolute Mode (&EDE) or the Indexed Mode X(Rn)) is used, the instructions have the address information contained in the following word.

5.3.3 Conditional Jumps

The conditional jumps allow program branches relative to the Program Counter. The possible range is from -511 to +512 words relative to the PC state of the Jump instruction. The 10-bit PC offset is treated as a signed 10-bit value which is doubled and added to the Program Counter. The conditional jumps do not affect the Status Bits.

The instruction code fetch and PC increment technique used ends with the formula:

$$PC_{new} = PC_{old} + 2 + PC_{offset} * 2$$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode			C			10-bit PC offset									

Figure 5.7: Conditional Jump Instruction Format

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

The instruction code fetch and PC increment technique used ends with the formula:

$$PC_{new} = PC_{old} + 2 + PC_{offset} * 2$$

5.3.4 Short form of emulated instructions

The basic instructions together with the constant generator form the emulated instruction which supplies popular instructions. The status bits are set according to the result of the basic instructions.

Mnemonic		Description	Statusbits				Emulation	
			V	N	Z	C		
Arithmetical instructions								
ADC[.W]	dst	Add carry to destination	*	*	*	*	ADDC	#0,dst
ADC.B	dst	Add carry to destination	*	*	*	*	ADDC.B	#0,dst
DADC[.W]	dst	Add carry decimal to destination	*	*	*	*	DADD	#0,dst
DADC.B	dst	Add carry decimal to destination	*	*	*	*	DADD.B	#0,dst
DEC[.W]	dst	Decrement destination	*	*	*	*	SUB	#1,dst
DEC.B	dst	Decrement destination	*	*	*	*	SUB.B	#1,dst
DECD[.W]	dst	Double-Decrement destination	*	*	*	*	SUB	#2,dst
DECD.B	dst	Double-Decrement destination	*	*	*	*	SUB.B	#2,dst
INC[.W]	dst	Increment destination	*	*	*	*	ADD	#1,dst
INC.B	dst	Increment destination	*	*	*	*	ADD.B	#1,dst
INCD[.W]	dst	Increment destination	*	*	*	*	ADD	#2,dst
INCD.B	dst	Increment destination	*	*	*	*	ADD.B	#2,dst
SBC[.W]	dst	Subtract carry from destination	*	*	*	*	SUBC	#0,dst
SBC.B	dst	Subtract carry from destination	*	*	*	*	SUBC.B	#0,dst
Logical instructions								
INV[.W]	dst	Invert destination	*	*	*	*	XOR	#0FFFFh,dst
INV.B	dst	Invert destination	*	*	*	*	XOR.B	#0FFFFh,dst
RLA[.W]	dst	Rotate left arithmetically	*	*	*	*	ADD	dst,dst
RLA.B	dst	Rotate left arithmetically	*	*	*	*	ADD.B	dst,dst
RLC[.W]	dst	Rotate left through carry	*	*	*	*	ADDC	dst,dst
RLC.B	dst	Rotate left through carry	*	*	*	*	ADDC.B	dst,dst
Data instructions (common use)								
CLR[.W]		lear destination	-	-	-	-	MOV	#0,dst
CLR.B		lear destination	-	-	-	-	MOV.B	#0,dst
CLRC		lear carry bit	-	-	-	0	BIC	#1,SR
CLRN		lear negative bit	-	0	-	-	BIC	#4,SR
CLRZ		lear zero bit	-	-	0	-	BIC	#2,SR
POP	dst	Item from stack	-	-	-	-	MOV	@SP+,dst
SETC		Set carry bit	-	-	-	1	BIS	#1,SR
SETN		Set negative bit	-	1	-	-	BIS	#4,SR
SETZ		Set zero bit	-	-	1	-	BIS	#2,SR
TST[.W]	dst	Test destination	0	*	*	*	CMP	#0,dst
TST.B	dst	Test destination	0	*	*	*	CMP.B	#0,dst
Program flow instructions								
BR	dst	Branch to	-	-	-	-	MOV	dst,PC
DINT		Disable interrupt	-	-	-	-	BIC	#8,SR
EINT		Enable interrupt	-	-	-	-	BIS	#8,SR
NOP		No operation	-	-	-	-	MOV	#0h,#0h
RET		Return from subroutine	-	-	-	-	MOV	@SP+,PC

5.3.5 Miscellaneous

No instructions without operands such as CPUOff etc. are provided. These functions are switched on or off by setting or clearing of the function bits in the Status Register or the appropriate I/O-register. Others are emulated by Dual Operand Instructions.

Some examples are given below:

```
BIC     #1,SR               ; Clear Carry
MOV     #0,#0               ; No Operation
BIC     #8,SR               ; Disable Interrupts
BIS     #28h,SR             ; Enter OscOff Mode
                            ; + enable gen. interrupt GIE
BIS     #18h,SR             ; Enter CPUOff Mode
                            ; + enable gen. interrupt GIE
BIC     #SVCC,ACTL          ; SWITCH SVCC OFF
```

5.4 Instruction map

The following instruction map is a proposal of how to encode the instructions. Room is free for more instructions if needed.

	000	040	080	0C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0
0x																
04x																
08x																
0Cx																
10x	RRC	RRC.B	SWPB		RRA	RRA.B	SXT		PUSH	PUSH.B	CALL		RETI			
14x																
18x																
1Cx																
20x	JNE/JNZ															
24x	JEQ/JZ															
28x	JNC															
2Cx	JC															
30x	JN															
34x	JGE															
38x	JL															
3Cx	JMP															
40x..4Cx	MOV, MOV.B															
50x..5Cx	ADD, ADD.B															
60x..6Cx	ADDC, ADDC.B															
70x..7Cx	SUBC, SUBC.B															
80x..8Cx	SUB, SUB.B															
90x..9Cx	CMP, CMP.B															
A0x..ACx	DADD, DADD.B															
B0x..BCx	BIT, BIT.B															
C0x..CCx	BIC, BIC.B															
D0x..DCx	BIS, BIS.B															
E0x..ECx	XOR, XOR.B															
F0x..FCx	AND, AND.B															

Figure 5.8: Core instruction map

6 Hardware Multiplier

The hardware multiplier is realized as each other 16 bit peripheral module, and not integrated into the CPU. The CPU is unchanged through all configurations, and the instruction set is not modified. It take no extra cycle for multiplication. Both operands are loaded into the multiplier’s register and the result can be accessed immediately after loading the second operand.

Topic **Page**

The Hardware Multiplier Module expands the capabilities of the MSP430 family without changing the basic architecture. Multiplication is possible for:

- 16 x 16 bit
- 16 x 8 bit
- 8 x 16 bit
- 8 x 8 bit

The hardware multiplier module supports three types of multiplication: unsigned multiplication (MPY), signed multiplication (MPYS) and unsigned multiplication and accumulation (MAC).

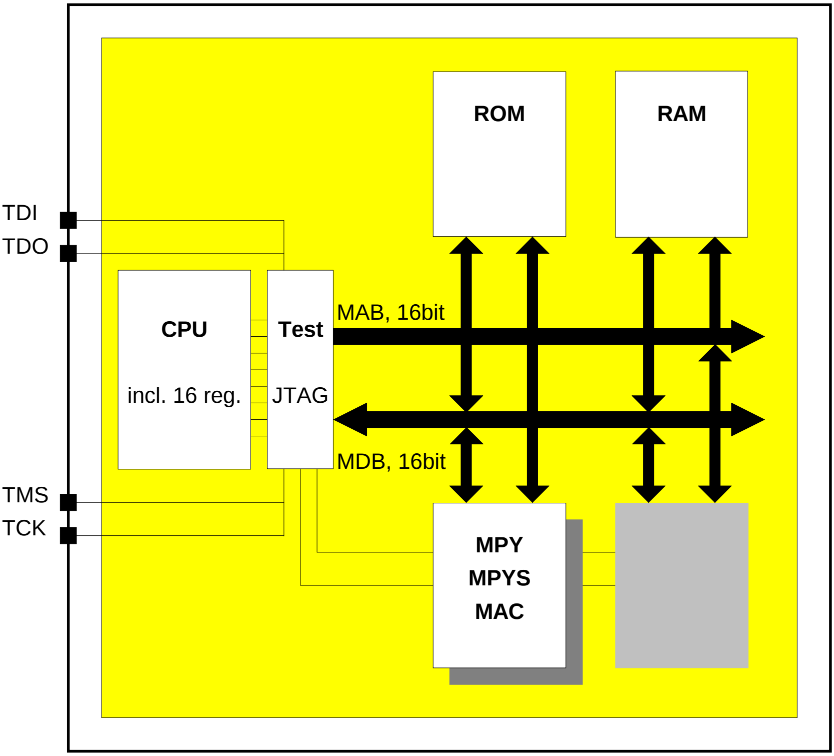


Figure 6.1: Connection of the Hardware Multiplier Module to the Bus System

6.1 Hardware Multiplier Operation

The hardware multiplier has two 16-bit registers for both operands, and three registers where the result of the multiplication is stored. The multiplication is executed correctly when the operand OP1 is written prior of the second operand OP2 to the operands' registers. The type of multiplication is selected when the first operand is written to the appropriate register. Writing the second operand to the appropriate register starts the multiplication. It is completed before the result registers are accessed using indexed address mode for the source operand. Another instruction is needed between the write of the second operand and the access to result registers, when indirect or indirect autoincrement address mode is used. Both operands - transferred to the hardware multiplier - have all seven address mode capabilities.

No instruction for the multiplication is added, which means that the real-time operation and the interrupt latency is unchanged.

Multiply unsigned, 16x16bit, 16x8bit, 8x16bit, 8x8bit

```
********************************************************************
*      TRANSFER BOTH OPERANDS TO THE REGISTERS IN THE HARDWARE     *
*      MULTIPLIER MODULE                                           *
*    USE CONSTANT OPERAND1 AND OPERAND 2 TO IDENTIFY BYTE DATA     *
********************************************************************
OPERAND1  .EQU     0               ; 0:  OPERAND1 IS WORD (16BIT)
                                   ; 8:  OPERAND1 IS BYTE ( 8BIT)
OPERAND2  .EQU     0               ; 0:  OPERAND2 IS WORD (16BIT)
                                   ; 8:  OPERAND2 IS BYTE ( 8BIT)
MPY       .EQU     0130H
MPYS      .EQU     0132H
MAC       .EQU     0134H
OP2       .EQU     0138H
RESLO     .EQU     013AH
RESHI     .EQU     013CH
SUMEXT    .EQU     013EH
          .BSS     OPER1,2,200H
          .BSS     OPER2,2
          .BSS     RAM,8

          .IF OPERAND1=8
          MOV.B    &OPER1,&MPY     ; LOAD 1ST OPERAND,
                                   ; DEFINES ADD. UNSIGNED MULTIPLY
          .ELSE
          MOV      &OPER1,&MPY     ; LOAD 1ST OPERAND,
                                   ; DEFINES ADD. UNSIGNED MULTIPLY
          .ENDIF

          .IF OPERAND1=8
          MOV.B    &OPER2,&OP2     ; LOAD 2ND OPERAND AND START
                                   ; MULTIPLICATION
          .ELSE
          MOV      &OPER2,&OP2     ; LOAD 2ND OPERAND AND START
                                   ; MULTIPLICATION
          .ENDIF

********************************************************************
*      EXAMPLE TO ADD THE RESULT OF THE HARDWARE MULTIPLICATION    *
*      TO THE RAM DATA, 64BITS                                     *
********************************************************************
          ADD      &RESLO,&RAM     ; ADD LOW RESULT TO RAM
          ADDC     &RESHI,&RAM+2   ; ADD HIGH RESULT RO RAM+2
          ADC      &RAM+4          ; ADD CARRY TO EXTENSION WORD
          ADC      &RAM+6          ; IF 64 BIT LENGHT IS USED
```

32 Bytes of program code, 32 execution cycles (16x16bit multiplication)

Multiply signed, 16x16bit, 16x8bit, 8x16bit, 8x8bit

```
********************************************************************
*      TRANSFER BOTH OPERANDS TO THE REGISTERS IN THE HARDWARE     *
*      MULTIPLIER MODULE                                           *
*      IF ONE OF THE OPERANDS IS 8BIT, SIGN EXTENSION IS NEEDED    *
*    USE CONSTANT OPERAND1 AND OPERAND 2 TO IDENTIFY BYTE DATA     *
********************************************************************
OPERAND1  .EQU     0               ; 0:  OPERAND1 IS WORD (16BIT)
                                   ; 8:  OPERAND1 IS BYTE ( 8BIT)
OPERAND2  .EQU     0               ; 0:  OPERAND2 IS WORD (16BIT)
                                   ; 8:  OPERAND2 IS BYTE ( 8BIT)
MPY       .EQU     0130H
MPYS      .EQU     0132H
MAC       .EQU     0134H
OP2       .EQU     0138H
RESLO     .EQU     013AH
RESHI     .EQU     013CH
SUMEXT    .EQU     013EH
          .BSS     OPER1,2,200H
          .BSS     OPER2,2
          .BSS     RAM,8

          .IF OPERAND1=0
          MOV      &OPER1,&MPYS    ; LOAD 1ST (WORD) OPERAND
                                   ; DEFINES ADD. SIGNED MULTIPLY
          .ELSE
          MOV.B    &OPER1,&MPYS    ; LOAD 1ST (BYTE) OPERAND,
                                   ; DEFINES ADD. SIGNED MULTIPLY
          SXT      &MPYS           ; EXPAND BYTE TO SIGNED WORD DATA
          .ENDIF
          .IF OPERAND2=0
          MOV      &OPER2,&OP2     ; LOAD 2ND (WORD) OPERAND  AND
                                   ; START SIGNED MULTIPLICATION
          .ELSE
          MOV.B    &OPER2,&OP2     ; LOAD 2ND (BYTE) OPERAND,
          SXT      &OP2            ; RE-LOAD 2ND OPERAND  AND START
                                   ; SIGNED ‘FINAL’ MULTIPLICATION
          .ENDIF

********************************************************************
*      EXAMPLE TO ADD THE RESULT OF THE HARDWARE MULTIPLICATION    *
*      TO THE RAM DATA, 64BITS                                     *
********************************************************************
          ADD      &RESLO,&RAM     ; ADD LOW RESULT TO RAM
          ADDC     &RESHI,&RAM+2   ; ADD HIGH RESULT RO RAM+2
          ADDC     &SUMEXT,&RAM+4  ; ADD SIGN WORD TO EXTENSION WORD
          ADDC     &SUMEXT,&RAM+6  ; IF 64 BIT LENGHT IS USED
```

36 Bytes program code, 36 execution cycles (16x16bit multiplication)

Multiply unsigned and accumulate, 16x16bit, 16x8bit, 8x16bit, 8x8bit

```
********************************************************************
*      TRANSFER BOTH OPERANDS TO THE REGISTERS IN THE HARDWARE     *
*      MULTIPLIER MODULE                                           *
*      THE RESULT OF THE MULTIPLICATION IS ADDED TO THE CONTENT    *
*      OF BOTH RESULT REGISTERS, RESLO AND RESHI                   *
*      USE CONSTANT OPERAND1 AND OPERAND 2 TO IDENTIFY BYTE DATA   *
********************************************************************
OPERAND1  .EQU     0               ; 0:  OPERAND1 IS WORD (16BIT)
                                   ; 8:  OPERAND1 IS BYTE ( 8BIT)
OPERAND2  .EQU     0               ; 0:  OPERAND2 IS WORD (16BIT)
                                   ; 8:  OPERAND2 IS BYTE ( 8BIT)
MPY       .EQU     0130H
MPYS      .EQU     0132H
MAC       .EQU     0134H
OP2       .EQU     0138H
RESLO     .EQU     013AH
RESHI     .EQU     013CH
SUMEXT    .EQU     013EH
          .BSS     OPER1,2,200H
          .BSS     OPER2,2
          .BSS     RAM,8

          .IF OPERAND1=8
          MOV.B    &OPER1,&MAC     ; LOAD 1ST OPERAND,
                                   ; DEFINES ADD. UNSIGNED MULTIPLY
          .ELSE
          MOV      &OPER1,&MAC     ; LOAD 1ST OPERAND,
                                   ; DEFINES ADD. UNSIGNED MULTIPLY
          .ENDIF

          .IF OPERAND1=8
          MOV.B    &OPER2,&OP2     ; LOAD 2ND OPERAND AND START
                                   ; MULTIPLICATION
          .ELSE
          MOV      &OPER2,&OP2     ; LOAD 2ND OPERAND AND START
                                   ; MULTIPLICATION
          .ENDIF
```

```
********************************************************************
*      EXAMPLE TO ADD THE RESULT OF THE HARDWARE MULTIPLICATION    *
*      TO THE RAM DATA, 64BITS                                     *
*      THE RESULT OF THE MULTIPLICATION IS HELD IN RESLO AND       *
*      RESHI REGISTERS. THE UPPER TWO WORDS IN THE EXAMPLE ARE     *
*      FURTHER LOCATED IN THEIR RAM LOCATION                       *
********************************************************************
         ADDC     &SUMEXT,&RAM+4  ; ADD SUMEXTENSION TO RAM+4
         ADC      &RAM+6          ; IF 64 BIT LENGHT IS USED
```

32 Bytes program code, 32 execution cycles (16x16bit multiplication)

6.2 Hardware Multiplier Registers

The hardware multiplier module hardware is word structured, but it can be accessed by word or byte processing instructions.

Register	short form	Register type	Address	Initial state
● Multiply Unsigned/Op.1	MPY	Type of read/write	0130h	unchanged
● Multiply Signed/Operand1	MPYS	Type of read/write	0132h	unchanged
● Multiply+Accumulate/Op.1	MAC	Type of read/write	0134h	unchanged
● Reserved			0136h	unchanged
● Second Operand	OP2	Type of read/write	0138h	unchanged
● Result Low Word	ResLo	Type of read/write	013Ah	undefined
● Result High Word	ResHi	Type of read/write	013Ch	undefined
● Sum Extend register	SumExt	Type of read	013Eh	undefined

There are two registers implemented for both operands, operand OP1 and OP2. The operand 1 use three different addresses to address the same register. The different address information is decoded and defines the type of multiplication - unsigned, signed and unsigned+accumulate.

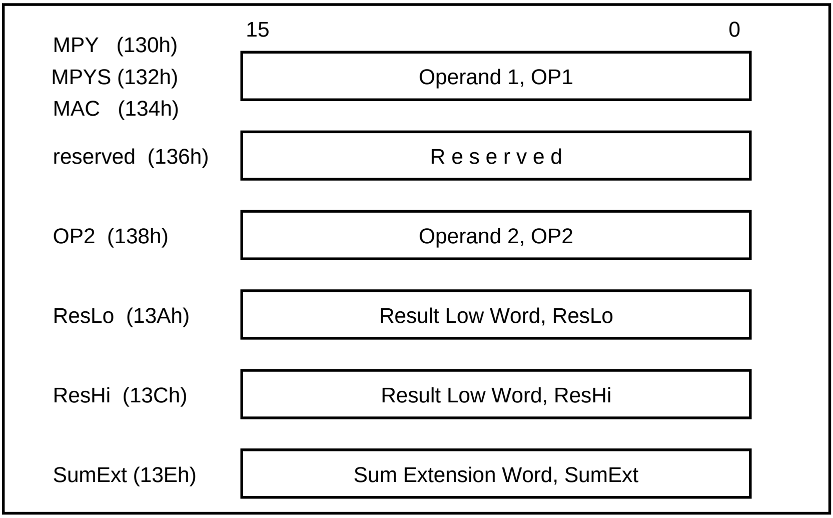


Figure 6.2: Registers of the Hardware Multiplier

The result is located in two word registers, the result high RESHI and result low RESLO register. The sum extend register SumExt holds the sign of the result of a signed 16x16bit multiplication, or holds the overflow of the multiply and accumulate (MAC) operation.
All registers have the LSB at bit0 and the MSB at bit7 (byte data) or bit15 (word data).

6.3 Hardware Multiplier Special Function bits

The hardware multiplier module completes all multiply operations fast without interrupt intervention, and therefore no special function bits are used.

6.4 Hardware Multiplier Software Restrictions

Two special cases need attention when the hardware multiplier is used:

- Use of indirect or indirect autoincrement address mode to process the result
- Use of the hardware multiplier in an interrupt routine

6.4.1 Hardware Multiplier Software Restrictions - Address mode

The access to the result of a multiplication works in indexed, indirect or indirect autoincrement mode. The access to the result registers can be done without any restrictions if indexed address mode is used - including symbolic and absolute address mode. Whenever the indirect and indirect autoincrement address mode is used to access the result registers, at least one instruction between the load of the second operand and access to one of the result registers is needed:

```
********************************************************************
*    EXAMPLE: MULTIPLY OPERAND1 AND OPERAND 2                      *
********************************************************************
RESLO     .SET      013AH           ; RESLO = ADDRESS OF RESLO
          PUSH      R5              ; R5 WILL HOLD THE ADDRESS OF
          MOV       #RESLO,R5       ; THE RESLO REGISTER

          MOV       &OPER1,&MPY     ; LOAD 1ST OPERAND,
                                    ; DEFINES ADD. UNSIGNED MULTIPLY
          MOV       &OPER2,&OP2     ; LOAD 2ND OPERAND AND START
                                    ; MULTIPLICATION

********************************************************************
*      EXAMPLE TO ADD THE RESULT OF THE HARDWARE MULTIPLICATION    *
*      TO THE RAM DATA, 64BITS                                     *
********************************************************************
          NOP                       ; MIN. ONE CYLES BETWEEN MOVING
                                    ; THE OPERAND2 TO HW-MULTIPIER
                                    ; AND PROCESSSING THE RESULT WITH
                                    ; INDIRECT ADDRESS MODE
          ADD       @R5+,&RAM       ; ADD LOW RESULT TO RAM
          ADDC      @R5,&RAM+2      ; ADD HIGH RESULT RO RAM+2
          ADC       &RAM+4          ; ADD CARRY TO EXTENSION WORD
          ADC       &RAM+6          ; IF 64 BIT LENGHT IS USED

          POP       R5
```

The example shows that the indirect or indirect address mode - used to transfer the result of a multiplication to the destination - needs more cycles and code than the absolute address mode. Obviously there is no special need to access the absolute hardware multiplier using indirect addressing mode.

6.4.2 Hardware Multiplier Software Restrictions - Interrupt Routines

The entire multiplication routine uses three major steps:

- move the operand OP1 to the hardware multiplier, the type of multiplication is defined
- move the operand OP2 to the hardware multiplier, the multiplication is started
- process the result of the multiplication in RESLO, RESHI SUMEXT registers

The following considerations are useful if the main routines use hardware multiplication. If no hardware multiplication is used in the main routines, the multiplication in an interrupt routine is protected from further interrupts, since the general interrupt enable bit is reset after entering the interrupt service routine. Normally a multiplication with the entire data processing should be done outside an interrupt routine following the rule: Keep interrupt routines as short as possible.

A multiplication in an interrupt routine has some feedback to the multiplication routine in the main routine:

Interrupt occurs after the first operand OP1 is transferred into hardware multiplier

The two LSBs of the first operand's address defines the type of multiplication. This information can not be recovered by any later operation. The interrupt should not be able to be accepted between the first two steps - move operand OP1 and operand OP2 to the multiplier.

Interrupt occurs after the second operand OP2 is transferred into hardware multiplier

After the first two steps, the result is already in the corresponding registers RESLO, RESHI and SUMEXT and can be saved e.g. on the stack (*PUSH* ...) and restored after completing another multiplication (*POP* ...). But additional code and cycles in the interrupt routine are used. This can be avoided when the entire multiplication routine is protected by disabling any interrupt (*DINT*) before entering the multiplication routine and enabling interrupts (*EINT*) after the multiplication routine is completed. A negative impact on this method is that the critical interrupt latency is increased drastically for events which occur during this period.

General recommendation

In general a hardware multiplication within an interrupt routine should be avoided when a hardware multiplication is already used in main routines. The application specific software, applied libraries or other included software should be taken into consideration. The different methods discussed show more negative implications than positive. Following the general recommendation to shorten interrupt routines is the best practice.

7 Oscillator and System Clock Generator

Topic Page

The oscillator and the system clock generator follow the major targets of low system cost and low power consumption.

External component count is reduced down to a commonly used crystal to achieve the target of low system cost. The use of a low frequency crystal and oscillator combined with a multiplier meets system cycle speed and the second target of low power consumption.

Features for current limited applications

Special other features are obviously mandatory in very low power consuming devices that use the various extended operating modes. These features include startup timing, long term frequency stability with voltage, temperature and time, and a highly-stable time base for real time clocks.

Current limited real-time applications demand two conflicting requirements: low system clock frequency for energy conservation, and high system clock frequency for fast reaction to requesting events. Especially battery based applications are very critical with respect to current consumption. Response to external events or time requests typically requires occasionally high speed in real-time applications.

A processor clock generator with fast start-up allowing exhaustive use of different power dissipation modes could theoretically solve this dilemma. On the other hand, fast start-up is closely combined with unacceptably low frequency stability. Design with multiple clock sources or different clock operations could take into account the clock requirements of certain peripheral components for real-time applications such as low frequency communication, display (e.g. LCD), timers and counters.

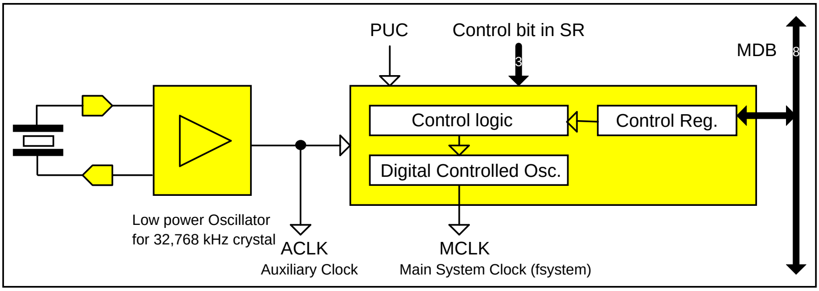


Figure 7.1: Principle of Clock Generation

The output of the low frequency crystal oscillator provides the clock signals for the CPU operation and the peripheral modules. The oscillator of the MSP430 operates with the widely used crystal, without any external components.

The different requirements of CPU and modules, from the point of view of current consumption objectives, requires the use of two clock signals:

- Auxiliary Clock ACLK with crystal's frequency
- System Clock MCLK with a higher frequency: N x $f_{crystal.}$

7.1 Crystal Oscillator

The special design of the oscillator supports the features of low current consumption and the use of a 32 768Hz crystal. The crystal is connected to two pins without any other external components. All components for stabilizing the operation state or phase shifter capacitors are integrated.

Two factors dominate the choice of the well-known and widely used watch crystal:

- oscillator and time base for low current consumption
- optimize system costs.

The oscillator starts operating after applying VCC due to reset of the control bit OscOff in the Status Register SR. It can be stopped by setting the OscOff bit.

15	8	7							0
reserved for future enhancements	V	SCG 1	SCG 0	Osc Off	CPU Off	GIE	N	Z	C
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Figure 7.2: Status Register SR

7.2 Processor Clock Generator

The System Clock of controllers has to meet different requirements, according to the application and system conditions:

- High frequency, to react fast onto system hardware requests or events
- Low frequency, to minimize current consumption, EMI, ...
- Stable frequency for timer applications e.g. real time clock RTC
- Low-Q oscillators to enable start-stop operation with 'zero' delay to operation.

All the conflicting but essential requests can not be handled, either with high-Q, fast frequency crystals, or with low-Q RC-type oscillators. Proper current consumption and the frequency stability mentioned require the use of a low frequency crystal. The compromise used in the MSP430 is to use a low frequency crystal, and to multiply its frequency up to the nominal operating range:

$$\boxed{f_{System} = N \text{ x } f_{crystal}}$$

Different ways for multiplication of the crystal's frequency to the system frequency are known, and several are practiced. The most known methods are the Phase-Locked-Loop PLL technique, and Frequency-Locked-Loop FLL.

The PLL technique has two major disadvantages in systems with frequently and time-undefined intermitted operating modes. PLL's are systems following second order response. All the on-off operating modes result in out-of-phase conditions, and therefore in continuous 'limp-mode' handling. The wide ranges of off-time conditions conflict with the use of analog filter-integrators in the closed loop. Changes in the capacitor's charge automatically result in phase and/or frequency deviations and an improper frequency until the system is in phase.

The FLL technique, in combination with a digital controlled oscillator (DCO), avoids both serious problems.

The major features of the DCO are:

- fast start-up
- digital (not analogs) control signals.

Beside these advantages one item needs careful consideration: the variation of the frequency with the supply voltage and temperature.

The DCO is absolutely monotone.

The FLL operates as a continuous frequency integrator. An up/down counter that follows the loop control corrects permanently the multiplication factor N. The follow-up or up-date rate is identically to the crystal's frequency rate. Using a 32,768 kHz crystal the rate is 30.5µs.

The accumulated frequency error is the same as that of the crystal's. The time deviation from one machine cycle to another is typically less than 10%.

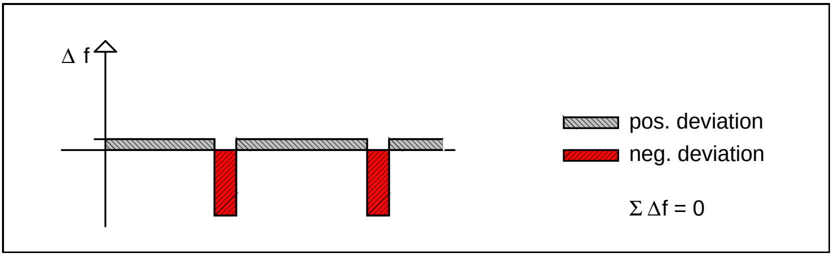


Figure 7.3: System frequency vs. time

The start-up operation of the system clock depends on the previous machine state. During a PUC the DCO is reset to its lowest possible frequency. The control logic starts operation immediately after removing the PUC condition. Proper working condition for the control logic needs the presence of stable crystal oscillation.

The frequency integrator of 10bit length controls the frequency at which the DCO is running with. The integrator - starting at zero digital value after PUC - counts up to run the frequency f_{System} at the selected value N. It takes slightly more periods of the crystal input than the suggested number of 10bit or 1024, if the maximum length of the frequency integrator is needed. The control logic system operates aperiodically.

Applications that run the controller with intermitted operation need some attention to the conditions of handling the system frequency control conditions. The correction of the frequency integrator is possible each period of the crystal (30.5 µs @ 32,768 Hz) plus the period of f_{System}/N. Longer integration periods are mandatory to avoid accumulating deviations in time.

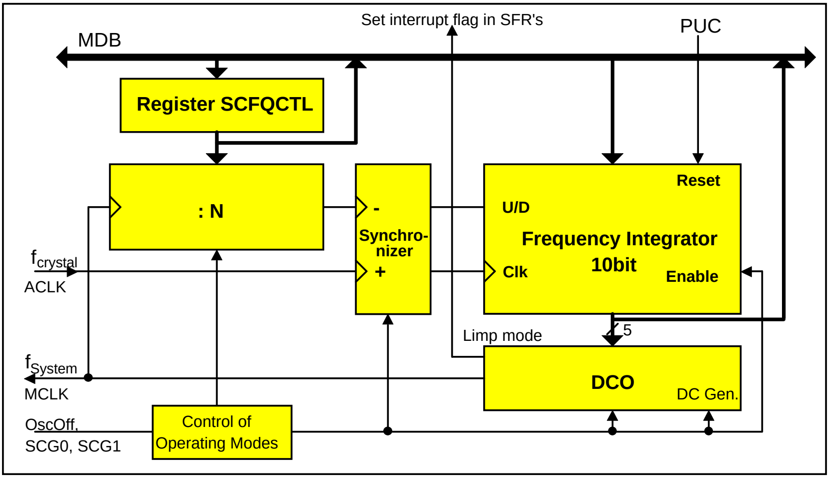


Figure 7.4: Schematic of system frequency generator

Two flags are incorporated in the special function register which allow the application program to get back control over the system, if the digital controlled oscillator is at its upper or lower frequency limit.

The operation at the upper or lower limit can be easily detected by controlling the frequency integrator via access to SCFI0 and/or SCFI1.

7.3 System Clock Operating Modes

The system clock generator and crystal oscillator are controlled by three signals. These signals are located in the status register SR and are reset during the four different power-up conditions.

These three control signals provide the system application with different operating conditions and maximum flexibility to optimize overall system power consumption. During some combinations of the three control signals the system clock MCLK stops operation; the existing value of the frequency integrator remains.

SCG1	SCG0	OscOff	Crystal oscillator	DC Generator	DCO	Loop control	Comments
0	0	0	ON	ON	ON	ON	Condition after PUC Crystal and DC oscillator are active Loop control is operating
0	1	0	ON	ON	ON	OFF	Low Power Mode **LPM1** Crystal and DC oscillator are active Loop control is off
1	0	0	ON	ON	OFF	OFF	Low Power Mode **LPM2** Crystal oscillator and DC Generator are active DCO and Loop control are off
1	1	0	ON	OFF	OFF	OFF	Low Power Mode **LPM3** Crystal oscillator is active All other functions are off
X	X	1	OFF	OFF	OFF	OFF	Low Power Mode **LPM4** All functions are disabled f_{MCLK} = f_{AClk} = 0Hz

The three control signals provide five different power down modes, supporting ultra-low power applications, by making intensive use of them. All these different modes provide the system application with the potential for operation with the smallest time slot possible, and the optimized current consumption in each time slot.

The SCG0 bit controls the FLL loop if it is operating (SCG0 is reset) or off (SCG0 is set).

Starting from PUC

The system clock control register SCFQCTL is set to 01Fh with PUC, and the frequency integrator is reset. The reset of the frequency integrator sets the system frequency to its lowest value, and counts up continuously until it locks at a system frequency that is equal to N times the crystal frequency.

Low Power Mode LPM4, Oscillator off
During the oscillator off mode all parts of the processor are inactive, and the current consumption is at its lowest limit. Starting with operation is only possible after power-up circuitry has detected a low supply voltage condition or any external interrupt event that will request an interrupt asynchronously. The appropriate enable for interrupt sources should be applied during the program flow.

The start-up sequence of the system clock generator out of oscillator off mode:

- the present system frequency defined by the output value of the frequency integrator and the DCO characteristic will continue running
- the frequency integrator is continuously counted down with the frequency of f_{System}/N till the DCO is running at its lowest frequency as long as the crystal oscillator has not started operation
- after the crystal oscillator starts operation, the loop control will settle the frequency integrator to the value following $f_{System} = N * f_{crystal}$.

Low Power Mode LPM3, DC Generator off
During the DC generator off mode only the crystal oscillator is active. The DC current of the DC generator that sets the basic timing conditions is switched off. The power consumption constraints force high impedance design. The start of the DCO from power-down mode with DC generator off can take some time (t_{DCGon}) to run with the selected frequency. The time is in the range between ns up to µs.

Low Power Mode LPM2, DCO off
The crystal oscillator and the DC generator are still active during LPM2, and an immediate start is possible. The start-up delay is limited to some gate delays.

Low Power Mode LPM1, Frequency-lock-loop off
The crystal oscillator, the DC generator and the DCO are still active during LPM1. The processor with all its peripheral modules is fully functional without any limitation. The frequency is determined from the output value of the frequency integrator. This value, with the characteristic of the DCO, determines the frequency of the MCLK signal that is identical to the system frequency f_{System}.
There is no start-up delay: the oscillator is already running. The loop control is activated asynchronously and with a slight frequency variation, but it settles fast and aperiodically.

7.4 System Clock Control Register

The system clock generator interacts with other processor parts via three general module registers and the special function registers. The general module registers are mapped into the lower peripheral file address range where all byte modules are located. Three control lines for the operating states, SCG1, SCG0 and OscOff, are supplied from the status register SR of the CPU.

7.4.1 General Module Registers

Two eight bit registers control the system clock generator. The user's software loads one of the registers with the multiplication factor N. The other register holds control bits or signals used for various operating modes. It should be accessed using byte instructions.

System Clock Frequency Control

SCFQCTL 052h	7							0
	M	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	rw-0	rw-0	rw-0	rw-1	rw-1	rw-1	rw-1	rw-1

The content of register SCFQCTL controls the multiplication of the crystal's frequency. The seven bits indicates a range of 3+1 to 127+1.

$$f_{System} = (x*2^6 + x*2^5 + x*2^4 + x*2^3 + x*2^2 + x*2^1 + x*2^0 + 1) * f_{crystal}$$

The default value in SCFQCTL after PUC was active is 31, which results in a factor of 32.
The range of the f_{System} is theoretical and depends on the adjustable frequency range of the DCO (see more information in electrical characteristics).

Note: Multiplication factor in SCG

The content of register SCFQCTL (2^6 to 2^0) controls the multiplication of the crystal's frequency. The seven bits must be in the range of 3 to 127. Any value below 3 results in unpredictable operation, but also any value than 127 will force the MCLK frequency above the device specification.

System Clock Frequency Integrator

SCFI0
050h

7							0
0	0	0	FN_4	FN_3	FN_2	2^1	2^0

r r r r rw-0 rw-0 rw-0 rw-0 rw-0

SCFI1
051h

7							0
2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

The output of the frequency integrator controls the DCO. This value can be read using the appropriate address of SCFI1 and SCFI0. The digital representation is:

$$N_{DCO} = (x* 2^9 + x*2^8 + x*2^7 + x*2^6 + x*2^5) + (1-M)*(x*2^4 + x*2^3 + x*2^2 + x*2^1 + x*2^0)$$

SCFI0, Bit 1...3: The three bits in the SC control register 0 define the nominal frequency of the DCO.

FN_4	FN_3	FN_2	Frequency
0	0	0	f_{NOM}
0	0	1	2 x f_{NOM}
0	1	X	3 x f_{NOM}
1	X	X	4 x f_{NOM}

7.4.2 Special function register bits, System Clock Generator related

Two bits in the SFR address range handle the system control interaction according to the function implemented in the SCG. These three bits are:

- OscFault Interrupt Flag OFIFG (located in IFG1.1, initial state is unchanged)
- OscFault Interrupt Enable OFIE (located in IE1.1, initial state is reset).

The interrupt flag is part of a multiple source interrupt request. The same interrupt vector is also used for the event at the , RST/NMI-pin when NMI function is selected. The interrupt is defined to be non-maskable. Non-maskable implies that the general interrupt enable bit GIE can not disable the interrupt request. Since the interrupt shares the same interrupt vector and an oscillator fault is active after PUC, the interrupt flag is not automatically reset.

Three different situations should be handled by the software:

- After PUC, a proper sequence should be programmed to identify or to set an oscillator condition that prevents active level at OscFault signal, and therefore a permanently set of OFIFG. The OFIFG should be reset by software.
  PUC resets the OFIE bit and no interrupt is requested.

- When an interrupt from the OscFault signal was requested and serviced, the interrupt enable bit OFIE is reset automatically to disable further continuous interrupt requests until proper response from the software conducts to a inactive OscFault signal. After reaching the inactive state, the OFIE bit can be set again following the general rules of module interrupts. An oscillator fault event is not affected by the general interrupt enable bit GIE.
- The interrupt flag OFIFG can be used to identify the interrupt source at the beginning of the interrupt service routine. The OFIFG is set independently of an additional NMI event and is dominant.

Note: Interrupt flag OFIFG

The interrupt flag OFIFG remains set when an interrupt request has been accepted and serviced. This is mandatory, because it is a multiple source interrupt together with NMI interrupt and it indicates to the software interrupt handle the event of an oscillator fault. Servicing first the OFIFG condition gives this event priority over the NMI event.

7.5 DCO Characteristic - typical

The digital controlled oscillator varies with temperature and supply voltage. Running the frequency loop this is unimportant for the application, because the period of control is identical with the period of the ACLK signal. With a 32,768Hz crystal, it is 30.5µs.

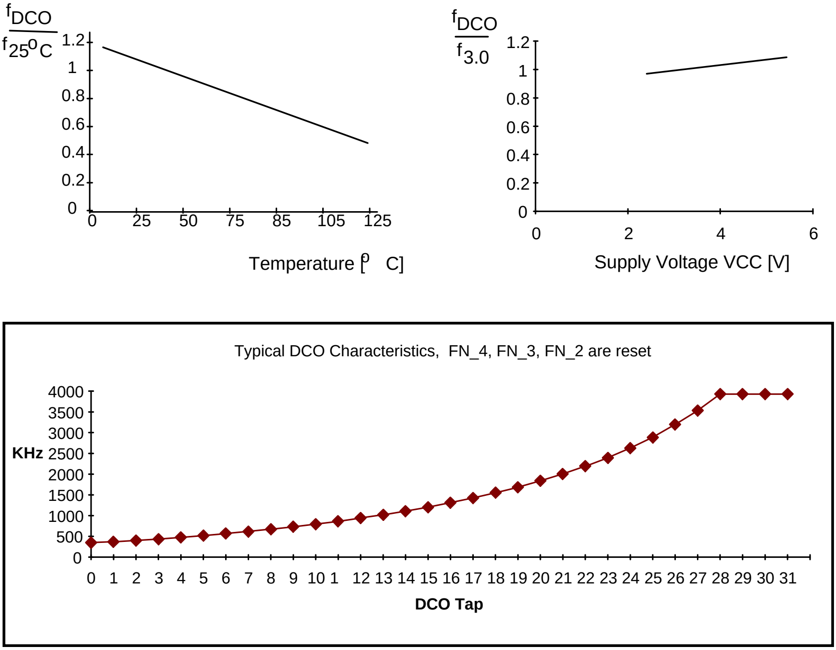


Figure 7.5: DCO Characteristics

Note: DCO Taps

The five most significant bits in the System Clock Frequency Integrator register SCFI1 are feed into the DCO. If the modulation bit M from the register SCFQCTL is set, only the DCO taps are determining the system frequency.

8 Digital I/O Configuration

Topic **Page**

8.1 General Port P0

The general port P0 incorporates all the functions to individually select the function of each pin and to use each signal as an interrupt source.
The six registers are used for the control of the Port's I/O pins.
The general module registers are mapped into the lower peripheral file address range where all byte modules are located. The register should be accessed with byte instructions, using absolute address mode.

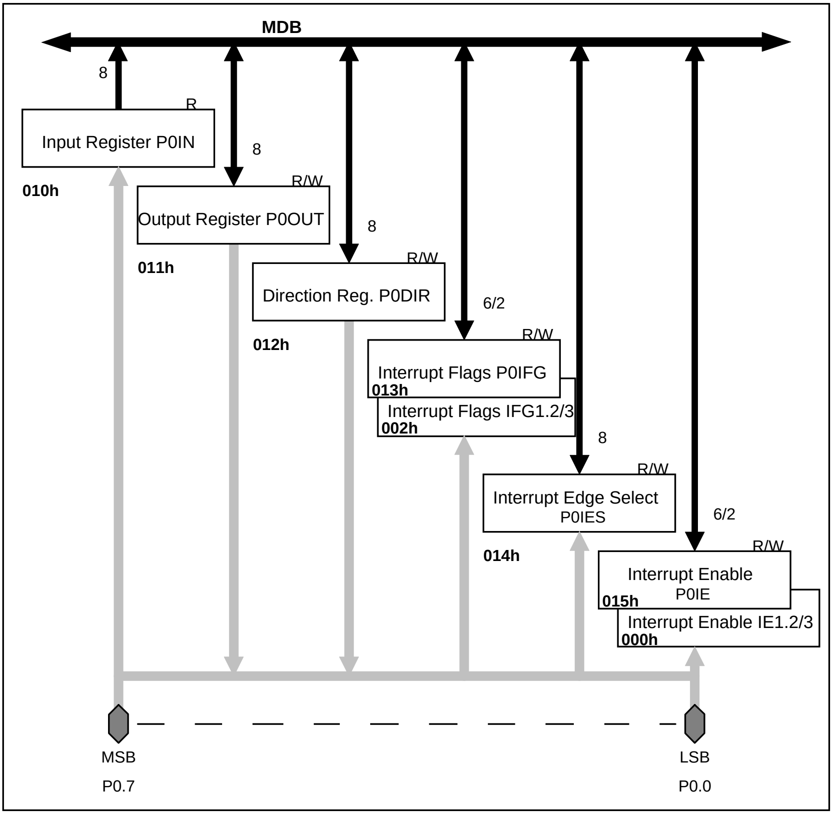


Figure 8.1: Port P0 Configuration

8.1.1 Port P0 Control Registers

The Port P0 is connected to the processor core via the 8-bit MDB structure and MAB. It should be accessed via byte instructions.

The six control registers give maximum flexibility of digital input/output to the application:

- All individual I/O bits are programmable independently:
- Any combination of input, output and interrupt condition is possible.
- Interrupt processing of external events is fully implemented for all eight bits of the port P0.

The six registers are:

Register	short form	Register type	Address	Initial State
• Input register:	P0IN	read only	010h	-----
• Output register:	P0OUT	read/write	011h	unchanged
• Direction register:	P0DIR	read/write	012h	reset
• Interrupt Flags:	P0IFG	read/write	013h	reset
• Interrupt Edge Select:	P0IES	read/write	014h	unchanged
• Interrupt Enable:	P0IE	read/write	015h	reset

All these registers contain eight bits except the two LSBs in the interrupt flag register, and the interrupt enable register. These two bits are included in the special function register SFR. The registers should be accessed with byte instructions.

Input Register P0IN
The input register is a read-only register to scan the signals at the I/O pins. The direction of the pin should be selected for input.

> **Note: Writing to read only register P0IN**
>
> Writing to this read-only register results in an increased current consumption as long as the write is active.

Output Register P0OUT
The Output Register shows the information of the output buffer, an eight bit register that contains the information output at the I/O pins if used as outputs. The output buffer can be modified by all instructions that write to a destination. If read, the contents of the output buffer are read independently of the direction. A direction change does not modify the output buffer contents.

Direction Register P0DIR

This register contains eight independent bits that define the direction of the I/O pin. All bits are reset by PUC:

Bit = 0: The I/O pin is switched to input direction
Bit = 1: The I/O pin is switched to output direction

Interrupt Flags P0IFG

This register contains six flags that contain information if an interrupt is pending or not- according to the corresponding I/O pin:

P0IFG
013h

7							0
P0IFG.7	P0IFG.6	P0IFG.5	P0IFG.4	P0IFG.3	P0IFG.2		
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r0	r0

Bit = 0: No interrupt is pending
Bit = 1: An interrupt is pending due to a transition at the I/O pin. Manipulation on P0OUT and P0DIR can also set bits of P0IFG.

Writing a zero to an Interrupt Flag resets it.
The six flags are located in bit 7 to 2 corresponding to the pins P0.7 to P0.2. The remaining interrupt flags of pin P0.1 and P0.0 are located in the SFRs.

Note: Interrupt Flags P0FLG.2...7

The Interrupt Flags P0FLG.2 to P0FLG.7 use only one interrupt vector: it is a multiple source interrupt vector. The interrupt flags P0IFG.2 to P0IFG.7 are not reset automatically when any interrupt from these events is served. The software decides which event will be served and should reset the appropriate flag.

Any external interrupt event should be as long as 1.5 times MCLK or longer to ensure that it is accepted and the corresponding interrupt flag is set.

Interrupt Edge Select P0IES

This register contains a bit for each I/O pin that selects which transition triggers the interrupt flag. All eight bits corresponding to pin P0.7 to P0.0 are located in this register. The bits have the following meaning:

Bit = 0: The interrupt flag is set with LO/HI transition
Bit = 1: The interrupt flag is set with HI/LO transition

Note: Change of P0IES bit(s)

Any change of P0IES bit(s) may result in setting the associated interrupt flags.

Interrupt Enable P0IE

This register contains a bit for six I/O pins to enable interrupt request on an interrupt event. Two interrupt enable bits for P0.0 and P0.1 are located in special function register IE1.2 and IE1.3. Six bits corresponding to pin P0.7 to P0.2 are located in the P0IE register.

	7							0
P0IE 015h	P0IE.7	P0IE.6	P0IE.5	P0IE.4	P0IE.3	P0IE.2		
	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r0	r0

The bits have the following meaning:

Bit = 0: The interrupt request is disabled
Bit = 1: The interrupt request is enabled

Note: Port0 interrupt sensitivity

Only transitions, not static levels cause interrupts.

The interrupt routine must reset the multiple-use Interrupt Flags P0IFG.2... P0IFG.7. The single source flags P0IFG.0 and P0IFG.1 are reset when they are serviced.

If an Interrupt Flag is still set (because the transition occurred during the interrupt routine) when the *RETI* instruction is executed, an interrupt occurs again after *RETI* is completed. This ensures, that each transition is seen by the software.

8.1.2 Port P0 Schematic

Port P0, Bits P0.3 to P0.7

The pin logic of each individual signal of port P0 is built from five identically register bits - P0DIR, P0OUT, P0IFG, P0IE, P0IES - and one read-only input buffer - P0IN. The bits three to seven are identically designed:

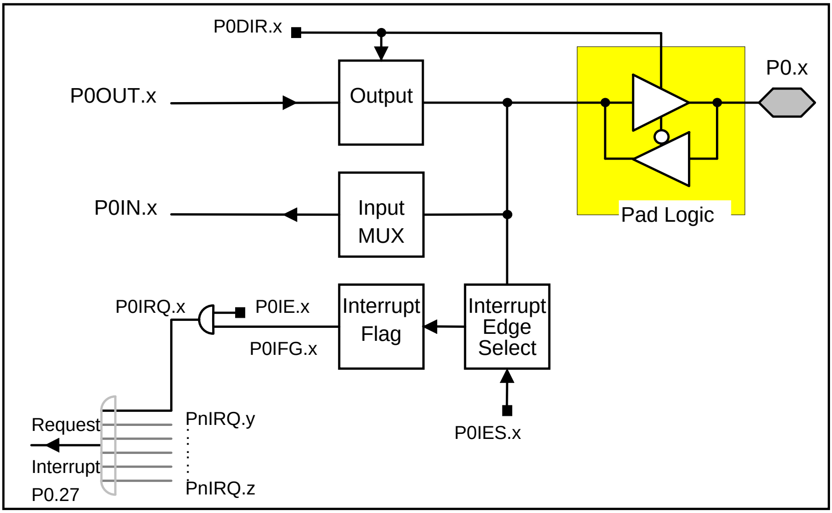


Figure 8.2: Schematic of bits P0.7 to P0.3

The interrupt flag may be set by a relevant input condition, but also by the software. Additionally, when the direction control bit or the interrupt edge select bit are modified a trigger condition may occur.

The port P0 bits two to seven share one common interrupt vector. The interrupt flags are not automatically reset after the P0.27 interrupt request was accepted. The individual flags P0IFG.2 to P0IFG.7 should be reset by software preferably in the corresponding interrupt service routine.

Port P0, Bit P0.2

The bit two is slightly different from bits three to seven. The output signal can be determined either by the port P0OUT.2 bit or by the 8bit Timer/Counter's signal TXD. Whenever output control register bit TXE is set TXD signal is selected to be the relevant output signal and the pad logic is switched to the output, independent of the direction control bit P0DIR.2:

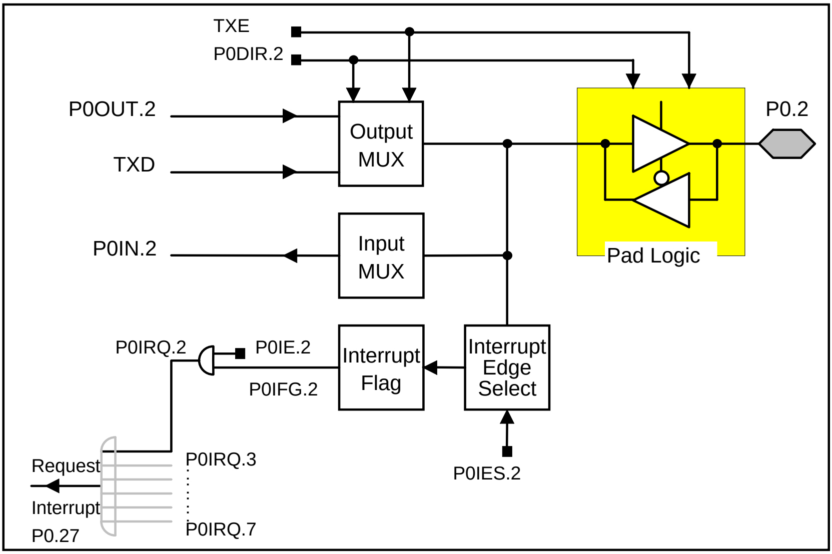


Figure 8.3: Schematic of bit P0.2

The interrupt flag P0IFG.2 shares the interrupt vector with interrupt flags P0IFG.3 to P0IFG.7.

Port P0, Bit P0.1
The bit one is slightly different from bits three to seven. The interrupt signal can be selected to be sourced whether by the input signal at the pin P0.1 or by the 8bit Timer/Counter's signal Carry. Whenever the interrupt source control bit ISCTL in the 8bit Timer/Counter control register TCCTL is set, the interrupt source is switched from the P0.1 pin to the Carry signal from the counter in the 8bit Timer/Counter:

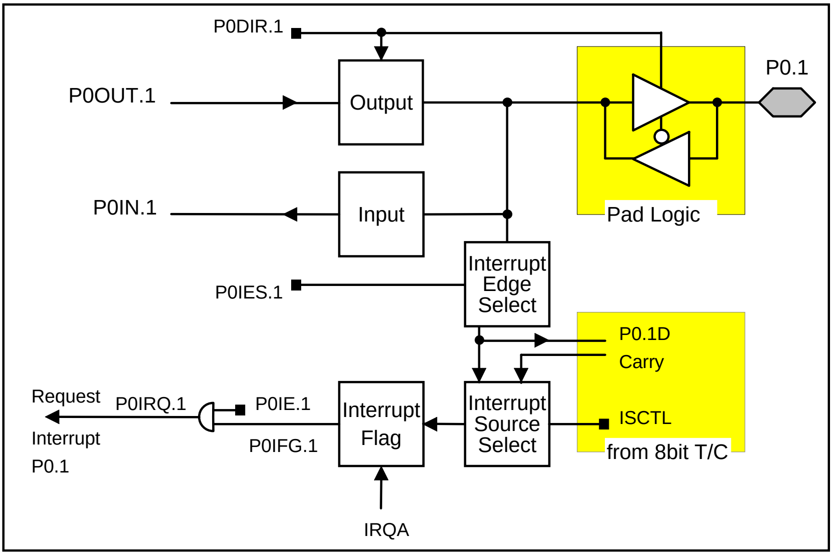


Figure 8.4: Schematic of bit P0.1

The interrupt flag P0IFG.1 is automatically reset when a P0IFG.1 interrupt request was accepted (IRQA).

Port P0, Bit P0.0
The bit zero is identical to bits three to seven, but uses an individual interrupt vector:

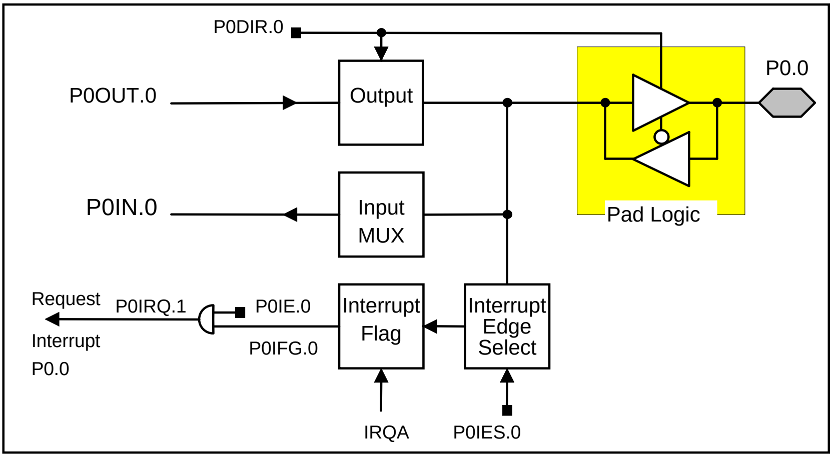


Figure 8.5: Schematic of bit P0.0

The interrupt flag P0IFG.0 is automatically reset when a P0IFG.0 interrupt request was accepted (IRQA).

8.1.3 Port P0 interrupt control functions

Port P0 uses eight bits for interrupt flags, eight bits for interrupt enable, eight bits to select the effective edge of an interrupt event, and three different interrupt vector addresses.
The three interrupt vector addresses are assigned to:

- P0.0
- P0.1/RXD
- P0.2 to P0.7

Two port P0 signals P0.0 and P0.1/RXD are used for dedicated signal processing. Four bits in the SFR address range and two bits in the port0 address frame handle the interrupt events on P0.0 and P0.1/RXD :

- P0.0 Interrupt Flag P0IFG.0 (located in IFG1.2, initial state is reset)
- P0.1/RXD Interrupt Flag P0IFG.1 (located in IFG1.3, initial state is reset)
- P0.0 Interrupt Enable P0IE.0 (located in IE1.2, initial state is reset)
- P0.1/RXD Interrupt Enable P0IE.1 (located in IE1.3, initial state is reset)
- P0.0 Interrupt Edge Select (located in P0IES.0, initial state is reset)
- P0.1/RXD Interrupt Edge Select (located in P0IES.1, initial state is reset)

Both interrupt flags are single source interrupt flags and are automatically reset when the processor system serves them. The enable bits and edge select bits remain unchanged.

The interrupt control bits of the remaining six I/O signals P0.2 to P0.7 are located in the I/O address frame. Each signal uses three bits that define reaction to interrupt events:

- interrupt flag, P0IFG.2 to P0IFG.7
- interrupt enable bit, P0IE.2 to P0IE.7
- interrupt edge select bit, P0IES.2 to P0IES.7

The interrupt flags P0IFG.2 to P0IFG.7 are part of a multiple source interrupt request. Any interrupt event on one or more pins of P0.2 to P0.7 will request an interrupt when two conditions are met: the appropriate individual bit P0IE.x ($2 \leq x \leq 7$) is set and the general interrupt enable bit GIE is set. The six interrupt sources use the same interrupt vector. Since the interrupts share the same interrupt vector, interrupt flags P0.2 to P0.7 are not automatically reset.

The software of the interrupt service routine handles the detection of the source, and also resets the appropriate flag when it is serviced.

Note: Multiple Source interrupt flags P0IFG.2 to P0IFG.7

The interrupt flags P0IFG.2 to P0IFG.7 remain set when an interrupt request has been accepted and serviced. This is mandatory, because it is a multiple source interrupt. Each flag that was served should be reset within its interrupt service routine.

8.2 General Ports P1, P2

The general port P1 and port P2 incorporates all the functions to individually select the function of each pin and to use each signal as an interrupt source.
The seven registers are used to control the Port's I/O pins.
The general module registers are mapped into the lower peripheral file address range where all byte modules are located. The register should be accessed with byte instructions, using absolute address mode.

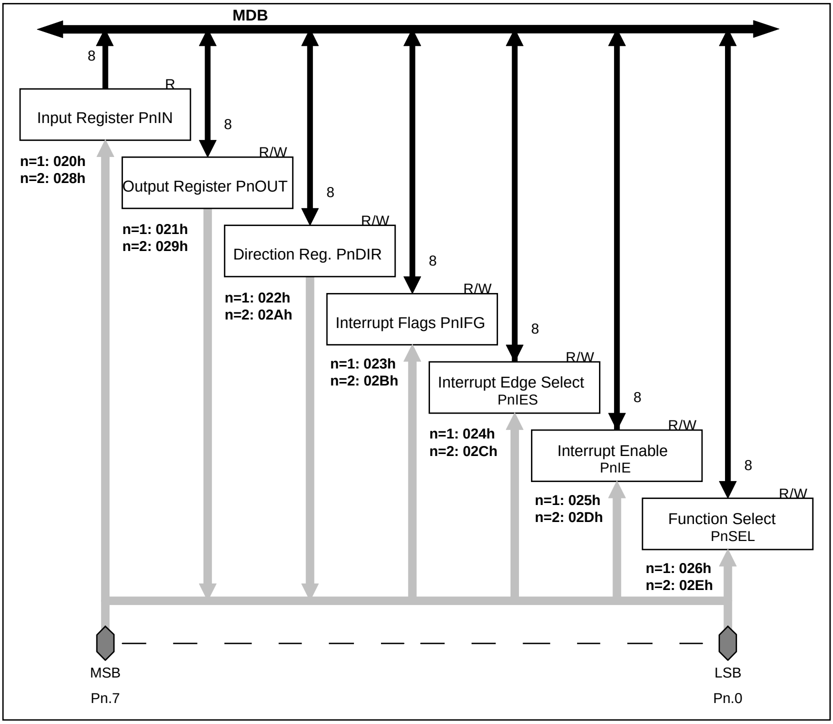


Figure 8.6: Port P1, Port P2 Configuration

8.2.1 Port P1, Port P2 Control Registers

The port P1 and port P2 are connected to the processor core via the 8-bit MDB structure and MAB. They should be accessed via byte instructions.

The seven control registers give maximum flexibility of digital input/output to the application:

- All individual I/O bits are programmable independently:
- Any combination of input, output and interrupt condition is possible.
- Interrupt processing of external events is fully implemented for all eight bits of the port P1 and port P2.

The seven registers for port P1 and the seven registers for port P2 are:

Register	short form	Register type	Address	Initial State
● Input register:	P1IN	read only	020h	-----
● Output register:	P1OUT	read/write	021h	unchanged
● Direction register:	P1DIR	read/write	022h	reset
● Interrupt Flags:	P1IFG	read/write	023h	reset
● Interrupt Edge Select:	P1IES	read/write	024h	unchanged
● Interrupt Enable:	P1IE	read/write	025h	reset
● Function Select reg.:	P1SEL	read/write	026h	reset
● Input register:	P2IN	read only	028h	-----
● Output register:	P2OUT	read/write	029h	unchanged
● Direction register:	P2DIR	read/write	02Ah	reset
● Interrupt Flags:	P2IFG	read/write	02Bh	reset
● Interrupt Edge Select:	P2IES	read/write	02Ch	unchanged
● Interrupt Enable:	P2IE	read/write	02Dh	reset
● Function Select reg.:	P2SEL	read/write	02Eh	reset

All these registers contain eight bits. The registers should be accessed with byte instructions and use absolute address mode.

Input Registers P1IN, P2IN

Both input registers are read-only registers to scan the signals at the I/O pins. The direction of the pin should be selected for input.

> **Note: Writing to read only registers P1IN, P2IN**
>
> Writing to this read-only register results in an increased current consumption as long as the write is active.

Output Registers P1OUT, P2OUT

Each output register shows the information of the output buffer, an eight bit register that contains the information output at the I/O pins if used as outputs. The output buffer can be modified by all instructions that write to a destination. If read, the contents of the output buffer is read independently of the direction. A direction change does not modify the output buffer contents.

Direction Registers P1DIR, P2DIR

Each register contains eight independent bits that define the direction of the I/O pin. All bits are reset by PUC:

Bit = 0: The I/O pin is switched to input direction
Bit = 1: The I/O pin is switched to output direction

Interrupt Flags P1IFG, P2IFG

Each register contains eight flags that contain information if an interrupt is pending or not - according to the corresponding I/O pin:

Bit = 0: No interrupt is pending
Bit = 1: An interrupt is pending due to a transition at the I/O pin. Manipulation on P1OUT and P1DIR as well as P2OUT and P2DIR can also set bits of P1IFG or P2IFG.

Writing a zero to an Interrupt Flag resets it.

Note: Interrupt Flags P1FLG02...7, P2FLG02...7

Each group of the Interrupt Flags P1FLG.0 to P1FLG.7 and P2FLG.0 to P2FLG.7 use only one interrupt vector: both are multiple source interrupt vectors. The interrupt flags P1IFG.0 to P1IFG.7 and P2FLG.0 to P2FLG.7 are not reset automatically when any interrupt from these events is served. The software decides which event will be served and should reset the appropriate flag.

Any external interrupt event should be as long as 1.5 times MCLK or longer, to ensure that it is accepted and the corresponding interrupt flag is set.

Interrupt Edge Select P1IES, P2IES

Each register contains a bit for e corresponding ach I/O pin that selects which transition triggers the interrupt flag. All eight bits according to pin P1.0 to P1.7 and to pin P2.0 to P2.7 are located in these registers. The bits have the following meaning:

Bit = 0: The interrupt flag is set with LO/HI transition
Bit = 1: The interrupt flag is set with HI/LO transition

Note: Change of P1IES, P2IES bit(s)

Changing P1IES, P2IES bit(s) may result in setting the associated interrupt flags:

Bit PnIES.x	PnIN.x	PnIFG.x
0 > 1	0	unchanged
0 > 1	1	may be set
1 > 0	0	may be set
1 > 0	1	unchanged

Interrupt Enable P1IE, P2IE

Each register contains a bit for all eight I/O pins to enable interrupt request on an interrupt event. Each of the eight bits corresponding to pin P1.0 to P1.7 and P2.0 to P2.7 are located in the P1IE and P2IE registers.

The bits have the following meanings:

Bit = 0: The interrupt request is disabled
Bit = 1: The interrupt request is enabled

Note: Port P1, Port P2 interrupt sensitivity

Only transitions, not static levels, cause interrupts.

The interrupt routine must reset all Interrupt Flags, since they follow the multiple interrupt bit scheme of the MSP430 family.

If an Interrupt Flag is still set (because the transition occurred during the interrupt routine) when the *RETI* instruction is executed, an interrupt occurs again after *RETI* is completed. This ensures, that each transition is seen by the software.

Function Select Registers P1SEL, P2SEL

Each register contains eight independent bits that define the functions that access the I/O pin. The port function or a defined module function puts data to the pin, or gets data from the pin. All bits are reset by PUC:

Bit = 0: Port function - output or input data are defined by the port module
Bit = 1: Module function - output or input data are defined by a module not by the port module

Note: Function Select with P1SEL, P2SEL

The interrupt edge select circuitry is disabled if control bit PnSEL.x is set. The input signal will not alter the interrupt flag.
The interrupt edge select and the interrupt flag operates with their full performance when the function select control bit PnSEL is reset.

8.2.2 Port P1, Port P2 Schematic

The pin logic of each individual signal of port P1 and port P2 is identical. Each bit can be read and written.

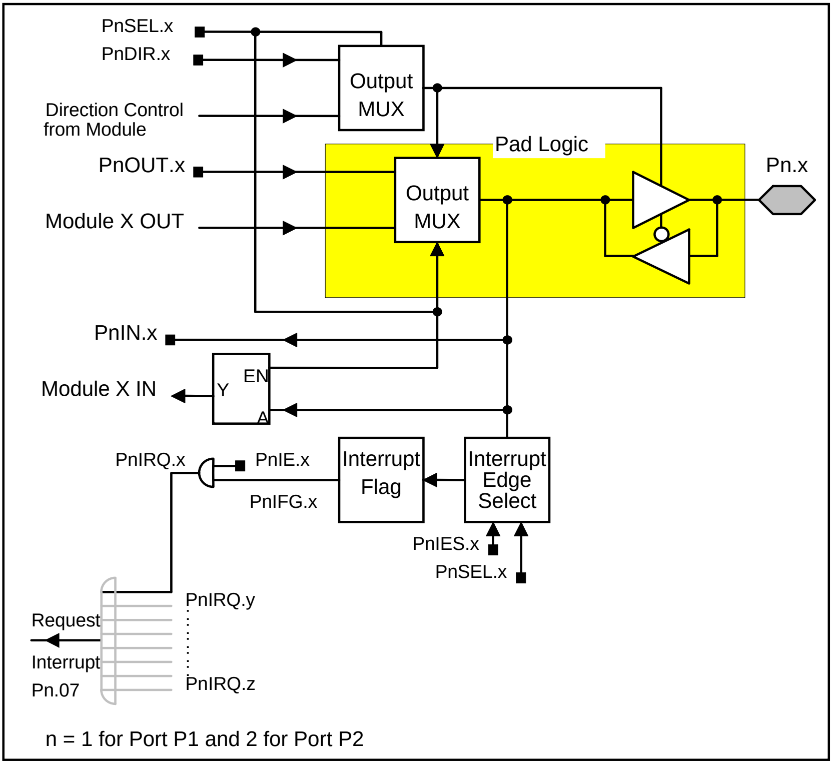


Figure 8.7: Schematic of one bit in Port P1, P2

Module X IN function

The input signal fed to a peripheral module follows the input when the module's function is selected - PnSEL.x = 1. It will be halted continuously at the last level of the input that was passed to the module before the control bit PnSEL.x was reset. Setting the control bit from reset state can alter the signal to the module, when the halted level and the actual level at the input are different.

8.2.3 Port P1, P2 interrupt control functions

Port P1 and port P2 use eight bits for interrupt flags, eight bits for interrupt enable, eight bits to select the effective edge of an interrupt event and one interrupt vector address for port P1 and one interrupt vector address for port P2.

All of the interrupt control bits are located in the I/O address frame. Each signal uses three bits that define reaction to interrupt events:

- interrupt flag, P1IFG.0 to P1IFG.7 and P2IFG.0 to P2IFG.7
- interrupt enable bit, P1IE.0 to P1IE.7 and P2IE.0 to P2IE.7
- interrupt edge select bit, P1IES.0 to P1IES.7 and P2IES.0 to P2IES.7

The interrupt flags P1IFG.0 to P1IFG.7 and P2IFG.0 to P2IFG.7 are part of a multiple source interrupt request. Any interrupt event on one or more pins P1.0 to P1.7 or P2.0 to P2.7 will request an interrupt when two conditions are met, the appropriate individual bit PnIE.x ($0 \leq x \leq 7$) is set, and the general interrupt enable bit GIE is set. The eight interrupt sources use the same interrupt vector. Since the interrupt shares the same interrupt vector, none of the interrupt flags P1.0 to P1.7 or P2.0 to P2.7 is reset automatically.

The software of the interrupt service routine must handle the detection of the source and also resets the appropriate flag when it is serviced.

Note: Multiple Source interrupt flags P1IFG.0 to P1IFG.7, P2IFG.0 to P2IFG.7

The interrupt flags P1IFG.0 to P1IFG.7 and P2IFG.0 to P2IFG.7 remain set when an interrupt request has been accepted and serviced. This is mandatory because it is a multiple source interrupt. Each flag that was served should be reset within its interrupt service routine.

8.3 General Ports P3, P4

The general port P3 and port P4 are identical. They incorporate all the functions to individually select the function of each pin. Each pin can be selected to operate with the port function, or to operate under control of another internal peripheral module.
Four registers control each of the two ports P3 and P4.
The general module registers are mapped into the lower peripheral file address range where all byte modules are located. The register should be accessed with byte instructions, using absolute address mode.

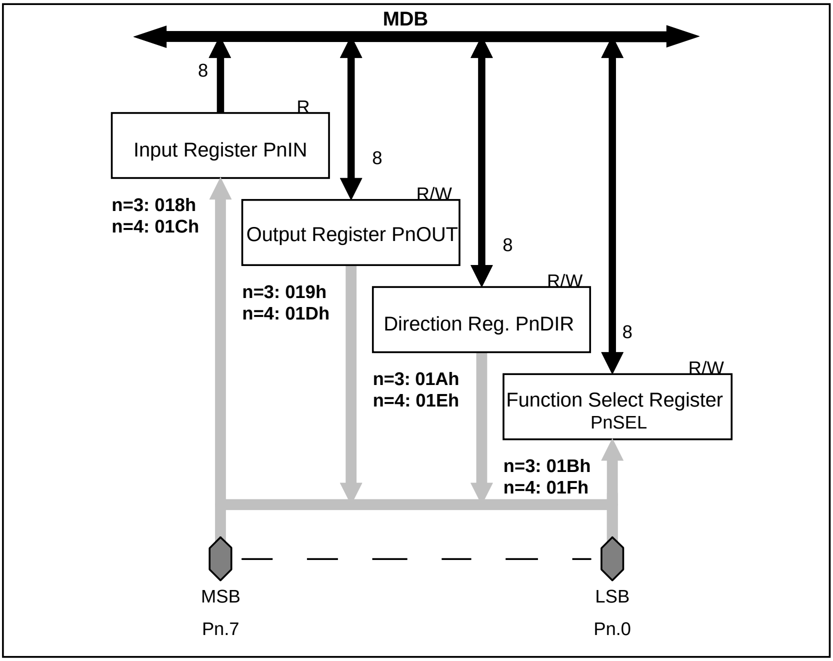


Figure 8.8: Port P3, Port P4 Configuration

8.3.1 Port P3, Port P4 Control Registers

The port P3 and port P4 are connected to the processor core via the 8-bit MDB structure and MAB. They should be accessed via byte instructions using absolute address mode.

The four control registers of each port give maximum flexibility of digital input/output to the application:

- All individual I/O bits are programmable independently:
- Any combination of input is possible.
- Any combination of port or module function is possible.

The four registers for each port are:

Register	short form	Register type	Address	Initial State
• Input register:	P3IN	read only	018h	-----
• Output register:	P3OUT	read/write	019h	unchanged
• Direction register:	P3DIR	read/write	01Ah	reset
• Port Select register:	P3SEL	read/write	01Bh	reset
• Input register:	P4IN	read only	01Ch	-----
• Output register:	P4OUT	read/write	01Dh	unchanged
• Direction register:	P4DIR	read/write	01Eh	reset
• Port Select register:	P4SEL	read/write	01Fh	reset

All these registers contain eight bits and should be accessed with byte instructions.

Input Register P3IN, P4IN

The input registers are read-only registers to scan the signals at the I/O pins. The direction of the pin and the port function should be selected for input

Note: Writing to read only register P3IN, P4IN

Writing to this read only registers results in an increased current consumption, as long as the write is active.

Output Registers P3OUT, P4OUT

The output registers P3OUT and P4OUT show the information of the output buffer, each an eight bit register that contains the output information at the I/O pins if used as outputs. The output buffer can be modified by all instructions that write to a destination. If read, the contents of the output buffer are read independently of the direction. A direction change does not modify the output buffer contents.

Direction Registers P3DIR, P4DIR

Each register contains eight independent bits that define the direction of the I/O pin. All bits are reset by PUC:

Bit = 0: The I/O pin is switched to input direction
Bit = 1: The I/O pin is switched to output direction

Function Select Register P3SEL, P4SEL

Each register contains eight independent bits that define the functions that access the I/O pin. The port function or a defined module function puts data to the pin, or gets data from the pin. All bits are reset by PUC:

Bit = 0:	Port function - output or input data are defined by the port module
Bit = 1:	Module function - output or input data are defined by a module not by the port module

8.3.2 Port P3, Port P4 Schematic

The pin logic of each individual signal of port P3 and port P4 is defined in the specific device configuration. In these device specifications, the function - purely digital port or port function shared with module functions - are defined.

Pins which are used only with digital port function are exclusively controlled by the bits in the corresponding four port registers.

Pins which are used with digital port and module function are controlled by

- the port control bits, when the corresponding select bit PnSEL.x is reset
- the module control bits, when the corresponding select bit PnSEL.x is set

All eight port signal can be configured by the hardware individually to be:

- port pin only
- module function pin only
- software configurable for port or module function

The specific realization is described in the device data sheet.

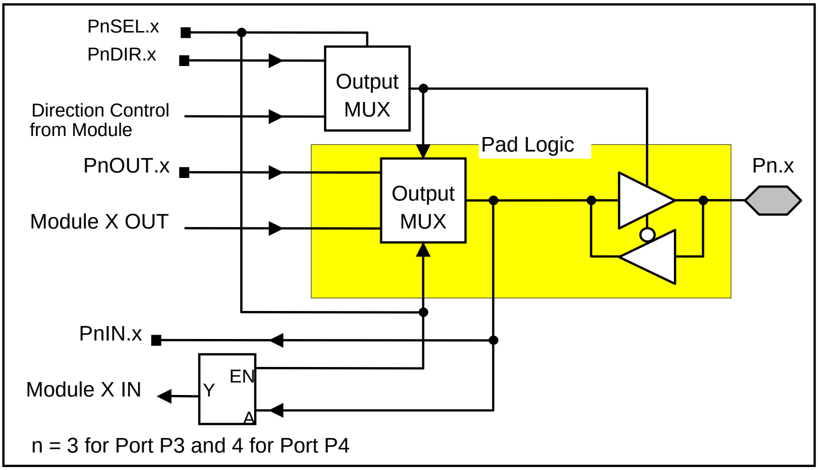


Figure 8.9: Schematic of bits P3.x/P4.x

Module XIN function

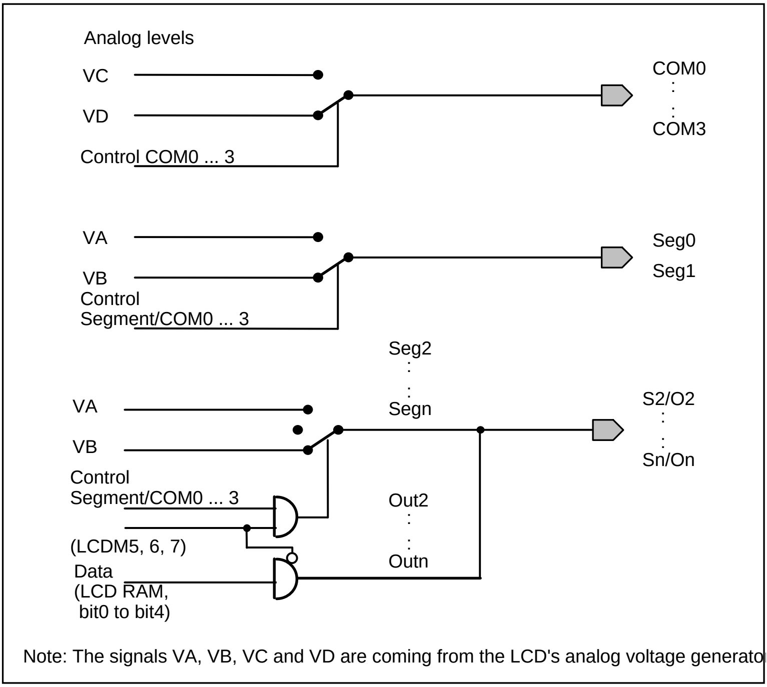
The input signal fed to a peripheral module follows the input when the module’s function is selected - PnSEL.x = 1. It will be halted continuously at the last level of the input that was passed to the module before the control bit PnSEL.x was reset. Setting the control bit from reset state can alter the signal to the module when halted level and actual level at the input are different.

8.4 LCD Ports

The LCD ports can be selected either to drive a liquid crystal display, or to act as digital outputs driving static output signals. The control of a liquid crystal display uses common and segment output stages to drive the analog signals needed for multiplex rates of 2Mux and higher.

LCD outputs

The LCD outputs use transmission gates to transfer the analog voltage to the output pin, when they are used to drive liquid crystal displays. Groups of LCD outputs can be configured by software to operate as digital outputs.



Note: The signals VA, VB, VC and VD are coming from the LCD's analog voltage generato

Figure 8.10: Schematic of LCD

Three bits in the LCD control register LCDM5, LCDM6 and LCDM7 control the function of these groups of signals. For more information on control of these outputs, see LCD description.

8.5 LCD Port - Timer/Port Comparator

The comparator associated with the Timer/Port module is shared typically with one segment line. The segment line function is selected for this pin after PUC signal was active. The comparator input is selected when the CPON bit - located in the Timer/Port module - is set the first time. It remains set as long as it is not reset by PUC.

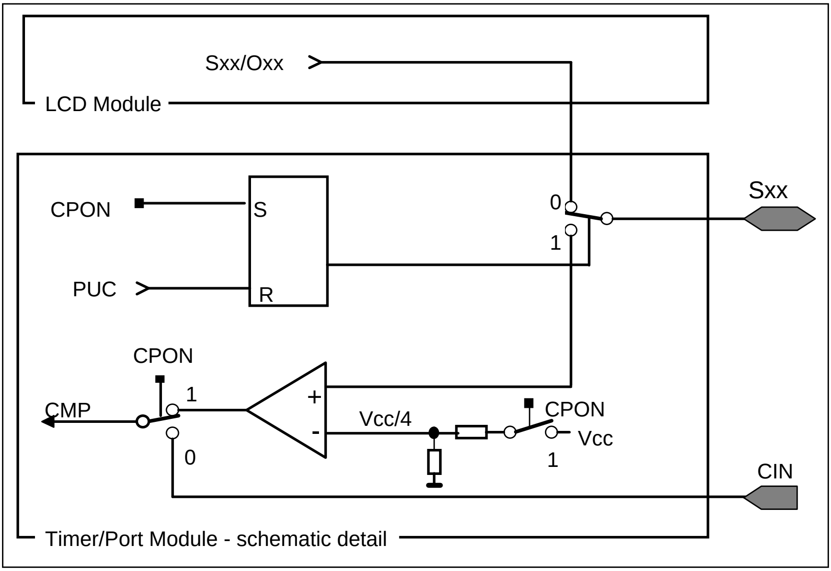


Figure 8.11: Schematic of LCD pin - Timer/Port Comparator

9 Universal Timer/Port Module

Topic **Page**

The universal Timer/Port Module supports several major system functions:

- Up to six independent outputs
- Two 8-bit counters, cascadeable for 16-bit mode
- Precision comparator for A/D conversion of slope converter type

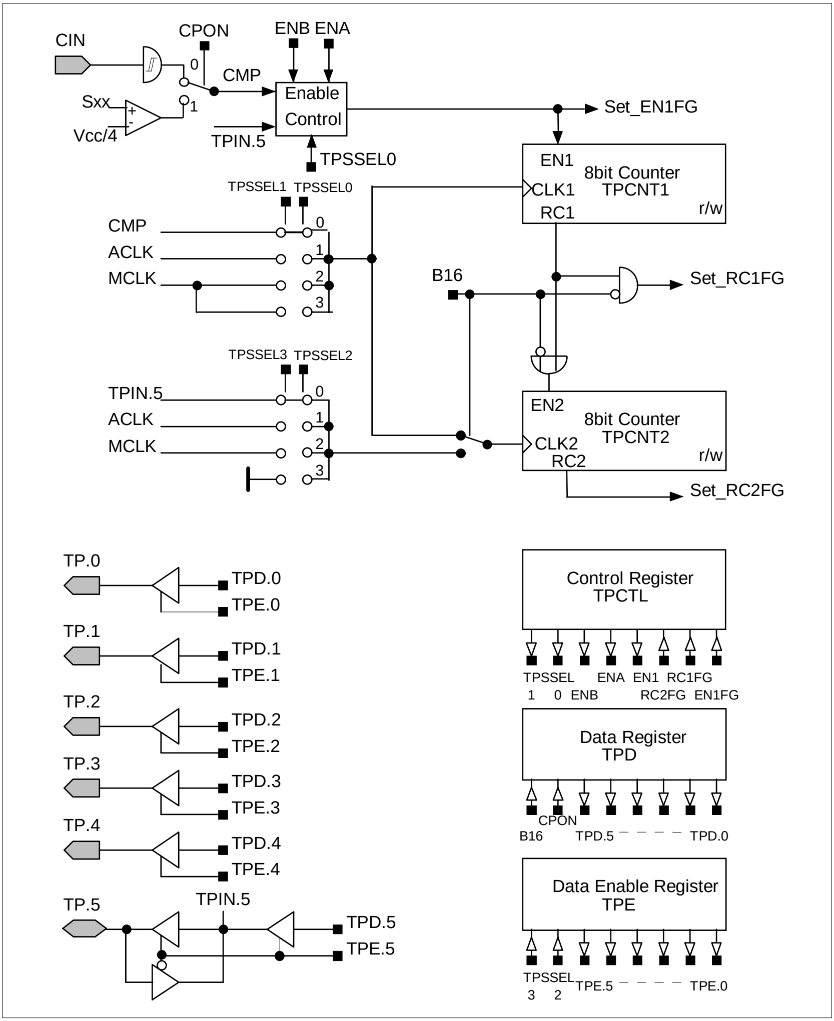


Figure 9.1: Timer/Port configuration

9.1 Timer/Port Module Operation

The Timer/Port Module can be configured through the bits in the control register TPCNTL to operate in different ways.

9.1.1 Timer/Port Counter TPCNT1, 8-bit Operation

The counter TPCNT1 can be read and written with appropriate instructions. The read access to the timer's data can be asynchronous to the clock source when CMP or ACLK is selected. A majority vote from 2 of 3 samples taken by software will ensure that the data read are correct.

When the clock source is MCLK the data read are correct. Since MCLK is the same frequency as the instructions which are clocked, the data read is only a sample of the status of the counter and the counter is incremented continuously while EN1 is set.

The counter can be written at any time. After the write cycle is performed a re-read of data from the counter can be different, if clocks are applied between the write and read access.

Three different clock sources can count-up the counter: MCLK, ACLK or CMP.

The counter is incremented with each positive edge at the clock input when the enable input EN1 of the counter is set. The counter is enabled when one or both signals ENA and ENB are set. With system reset, both enable bits are reset and the counter function is disabled. Resetting both enable bits will freeze the present counter data.

The ripple carry signal RC1 is high, as long as the counter data is 0FFh. The negative edge of the ripple carry signal RC1 sets the RC1FG bit in the register TPCTL.

The flag RC1FG is set when the counter TPCNT1 rolls from 0FFh to 0. The flag EN1FG is set when EN1 is switched to disable, but not when ENA or ENB is the source of disable. An interrupt service is requested when the enable bit TPIE is set and one of the flags RC1FG, RC2FG or EN1FG is set. The flags RC1FG, RC2FG and EN1FG should be reset by software.

9.1.2 Timer/Port Counter TPCNT2, 8-bit operation

The counter's TPCNT2 operation is different from the counter's TPCNT1 operation by the source of the enable signal and clock signals. The counter is always enabled with 8-bit operation selected. Three different clock sources can count-up the counter: MCLK, ACLK or TPIN.5 .

The ripple carry signal RC2 is high as long as the counter data is 0FFh. The negative edge of the ripple carry signal RC2 sets the RC2FG bit in the register TPCTL.

The interrupt flag RC2FG is set when the counter TPCNT2 rolls from 0FFh to 0.

9.1.3 Timer/Port Counter , 16-bit operation

The 8-bit counter TPCNT1 and the 8-bit counter TPCNT2 can be cascaded to form a 16-bit counter. The bit B16 in the control register is set for this operation.

Any read or write access to the counters remains a byte access. The data of the counter TPCNT1 and TPCNT2 are read or written sequentially. This needs special considerations if the access is done during counter operation.

The enable signal of the counter TPCNT1 is the enable of the 16-bit counter and the clock source of TPCNT2 is the same as for the counter TPCNT1; it is selected only by TPSSEL0 and TPSSEL1. The source select signals TPSSEL2 and TPSEL3 are “don't care”.

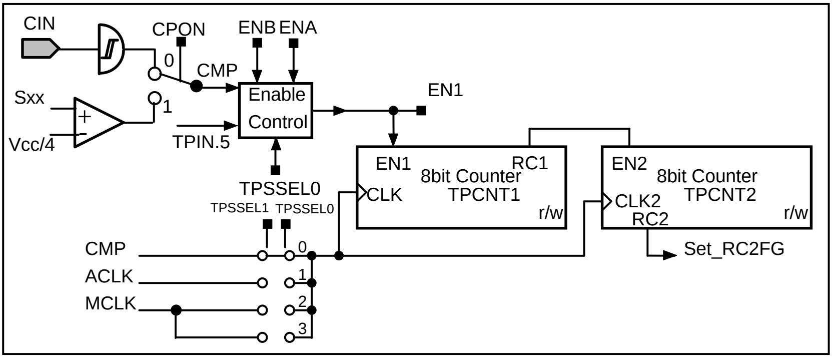


Figure 9.2: Timer/Port counter, 16-bit operation

The four signals ENA, ENB, TPSSEL0 and TPSSEL1 control the operation of the cascaded counters, the count enable and the counter's clock source:

ENB	ENA	TPSSel1	TPSSel0	EN1	CLK1
0	0	0	0	0	CMP
0	0	0	1	0	ACLK
0	0	1	X	0	MCLK
0	1	0	0	1	CMP
0	1	0	1	1	ACLK
0	1	1	X	1	MCLK
1	0	0	0	$\overline{\text{TPIN.5}}$	CMP
1	0	0	1	TPIN.5	ACLK
1	0	1	0	$\overline{\text{TPIN.5}}$	MCLK
1	0	1	1	TPIN.5	MCLK
1	1	0	0	$\overline{\text{CMP}}$	CMP
1	1	0	1	CMP	ACLK
1	1	1	0	$\overline{\text{CMP}}$	MCLK
1	1	1	1	CMP	MCLK

The 16-bit counter can therefore be halted or counted-up unconditionally, with the signal applied to pin CIN, Sxx or with one of the three clocks: ACLK, MCLK or CMP.

The application of TPIN.5, TPIN.5 inverted, CMP or CMP inverted signal to the counter enable input EN1 the 16-bit counter will be incremented with each ACLK or MCLK. This feature is used to measure the time period of signals applied to pin CMP or TPIN.5.

The ripple carry signal RC2 is set as long as the counter data is 0FFFFh.

The flag RC2FG is set when the counter rolls from 0FFFFh to '0'. The flag EN1FG is set when the EN1 is switched to disable. The source of enable EN1 is TPIN.5 or CMP. It is not set when EN1 is switched to disable via software using ENA and ENB.

9.2 Timer/Port Registers

The Timer/Port module hardware is byte structured and should be accessed by byte processing instructions (suffix 'B').

Register	short form	Register type	Address	Initial state
• TP control register:	TPCTL	Type of read/write	04Bh	Reset
• TP Counter 1:	TPCNT1	Type of read/write	04Ch	unchanged
• TP Counter 2:	TPCNT2	Type of read/write	04Dh	unchanged
• TP O/P Data register:	TPD	Type of read/write	04Eh	Reset
• TP Data enable register:	TPE	Type of read/write	04Fh	Reset

Timer/Port Control register
The information stored in the control register determines the operation of the Timer/Port module.

TPCTL
04Bh

7							0
TPSSEL 1	TPSSEL 0	ENB	ENA	EN1	RC2FG	RC1FG	EN1FG
rw-0	rw-0	rw-0	rw-0	r-0	rw-0	rw-0	rw-0

Figure 9.3: Timer/Port Control Register

Bit 0: The enable flag EN1FG is set with the negative edge of enable signal EN1 of counter TPCNT1, if the enable signal came from CMP or TPIN.5 pin. This event sets the EN1FG bit and the software should reset it. Otherwise, it will remain set.
The EN1FG bit can be used during the Timer/Port interrupt service routine to decide if the interrupt event was from enable EN1 or from a ripple/carry RC that is set when a counter rolls from 0FFh to 0h.

Bit 1: The bit RC1FG indicates that the counter TPCNT1 has rolled from 0FFh to 0h (overflow condition). This event sets the RC1FG bit, and the software should reset it. Otherwise, it will remain set.
It is used in the Timer/Port Interrupt Service routine to identify the source of an interrupt event.

Bit 2: The bit RC2FG indicates that the counter TPCNT2 has rolled from 0ffh to 0h (overflow condition). This event sets the RC2FG bit, and the software should reset it. Otherwise, it will remain set.
It is used in the Timer/Port Interrupt service routine to identify the source of an interrupt event.

Bit 3, 4, 5: The enable signal EN1 of the counter TPCNT1 can be read. The level or the signal of the bit EN1 is defined with control signals ENA, ENB, TPSSEL0.

ENB	ENA	TPSSel0	EN1
0	0	X	0
0	1	X	1
1	0	0	$\overline{\text{TPIN.5}}$
1	0	1	TPIN.5
1	1	0	$\overline{\text{CMP}}$
1	1	1	CMP

Bit 7, 6: The Timer/Port clock source select bits TPSSEL0 and TPSSEL1 control the multiplexer to supply 1 of 3 clock sources to the counter TPCNT1.

TPSSel1	TPSSel0	CLK1
0	0	CMP
0	1	ACLK
1	X	MCLK

Timer/Port counter TPCNT1 and TPCNT2

Both counters are 8-bit, and any read or write access should be done with byte instructions.

	7							0
TPCNT1 04Ch	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	rw	rw	rw	rw	rw	rw	rw	rw

	7							0
TPCNT2 04Dh	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	rw	rw	rw	rw	rw	rw	rw	rw

Figure 9.4: Timer/Port Counter Registers

Both counters can be read and written independently. Any reset of a counter is done using the CLEAR instruction.

Timer/Port Data Register

The Data Register holds the value of six outputs and two control bits of the comparator.

	7							0
TPD 04Eh	B16	CPON	TPD.5	TPD.4	TPD.3	TPD.2	TPD.1	TPD.0
	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Figure 9.5: Timer/Port Data Register

Bit 0 ... 5: The bits TPD.0 to TPD.5 hold the data for the output pins TP.0 to TP.5. The digital signals will be applied to these pins when the 3-state output is enabled by TPE.0 to TPE.5 . They are reset whenever a system reset PUC happens .
The signal at TP.5 is used internally in the module and can be read via the enable bit EN1 located in the control register TPCTL.

Bit 6: The comparator CPON bit switches on the supply of the comparator. It is used to save current during its reset state. Whenever system reset PUC becomes active, the comparator on bit CPON is reset and the comparator is inactive.

Bit 7: The control bit B16 selects the operation of the two counters TPCNT1 and TPCNT2. They can operate as two independent 8-bit counters or as one 16-bit counter. The access is always in byte mode. In the 16-bit mode, any read or write access is done separately to counter TPCNT1 and counter TPCNT2.

B16 = 0: Two 8-bit counter mode selected.

B16 = 1: One 16-bit counter with TPCNT1 for low byte and TPCNT2 for high byte. The counter TPCNT2 increments its data when the counter TPCNT1 rolls from 0FFh to 0h.

Timer/Port Enable Register

The Timer/Port Enable Register holds the control value of six outputs and two bits indicating counter overflow.

	7							0
TPE 04Fh	TPSSEL 3	TPSSEL 2	TPE.5	TPE.4	TPE.3	TPE.2	TPE.1	TPE.0
	rw-1	rw-1	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Figure 9.6: Timer/Port Enable Register

Bits 0 ... 5: The bits TPE.0 to TPE.5 hold the enable control (3-state) signals of the outputs TP.0 to TP.5. They are reset whenever a system reset PUC happens and the outputs are high-impedance.

Bit 6, 7: The Timer/Port clock source select bits TPSSEL2 and TPSSEL3 control the multiplexer to supply 1 of 4 clock sources to the counter TPCNT2. The control bit B16 should be reset. When the control bit B16 is set, TPSSEL2/3 are “don't care” and the clock source of counter TPCNT2 is the same as of the counter TPCNT1.

B16	TPSSel3	TPSSel2	CLK2
0	0	0	TPIN.5
0	0	1	ACLK
0	1	0	MCLK
0	1	1	‘0’ or VSS
1	X	X	≡ CLK1

9.3 Timer/Port Special Function bits

The Timer/Port module covers one interrupt vector, multiple source interrupt flags (not located in the SFR) and one interrupt enable bit.

The Timer/Port interrupt enable TPIE is located in IE.2 register. Initial state is reset.

The multiple source interrupt flags RC1FG, RC2FG and EN1FG are located in the register TPCTL. Their initial state is reset.

The interrupt flags RC1FG, RC2FG and EN1FG are not reset automatically by hardware when the interrupt request is served. The flag EN1FG is set with the negative edge of the counter enable signal EN1 and indicates that the counter was halted. It is not set if software toggles from enable to disable of TPCNT1 by using ENA and ENB control signals.

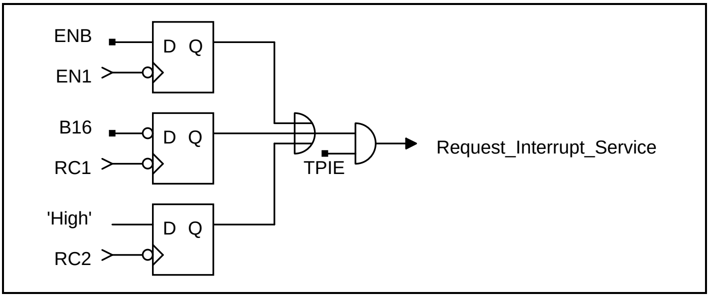


Figure 9.7: Timer/Port Interrupt Scheme

When a Timer/Port interrupt is asserted, the interrupt service routine should consider the different interrupt sources and decide how to proceed. When the 8-bit counter mode is selected, three interrupt sources can request an interrupt service: the negative edge of EN1 signal, an overflow from counter TPCNT1 (RC1 signal), or an overflow from counter TPCNT2 (RC2 signal). In the 16-bit counter mode (B16 is set) two sources can request an interrupt service: the negative edge of EN1 signal, and an overflow from counter TPCNT2.

B16	RC2		RC1		EN1
0	↓	OR	↓	OR	↓
1	↓	OR	X	OR	↓

Figure 9.8: Conditions for Timer/Port Interrupt request

The interrupt request bits should be reset during the interrupt service routine. If this is not done, another interrupt is requested immediately after the GIE is set or the RETI instruction is executed. The Timer/Port interrupt enable bit TPIE is reset with PUC. When the TPIE bit is reset, no interrupt request is done. When the TPIE bit and the general interrupt enable bit GIE are set, the System/CPU will serve the interrupt if no PUC or NMI is active.

9.4 Timer/Port in ADC Application

The Timer/Port peripheral incorporates all functions to support an A/D converter function for resistive or capacitive sensors.

For temperature measurement the most popular sensor elements are resistors that have positive or negative temperature coefficients. Silicon sensors, NTC resistors and platinum sensors are such sensor types.

9.4.1 Principle of conversion, R/D

The conversion of a resistor value to a digital representation is done by measuring the time that is needed to discharge a capacitor. This capacitor is charged-up before the discharge phase.

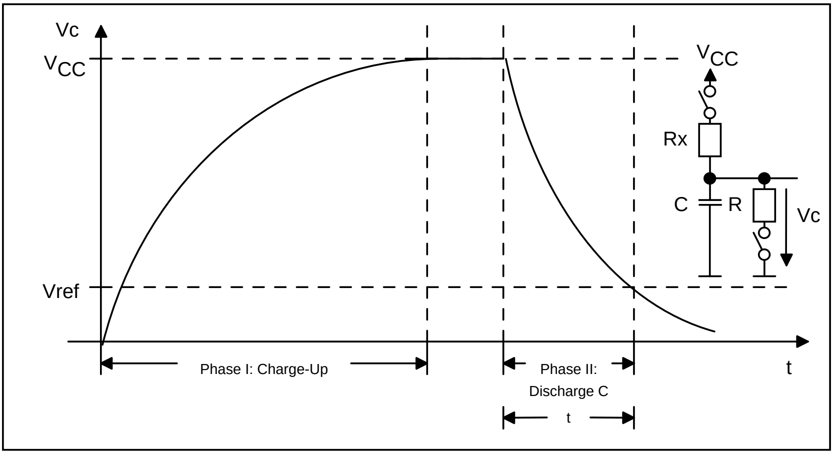


Figure 9.9: Charge-Discharge timing of RC

During the discharge of the capacitor down to the reference voltage Vref, the time of this period is measured using a voltage comparator and a counter. The counter is incremented continuously as long as the voltage level at the capacitor C is above the reference level Vref. The increment of the counter is possible because the comparator output is used to enable the counter's operation.

The formula for this time measurement principle is

$$t = -R * C * \ln\frac{Vref}{Vcc}$$

$$t = N * t_{Clock}$$

$$N * t_{Clock} = -R * C * \ln\frac{Vref}{Vcc}$$

$$N = -R\ C\ f_{Clock} * \ln\frac{Vref}{Vcc}$$

The value of C, f_{Clock} and Vref/$_{VCC}$ should be known to determine the value of resistor R. Using a second conversion with a well defined and stable reference resistor, the sensor to be measured can be determined by:

$$\frac{N_{meas}}{N_{ref}} = \frac{-R_{meas} * C * \ln\frac{Vref}{Vcc}}{-R_{ref} * C * \ln\frac{Vref}{Vcc}}$$

$$\frac{N_{meas}}{N_{ref}} = \frac{R_{meas}}{R_{ref}}$$

$$R_{meas} = R_{ref} * \frac{N_{meas}}{N_{ref}}$$

This assumes that the circuit uses the same capacitor, and that both voltages and the clock period are constant during both conversions.

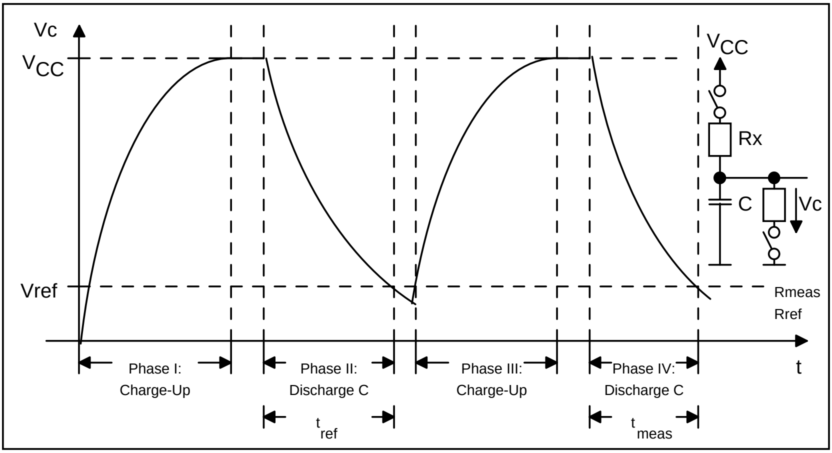


Figure 9.10: Charge-Discharge timing during R/D conversions using Rref and Rmeas

The capacitior C is charged through any resistor, Rx, up to V_{CC} during Phase I and Phase III. It is discharged via Rref or Rmeas. The current is then limited by the resistor(s). If only the resistor Rref used, the time for charging the capacitor is well defined.

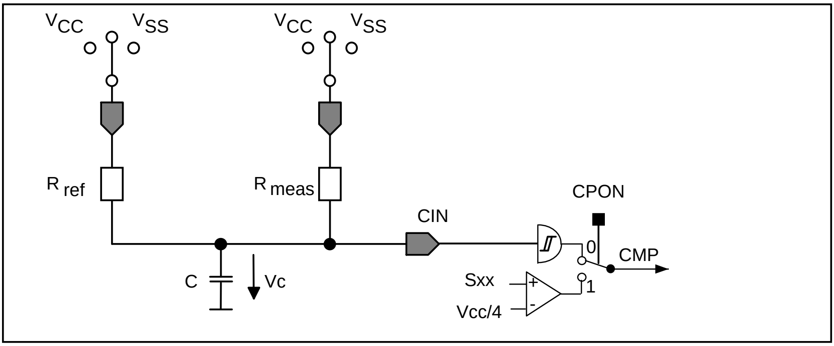


Figure 9.11: Principle Conversion Scheme

The capacitor C is discharged while one of the resistors is connected to Vss. All parasitics of the discharge current path will influence the time to discharge the capacitor C down to Vref.

9.4.2 Conversion with Resolution of >8 bit

The conversion is demonstrated using the comparator, the counters in 16-bit mode and the digital outputs.

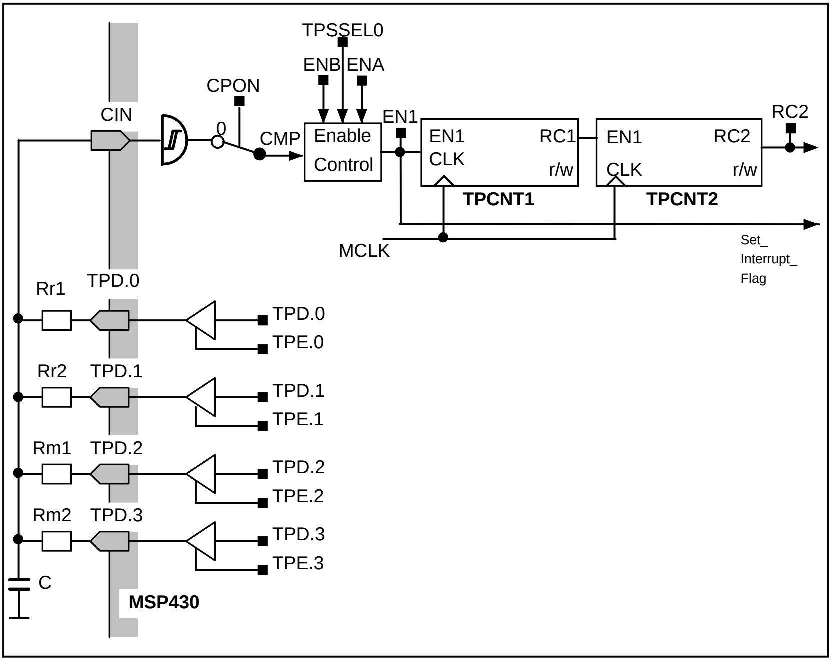


Figure 9.12: ADC Application example

The external components in this application are two reference resistors, Rr1 and Rr2, and two sensor resistors to be measured, Rm1 and Rm2.

The internal configuration is selected via the control register TPCTL to meet the circuit function of the application example. The schematic is reduced to the active connections and block. The control register TPCTL is loaded with 9Eh which means that bits B16, TPSSEL1, TPSSEL0, ENB and ENA are set.

The capacitor C is charged through Rr1 and/or Rr2 up to V_{CC}.

The time of the four discharge phases is measured:

$$t_{r1} = N_{r1} * t_{MCLK} = - R_{r1} * C * \ln \frac{V_{ref}}{V_{CC}}$$

$$t_{r2} = N_{r2} * t_{MCLK} = - R_{r2} * C * \ln \frac{V_{ref}}{V_{CC}}$$

$$t_{m1} = N_{m1} * t_{MCLK} = - R_{m1} * C * \ln \frac{V_{ref}}{V_{CC}}$$

$$t_{m2} = N_{m2} * t_{MCLK} = - R_{m2} * C * \ln \frac{V_{ref}}{V_{CC}}$$

and the following formula is used to determine the resistors Rm1 and Rm2:

$$\frac{R_{r1} - R_{r2}}{R_{mx} - R_{r2}} = \frac{N_{r1} - N_{r2}}{N_{mx} - N_{r2}}$$

$$R_{mx} = R_{r2} + (R_{r1} - R_{r2}) * \frac{N_{mx} - N_{r2}}{N_{r1} - N_{r2}}$$

$$R_{m1} = R_{r2} + (R_{r1} - R_{r2}) * \frac{N_{m1} - N_{r2}}{N_{r1} - N_{r2}}$$

$$R_{m2} = R_{r2} + (R_{r1} - R_{r2}) * \frac{N_{m2} - N_{r2}}{N_{r1} - N_{r2}}$$

10 Timers

Topic **Page**

10.1 Basic Timer1

The intention of the basic timer operation is to support the software and various peripheral modules with a low power consumption, low frequency reference.

The following are examples of software functions controlled by the stability of the crystal:

- real time clock RTC
- debouncing keys, keyboard
- software time incremental feature**

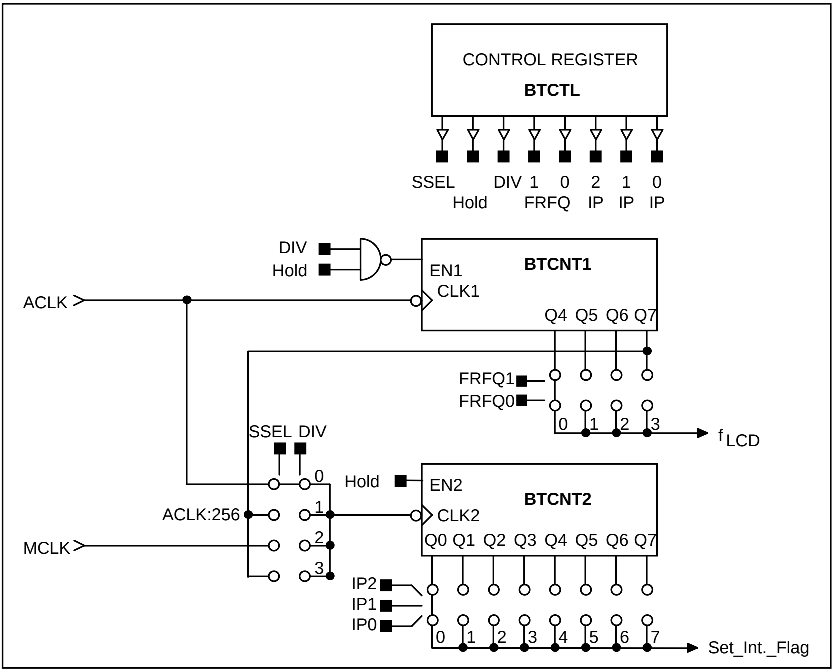


Figure 10.1: Basic Timer Configuration

The Basic Timer1 supplies other peripheral modules with low frequency control signals. The software can access both 8-bit counters.

The control register BTCTL holds the flags to control or select the different operational functions. The register BTCTL, the 8-bit counter BTCNT1 and the 8-bit counter BTCNT2 are under full control of the software. When supply voltage is applied, a reset of the device or a watchdog overflow or any other operational condition occurs, and all bits in the register hold an undefined or unchanged status. The user's software usually configures the operational conditions of the Basic Timer1 during initialization.

10.1.1 Basic Timer1 Register

The Basic Timer1 module hardware is byte structured, and should be accessed by byte processing instructions (suffix .B).

Register	short form	Register type	Address	Initial state
● BT1 control register	BTCTL	Type of read/write	040h	unchanged
● BT counter 1	BTCNT1	Type of read/write	046h	unchanged
● BT counter 2	BTCNT2	Type of read/write	047h	unchanged

Basic Timer1 Control Register

The information stored in the control register determines the operation of the basic timer. The status of the different bits selects the frequency source, the interrupt frequency and the framing frequency of the LCD control circuitry.

	7							0
BTCTL 040h	SSEL	Hold	DIV	FRFQ1	FRFQ0	IP2	IP1	IP0
	rw	rw	rw	rw	rw	rw	rw	rw

Figure 10.2: Basic Timer1 Register

Bit 0 ... 2: The three least significant bits IP2..0 determine the interrupt interval time. It is the interval of consecutive settings of interrupt request flag BTIFG.

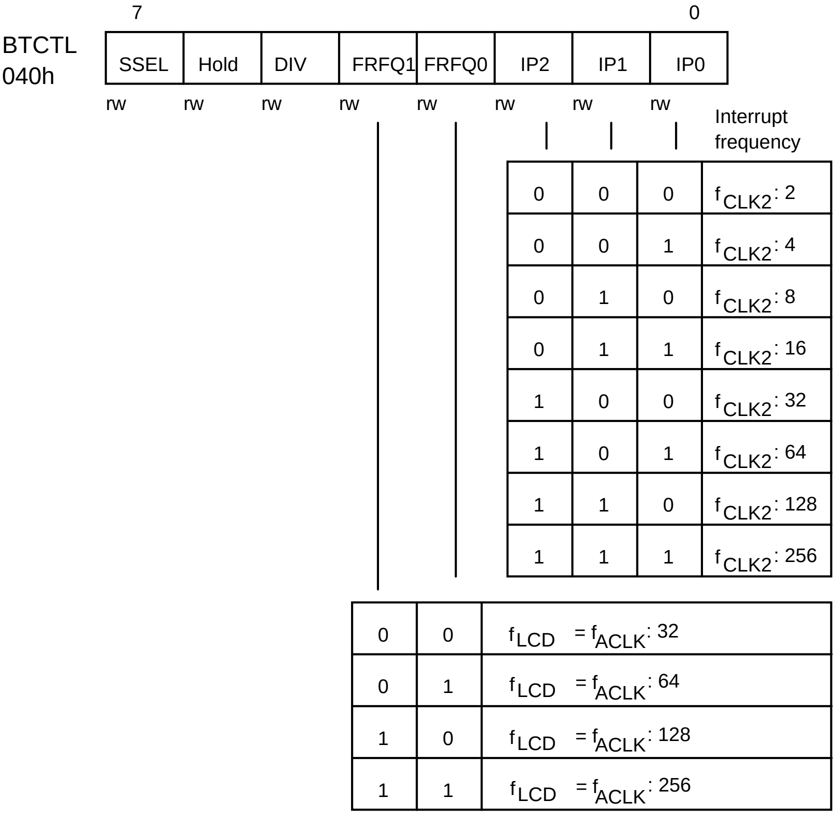
Bit 3 ... 4: The two bits FRFQ1 and FRFQ0 select the frequency f_{LCD}. Devices with LCD peripheral on-chip use this frequency to generate the timing of the common and select lines.

Bit 5: see Bit 7.

Bit 6: The Hold bit stops the counters operation.
The BTCNT2 is held, if Hold bit is set.
The BTCNT1 is held, if Hold bit and DIV bit are set.

Bit 7: The SSEL bit and DIV bit select the input frequency source of BTCNT2.

SSEL	DIV	CLK2
0	0	ACLK
0	1	ACLK/256
1	0	MCLK
1	1	ACLK/256



BTCTL 040h

7							0
SSEL	Hold	DIV	FRFQ1	FRFQ0	IP2	IP1	IP0
rw	rw	rw	rw	rw	rw	rw	rw

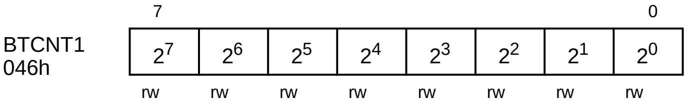
IP2	IP1	IP0	Interrupt frequency
0	0	0	f_{CLK2}: 2
0	0	1	f_{CLK2}: 4
0	1	0	f_{CLK2}: 8
0	1	1	f_{CLK2}: 16
1	0	0	f_{CLK2}: 32
1	0	1	f_{CLK2}: 64
1	1	0	f_{CLK2}: 128
1	1	1	f_{CLK2}: 256

FRFQ1	FRFQ0	
0	0	$f_{LCD} = f_{ACLK}$: 32
0	1	$f_{LCD} = f_{ACLK}$: 64
1	0	$f_{LCD} = f_{ACLK}$: 128
1	1	$f_{LCD} = f_{ACLK}$: 256

Figure 10.3: Basic Timer1 Register Function

Basic Timer1 Counter BTCNT1

The Basic Timer Counter BTCNT1 divides the auxiliary clock ACLK. The frame frequency for LCD-Drive is selected from four outputs of the counter FFs. The output of the most significant FF can be used for the clock input to the second counter BTCNT2. The output of the counter Q0...7 can be read and the counter Q0..7 can be written by software.



BTCNT1 046h

7							0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
rw	rw	rw	rw	rw	rw	rw	rw

Basic Timer1, Counter BTCNT2

The Basic Timer Counter BTCNT2 divides the input clock frequency. The input clock source can be selected to be MCLK, ACLK or ACLK:256 signal. The interrupt period can be selected using IP0...2 located in the basic timer control register BTCTL, and selects one of the eight FF outputs.

	7							0
BTCNT2 047h	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	rw	rw	rw	rw	rw	rw	rw	

The output of the counter Q0...7 can be read, and the counter (Q0...7) can be written, by software.

10.1.2 Special function register bits

Bits in the SFR address range handle the system control interaction, according to the function implemented in the basic timer:

- Basic Timer Interrupt Flag BTIFG (located in IFG2.7)
- Basic Timer Interrupt Enable BTIE (located in IE2.7).

The Hold bit inhibits all functions of the module and reduces the power consumption to its minimum - the leakage current.

No additional counts occur when the counter is enabled or disabled. The access of the system to the general module register BTCTL is not affected. It can be read or written in the usual manner.

The interrupt flag and interrupt enable follow the general rules of module interrupts. Beside the individual interrupt enable bit, the interrupt request is also controlled by the general interrupt enable bit GIE. The interrupt enable flag BTIE is reset on PUC. The interrupt flag BTIFG is reset when an interrupt request of the basic timer is accepted.

10.1.3 Basic Timer1 Operation

The basic timer is constantly incremented by the clock ACLK or MCLK. The SSEL control signal selects either the auxiliary clock ACLK or the main clock MCLK (system clock f_{system}) for Counter BTCNT2.

An interrupt can be used to control system operation. The interrupt is a single source interrupt.

The Basic Timer can operate in two different modes:

- Two independent eight-bit timer/counters
- One sixteen-bit timer/counter

Mode, 8-bit counters

In the 8-bit mode the basic timer BTCNT1 is incremented constantly with ACLK. When the counter is read the asynchronous behavior of the counter (ACLK) and the system

(MCLK) should be considered. The counter can be written to using software asynchronous to the counter's clock.

The BTCNT2 clock signal can be selected for MCLK, ACLK or ACLK/256 with the control signals SSEL and DIV. The counter BTCNT2 is incremented with the signal selected.

One of the eight counter outputs can be selected to set the basic timer interrupt flag. Read and write access can be asynchronous when ACLK or ACLK/256 is selected.

Mode, 16-bit counter

The sixteen-bit timer/counter mode is selected with the control bit DIV set. In this mode, the clock source of counter BTCNT1/BTCNT2 is ACLK signal.
The Hold bit stops operation of both eight-bit counters.

10.1.4 Basic Timer1 Operation: Signal f_{LCD}

The peripheral LCD module uses the signal f_{LCD} to generate the timing for common and segment lines. The frequency of the signal f_{LCD} is generated from ACLK. Using a 32,768 Hz crystal in the oscillator, the frequency at f_{LCD} is 1024 Hz, 512 Hz, 256 Hz or 128 Hz. The bits FRFQ1 and FRFQ0 allow the correct choice of the frame frequency. The proper frequency f_{LCD} depends on the LCD's characteristic data for the framing frequency and the multiplex rate of the LCD.

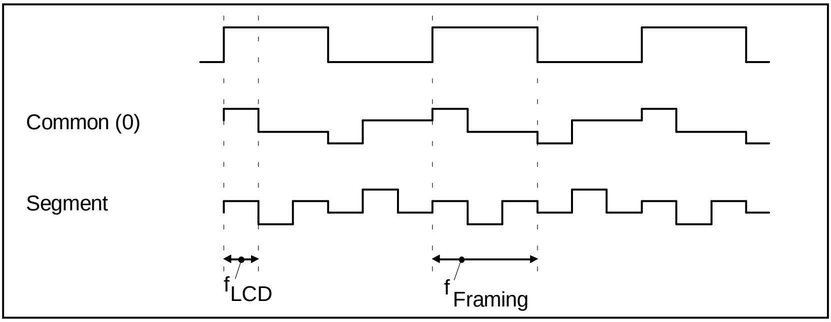


Figure 10.4: Frequency Select for LCD (Example for 3MUX)

Example for 3MUX:

LCD data sheet: $f_{Framing}$ = 100 Hz .... 30 Hz

FRFQ: $f_{LCD} = 6 \times f_{Framing}$

f_{LCD} = 6 x 100 Hz ... 6 x 30 Hz = 600 Hz ... 180 Hz

Select f_{LCD} : 1024 Hz or 512 Hz or 256 Hz or 128 Hz

f_{LCD} = 256 Hz → FRQ1 = 1; FRFQ0 = 0

10.2 8-bit Interval Timer/Counter

The 8-bit interval timer supports three major functions for the application:

- serial communication or data exchange
- pulse counting or pulse accumulation
- timer

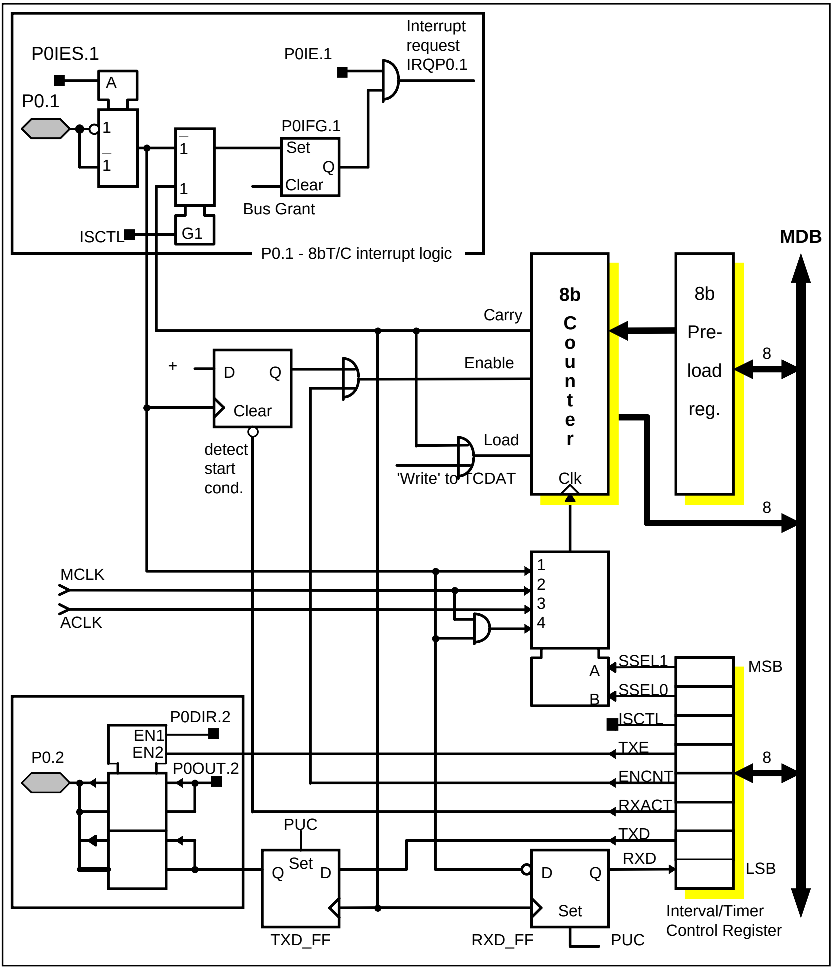


Figure 10.5: Principle Schematic of 8-bit Timer/Counter

10

10.2.1 Operation of 8-bit Timer/Counter

The 8-bit Timer/Counter includes the following major blocks:

- 8-bit Up-Counter with pre-load register
- 8-bit Control Register
- Input clock selector
- Edge detection, e.g. Start bit at asynchronous protocols
- Input and output data latch, triggered by carry-out-signal from 8-bit counter.

8-bit Up-Counter with pre-load register

The 8-bit counter counts up with the input clock selected via two control bits (SEL0, SEL1) of the control register. Two inputs (Load, Enable) at the counter control the operation.

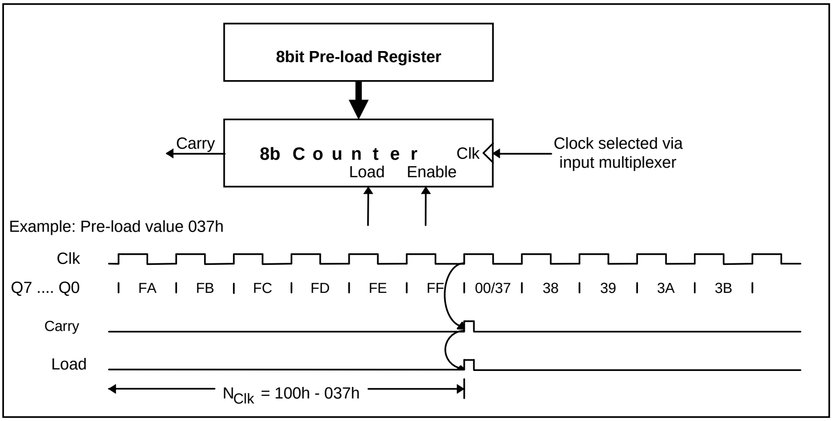


Figure 10.6: Schematic of 8-bit Counter

One of the two inputs controls the load function. A load operation loads the counter with the data of the pre-load register. A write access to the counter results in loading the pre-load register contents into the counter.

The software writes or reads the pre-load register with full control over all instructions. The pre-load register acts as a buffer, and can be written immediately after the load of the counter has completed.

The second of the two inputs enables the count operation. When the enable signal is set high, the counter will count-up each time a positive clock edge is applied to the clock input of the counter.

8-bit Control Register

The information stored in the 8-bit control register selects the operating mode of the timer/counter and controls the function.

Input clock selector

Two signals out of the 8-bit control register select the source for the clock input of the 8-bit up-counter. The four sources are the system clock MCLK, the auxiliary clock ACLK, the external signal from pin P0.1 and the signal from the logical .AND. of MCLK and pin P0.1.

Edge detection

Serial protocols like UART protocol needs start-bit edge detection to determine the start of a data transmission at the receiver.

Input and output data latch, RXD_FF and TXD_FF

The clock to latch data into the input and output data latch is the carry signal from the 8-bit counter. Both latches are used as single bit buffers and change their outputs with the pre-defined timing.

10.2.2 8-bit Timer/Counter Registers

The Timer/Counter module hardware is controlled using access via the 8-bit MDB structure and MAB. It should be accessed using byte instructions.

Register	short form	Register type	Address	Initial state
• T/C control register:	TCCTL	Type of read/write	042h	Reset
• Pre-load register:	TCPLD	Type of read/write	043h	Unchanged
• Counter:	TCDAT	Type of read/(write)	044h	Unchanged

8-bit Timer/Counter Control Register

The information stored in the control register determines the operation of the timer/counter.

TCCTL
042h

7							0
SSEL1	SSEL0	ISCTL	TXE	ENCNT	RXACT	TXD	RXD
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r(-1)

Figure 10.7: 8-bit Timer/Counter Control Register

Bit 0: The RXD bit is read only. The signal from the external pin P0.1 is latched with the carry of the 8-bit counter. The external signal is scanned in a fixed timing sequence independent of the different run-times from software.

Bit 1: The TXD register bit is the buffer for the TXD signal clocked out with the carry from the 8-bit counter at the pin P0.2.

Bit 2: The RXACT bit controls the edge detect logic. The edge detect logic needs a reset ENCNT bit (bit 3) for proper counter enable operation.
RXACT = 0: The edge detect FF is cleared and it can not be the source of enabling the counter operation.
RXACT = 1: The edge detect FF is enabled for operation. A positive or negative edge at pin P0.1, selected by P0IES.1, sets the FF and the counter is prepared for count operation. If the FF is set it remains set.

Bit 3: The ENCNT bit sets the counter enable signal. The 8-bit counter increments its value with each rising edge at clock input.
Together with the RXACT bit (bit2, '0') this bit provides start/stop operation.

Bit 4: The TXE signal controls the 3-state output buffer for the TXD bit.
TXE = 0: 3-state
TXE = 1: Output buffer active.

Bit 5: The ISCTL signal controls the interrupt source between the I/O pin P0.1 and the carry of the 8-bit counter.
ISCTL = 0: The I/O pin P0.1 is the source of interrupt P0IFG.1.
ISCTL = 1: The carry from the 8-bit counter is the source of interrupt P0IFG.1.

BIT 6,7: The bits SSEL0 and SSEL1 select the source of the clock input.

SSEL1	SSEL0	Clock source
0	0	Signal at pin P0.1 according to P0IES.1
1	0	MCLK
0	1	ACLK
1	1	Signal pin P0.1(according to P0IES.1) .AND. MCLK

8-bit Timer/Counter Pre-load Register

The information stored in the pre-load register is loaded into the 8-bit counter when a write access to the counter (TCDAT) is performed:

```
;=========      Definitions ============================================
Dummy           .EQU    0         ; Value for dummy is not loaded into
                                  ; counter
TCDAT           .EQU    044h      ; Address of 8-bit Timer/Counter
;=========      Write pre-load register content to 8-bit Timer/Counter =
;
                MOV.B  #Dummy,&TCDAT
;
```

The pre-load register (TCPLD) can be accessed using the address 043h.

8-bit Counter Data

The data of the 8-bit counter can be read using the address 044h. Writing to the counter loads the content of the pre-load register - not the data mentioned by the instruction.

10.2.3 Special function register bits, 8-bit Timer/Counter related

The 8-bit Timer/Counter has no individual interrupt bits; it shares the interrupt bits with the port P0. One bit in the control register TCCTL, the bit ISCTL, selects the interrupt source for the interrupt flag.

The port0 signal P0/RXD.1 or the carry of the 8-bit counter is used for interrupt source. Two bits in the SFR address range and one bit in the port0 address frame handle the interrupt events on P0/RXD.1 :

- P0/RXD.1 Interrupt Enable P0IE.1 (located in IE1.3, initial state is reset)
- P0/RXD.1 Interrupt Edge Select P0IES.1(located in P0IES, initial state is reset)

The interrupt flag is a single source interrupt flag and is automatically reset when the processor system serves it. The enable bit and edge select bit remain unchanged.

10.2.4 8-bit Timer/Counter in UART Applications

The Timer/Counter peripheral incorporates some features to support serial data exchange with software. The data exchange consists of the transmit cycle and receive cycle. The peripheral hardware supports half duplex protocols.

Software operation can be separated into three categories to support the different conditions and requirements of individual applications:

- Control the bit information immediately after each receive cycle
- Control all bits of one frame immediately after each receive cycle
- Receive the complete message, store the frames in memory and check it after completion of receive cycle.

UART Protocol

The UART protocol is a serial bit stream which includes start bit, 1 to 8 data bits, an optional parity bit, an optional address bit and 1 or 2 stop bits. The least significant data bit is sent first after the start bit.

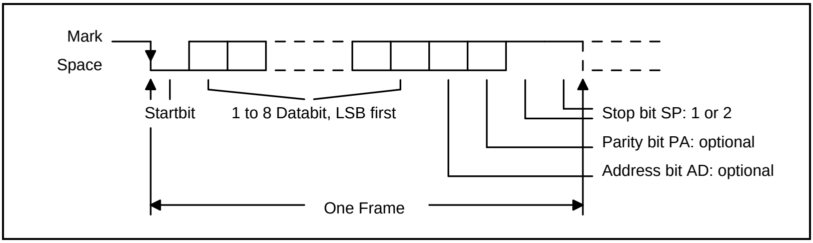


Figure 10.8: Asynchronous communication format

UART Protocol Receive Mode

The Timer/Counter acts as a timer and the carry out latches the bit information available at the pin P0.1. The negative edge - from mark to space of the start bit - indicates the start of one frame. Each bit is scanned right in the middle.

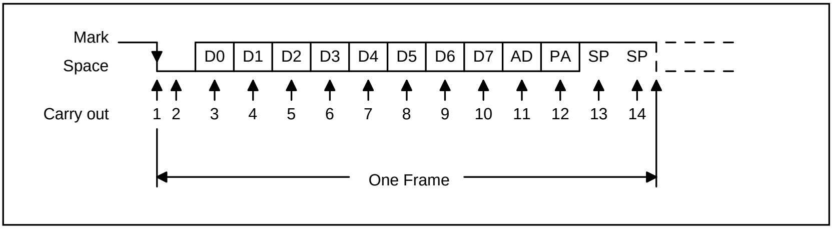


Figure 10.9: Scanning of the asynchronous bits of one frame

The software UART is closely combined with the start of a receive cycle and the baud rate. All timings vary from bit-to-bit if a reference or clock frequency clocks the timer at a frequency which is not a multiple of the selected baud rate. The example in this section should run with a baud rate of 2400 baud with a crystal frequency of 32,768 Hz. The result of these conditions is that each bit interval has got its own timing and therefore its own pre-load value.

UART Protocol Transmit Mode

The Timer/Counter acts as a timer and the carry out latches the bit information available from the control register TCCTL, bit 1: TXD. The software UART should load the TXD bit with the information which should be on I/O pin P0.2. The next carry from the timer latches the TXD bit into the TXD_FF. The transmission of data out of I/O pin P0.2 is enabled by setting of bit TXE in the control register. The reset state of the signal TXE disables the output buffer connected to pin P0.2 and sets parallel the TXD_FF output. This corresponds to the mark state defined for UART format.

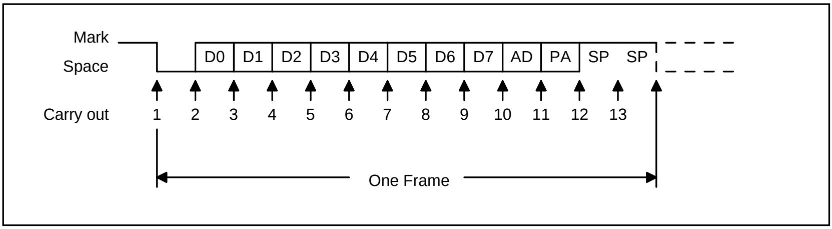


Figure 10.10: Transmitting of the asynchronous bits of one frame

The timing of the transmit part of the software UART depends on the baud rate. All timings vary from bit-to-bit if a reference or clock frequency clocks the timer at a frequency which is a non-multiple of the selected baud rate. The example in this section should run with a baud rate of 2400 baud with a crystal frequency of 32,768 Hz. The result of these conditions is that each bit interval has got its own timing, and therefore its own pre-load value.

Each communication needs features to recognize errors that happen during the data transmission. Four different error conditions are defined:

- Parity Error
- Overrun Error
- Framing Error
- Break detect

In addition to the previous fundamentals, there is an optional function included to support protocol handling: the identification of the start of a block of frames and the destination of the telegram. Two different modes are used in the industry for identification: the idle line multiprocessor protocol, and the address bit multiprocessor protocol.

Idle line multiprocessor mode format

The blocks of data are separated by idle time between them. An idle receive line is detected when ten or more mark state (1's) in a row are received after the first stop bit of a character. When two stop bits are used, the second is counted as the first mark bit of the idle. The first character received after an idle period is an address character. The idle line periods detected by the receiver are illustrated with one and two stop bits:

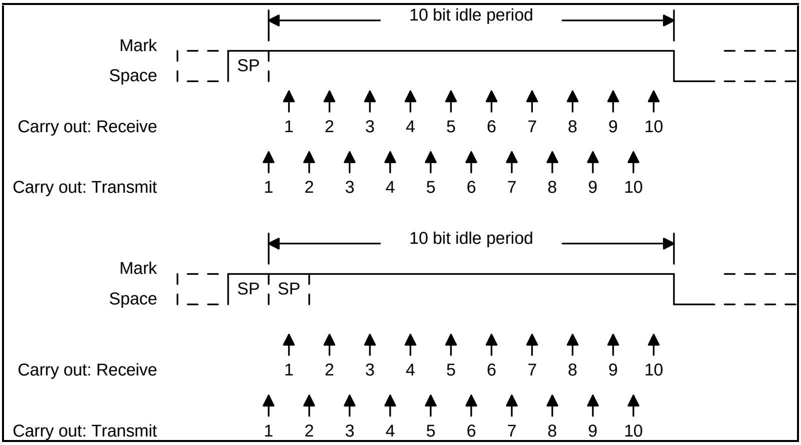


Figure 10.11: UART idle period

It is recommended to transmit an idle period of 11 bits instead of 10 bits.

The precise idle period generates an efficient address character identifier. The first character of a block of frames can be identified as an address. The idle periods of frames within a block of information should not exceed the idle period detect time of 10 bits.

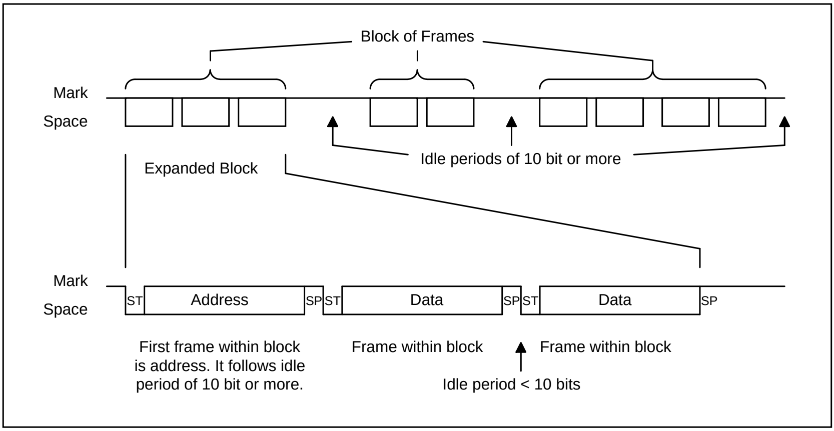


Figure 10.12: Idle line multiprocessor protocol

Address bit multiprocessor mode format

Each character contains an extra bit that represents an address indicator. The first character in a block of data (frames) carries an address bit that is set. This indicates that the character is an address.

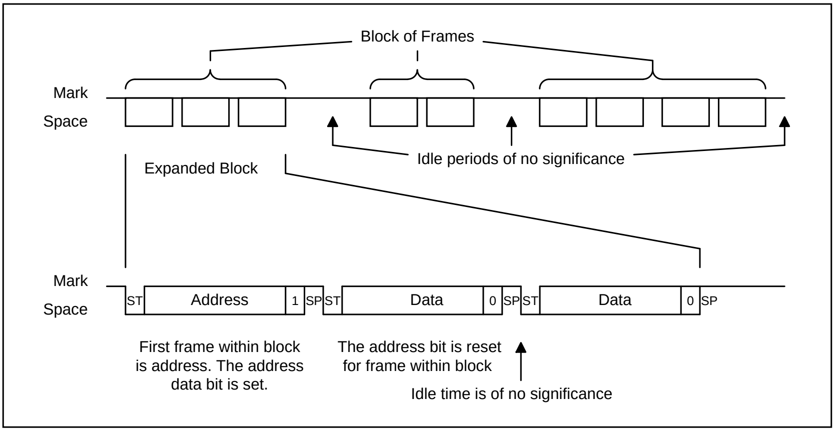

Block of Frames
Mark
Space
Idle periods of no significance
Expanded Block
Mark
Space
ST
Address
1
SP
ST
Data
0
SP
ST
Data
0
SP
First frame within block is address. The address data bit is set.
The address bit is reset for frame within block
Idle time is of no significance

Figure 10.13: Address bit multiprocessor mode format

Transmit/Receive Application Example

This programming example of a serial UART communication protocol, using the features of the 8-bit Timer/Counter, has these characteristics:

- Baud rate 2400 baud
- ACLK = 32 768 Hz
- Parity Even
- Two stop bits
- Half duplex

The carry signal of the Timer/Counter module is selected for the interrupt source instead of the P0.1 source. The associated vector contains the address of the transmit/receive interrupt routine. The first instructions separate the program flow into the transmit or receive part, and use the bit TXRX as an indicator for the running mode.

```
; --------      Define interrupt vector ------------------------------
                .SECT  "RXTX_vec",FFF8     ;Vector of P0.1 or carry from
                                           ;8-bit Timer/Counter
                .Word      VECRXTX         ;Address of UART
                                           ;handler
                ...........
```

The time intervals between two carry signals differs during each transmit or receive cycle. The selected baud rate of 2400 baud and the clock frequency of 32,768 Hz would require a divider of **Error!** = 13.67. The deviation from this ideal factor is accomplished using the sequence 14-13-14 for the division.

```
; Definitions of used expressions
RXD           .EQU   1                 ; Receive data bit in control register TCCTL
TXD           .EQU   2                 ; Transmit data bit in control register TCCTL
RXACT         .EQU   4
ENCNT         .EQU   8                 ; Counter enable bit in control register TCCTL
TXE           .EQU   010h              ; 1: TX buffer active,  0: TX buffer 3-state
ISCTL         .EQU   020h
TCCTL         .EQU   042h              ; Address of Timer/Counter control register
TCPLD         .EQU   043h              ; Address of Timer/Counter pre-load register
TCDAT         .EQU   044h              ; Address of Timer/Counter
BitTime1      .EQU   0100h - 0Eh       ; Error!  sec. = 427.2µs bit length
BitTime1_2    .EQU   0100h - 07h       ; Half of bitime1
BitTime2      .EQU   0100h - 0Dh       ; Error!  sec. =  396.7 µs bit length
AdP0_0        .EQU   0h                ; Interrupt enable 1 register address (SFR)
IEnP0_0       .EQU   08h               ; Bit in Interrupt enable 1 register (SFR)
ParVal        .EQU   0h                ; Parity Even selected
;
(P0.1 respectively TC interrupt-enable)
; Registers or RAM used for data handling
RCstatus      .EQU   0200h             ; RAM (or Register), stores actual status of
                                       ; receive sequence
TXStatus      .EQU   0201h             ; RAM (or Register), stores actual status of
                                       ; transmit sequence
TXData        .EQU   R6                ; Register that contains the transmit data
RCData        .EQU   R6                ; Stores receive data (RXD) in HighByte
Parity        .EQU   0yyyh             ; LSB is actual status of parity. The start value
                                       ; determines odd or even parity
Bend          .EQU 2 x 12
```

The main loop in the program for both the transmit and receive function of an information sequence is demonstrated using the outputting of a table's data and receiving and storing data into a table:

```
; ---------------- Transmitting of frames: a table is to be output ----------------------------------
; Ry points to the table
;
            MOV       #Table2,Ry        ; Start of table copied to Ry
L$5         CRL.B     &TXStatus
            CMP       #TabEnd+1,Ry      ; All frames transmitted?
            JEQ       TabFin            ; Yes, stop transmision and continue program
            MOV.B     @Ry+,TXData       ; Info to TXData
            CALL      #TXInit           ; No, initialize transmission
TXStat      CMP       #Bend,TXStatus    ; output of one frame completed?
            JEQ       L$5               ; Yes, transmit next data of table!
            JMP       TXStat            ; No, wait for completion
            ......
            ......
Table2      .         0xxh Byte         ; Start of table containing data for transmission
```

```
             ......
             ......
TabEnd       .          0zzh Byte          ; End of table containing data for transmission
             ......
TabFin       ......                        ; Transmission of table is completed
                                           ; Continue program here

; ---------------- Prepare receiving of one frame ----------------------------------------------------
; Rx points to the table
;
             MOV        #Table1,Rx         ; Start of receive table copied into Rx
             CALL       #RCPrep            ; Receive of next frame
             ......
; ---------------- Processing part of received frame: Store frame in table1 ----------------------
RECCMPL      RLA        RCData             ; Adjust info to HighByte (remove parity bit)
             SWPB       RCData             ; Swap info to LowByte
             MOV.B      RCData,0(Rx)       ; Store info in table1
             INC        Rx                 ; and pre-increment of table pointer
             CALL       #RCInit            ; Prepare for next frame
             ......                        ; Continue with background program
```

Transmit Mode Application Example:

2400 Baud, ACLK, 8 data bits, Parity Even, Two Stop bits

The transmit mode uses the 8-bit timer/counter, the pre-load register, the control register, the clock selector and the TXD data latch.

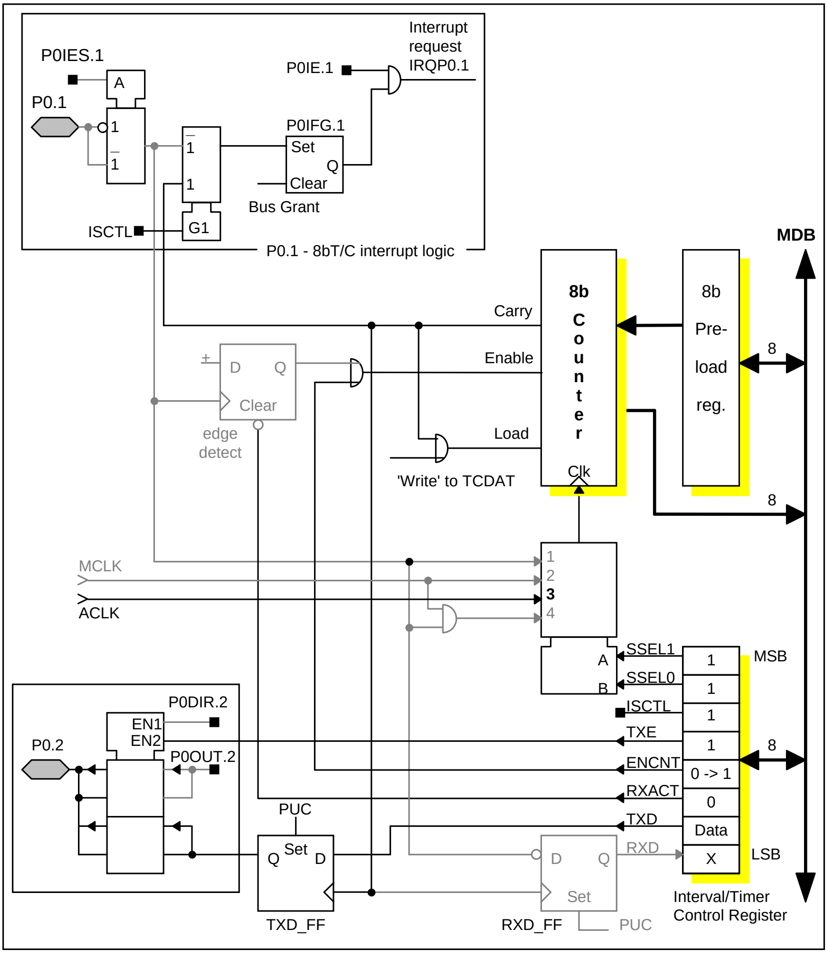


Figure 10.14: 8-bit Timer/Counter configuration for transmit example 2400Baud, ACLK clock

Before a serial communication character transmit starts there are some operation conditions to be defined:

- The output buffer should be enabled -> TXE bit is set.
- The clock input into the 8-bit timer should be selected -> ACLK is selected with SSEL0 bit is reset and SSEL1 bit is set.
- The interrupt source control bit ISCTL selects carry out of timer.

The appropriate bits regarding P0.1 direction and interrupt edge bits should be chosen properly.

- The pre-load register is loaded.
- The write access to the counter loads the pre-load value into the timer .
- The RXACT bit is reset.

```
; ---------------- Prepare Transmit Cycle -------------------------------------------------------------
TXINIT    MOV.B   #BitTime1_2,&TCPLD   ; Load time until start bit starts
                                       ; Disable P0.1 O/P buffer (direction in) if
                                       ; needed
          MOV.B   #072h,&TCCTL         ; TXD = 1 : Defined Start,  ACLK selected
                                       ; TXEN = 1
          MOV.B   #0,&TCDAT            ; Dummy write to load 8b counter/timer
          MOV.B   #BitTime1,&TCPLD     ; Load bit time of first bit for transmission
                                       ; into pre-load register, time of Startbit
          BIS.B   #ENCNT,&TCCTL        ; Set transmit start condition
          BIS.B   #IEnP0_0,&AdP0_0     ; Interrupt enabled for P0.1 in SFR,
                                       ; address is 0.
          CLR.B   &TXStatus            ; Temporary register is prepared.
          MOV.B   #ParVal,Parity       ; ParVal = 0 for Even, ParVal = 1 for Odd
                                       ; Parity
          RET
```

```
; ---------------- Acknowledge Transmit/Receive Cycle, UART Handler ------------------------
; The following two instructions decide whether to transmit or to receive data
; It is necessary because they use a common interrupt vector address
VECRCTX  BIT.B    #RXACT,&TCCTL   ; Test which interrupt handler is active
         JNZ      RCINTRPT        ; Receive mode is active -> Jump
;
TXINTRPT PUSH     R5              ; RXACT = 0 -->  Transmit
         MOV.B    &TXStatus,R5    ; Use TXStatus for
         BR       TXTAB(R5)       ; branching
TXTAB    .Word    TXStat0         ; Startbit      ; Bitime2, 13 clocks of ACLK
         .Word    TXStat1         ; Bit 0, LSB    ; Bitime1, 14 clocks of ACLK
         .Word    TXStat1         ; Bit 1         ; Bitime1, 14 clocks of ACLK
         .Word    TXStat2         ; Bit 2         ; Bitime2, 13 clocks of ACLK
         .Word    TXStat1         ; Bit 3         ; Bitime1, 14 clocks of ACLK
         .Word    TXStat1         ; Bit 4         ; Bitime1, 14 clocks of ACLK
         .Word    TXStat2         ; Bit 5         ; Bitime2, 13 clocks of ACLK
         .Word    TXStat1         ; Bit 6         ; Bitime1, 14 clocks of ACLK
         .Word    TXStat1         ; Bit 7         ; Bitime1, 14 clocks of ACLK
         .Word    TXPar           ; Parity bit    ;  Bitime2, 13 clocks of ACLK
         .Word    TXStop          ; Stop bit 1    ;  Bitime1, 14 clocks of ACLK
         .Word    TXStop          ; Stop bit 2    ;  Bitime1, 14 clocks of ACLK
         .Word    TXCCmpl         ; Frame transmitted
TXStat0  BIC.B    #TXD,&TCCTL
         MOV.B    #BitTime2,&TCPLD
                                  ;Load time 13/32768 [sec] into pre-load register
         JMP      TXRET
TXStat2  MOV.B    #BitTime2,&TCPLD
                                  ;Load time 13/32768 [sec] into pre-load register
         JMP      L$3             ; Shift next bit out at P0.2
TXStat1  MOV.B    #BitTime1,&TCPLD
                                  ;Load time 14/32768 [sec] into pre-load register
L$3      RRA      TXData          ; LSB is shifted to Carry
         JNC      L$1             ; Jump to L$1 if bit = 0
L$2      BIS.B    #TXD,&TCCTL     ; Bit =1, set TXD bit in control register TCCTL
         XOR.B    Parity          ; Count 1's for parity
         JMP      TXRET           ; Bit output completed
L$1      BIC.B    #TXD,&TCCTL     ; Bit =0, reset TXD bit in control register TCCTL
TXRET    INCD.B   &TXStatus       ; Bit output completed
TXStat12 POP      R5
         RETI                     ; Transmit of one bit completed
; ---------------- Parity bit check: Count of 1's in Parity must be even ---------------------------
TXPar    MOV.B    #BitTime2,&TCPLD
         BIT.B    #1,Parity       ; Check parity bit value
         JNZ      L$2             ; Parity bit should be Mark
         JMP      L$1             ; Parity bit should be Space
; ---------------- Output of stop bit(s) -----------------------------------------------------------------
TXStop   MOV.B    #BitTime1,&TCPLD
         JMP      L$2             ; Send stop bit 1 or 2
```

```
; ----------------- Output of one frame completed ------------------------------------------------------
TXCmpl      BIC.B     #IEnP0_0,&AdP0_0 ; Interrupt disabled for P0.2 in SFR,
                                       ; address is 0.
;           BIC.B     #ENCNT,&TCCTL; Stop counter to conserve power consumption
            JMP       TXStat12
; ----------------- End of transmit interrupt handler ----------------------------------------------------
```

Receive Mode Application Example:

2400 Baud, ACLK, 8 data bits, Parity Even, Two Stop bits
The receive mode uses the 8-bit timer/counter, the pre-load register, the control register, the clock selector, the edge detect logic and the RXD data latch.

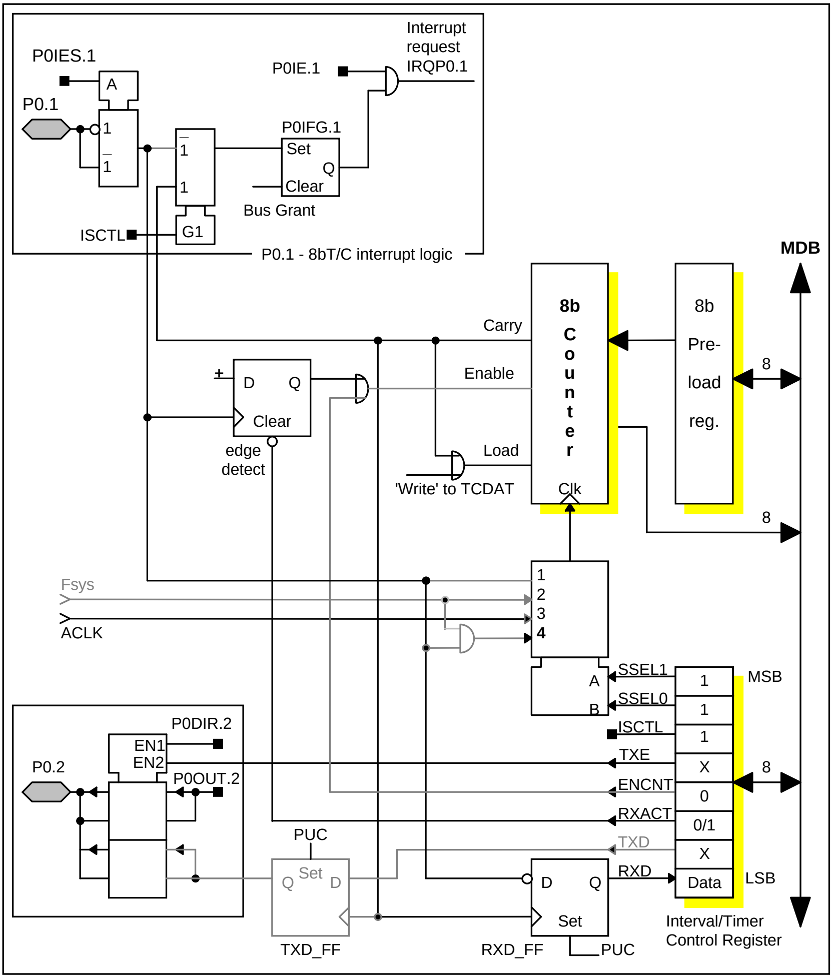


Figure 10.15: 8-bit Timer/Counter configuration for receive example 2400Baud, ACLK clock

Before a serial communication character receive starts there are some operation conditions to be defined (assuming RXACT bit is reset):

- The output buffer should be disabled -> TXE bit is reset.
- The clock input into the 8-bit timer should be selected -> ACLK is selected with SSEL0 bit is reset and SSEL1 bit is set.
- The interrupt source control bit ISCTL selects carry out of timer.

The appropriate bits regarding P0.1 direction and interrupt edge bits should be chosen properly.

- The pre-load register is loaded.
- The write access to the counter loads the pre-load value into the timer .
- The RXACT bit is set.

```
; ---------------- Prepare Receive Cycle ---------------------------------------------------------------
RCPREP   MOV.B   #062h,&TCCTL     ; SSEL0 = 0, SSEL1 = ISCTL = 1,
                                  ; all other bits are cleared
                                  ; Select ACLK for clock source to 8-bit timer
                                  ; Use #072h if TXEN should be enabled
RCINIT   MOV.B   #BitTime1_2,&TCPLD
                                  ; Set Preload register with t1-2 = 0100h - 7
         MOV.B   #0,&TCDAT        ; Prepare timer interval for start bit scanning
         MOV.B   #BitTime1,&TCPLD
                                  ; Set Preload register with Bittime 1 for receive
                                    of first data bit
         CLR     RCstatus         ; Prepare temporary registers
         MOV.B   #ParVal,Parity   ; Register Parity=0 for Even parity receive mode
                                  ; and Parity=1 for Odd parity
         BIS.B   #RXACT,&TCCTL ; activate neg. edge detect of P0.1
                                  ; ( -> RX data )
         BIS.B   #IENP0_0,&ADP0_0
                                  ; Enable interrupt according to P0.1
                                  ; Interrupt source is carry from 8-bit timer
                                  ; according to state of ISCTL
         RET
```

As long as the RXACT and ENCNT bit are reset the timer/counter is halted. The change to the receive active state with a set of RXACT bit enables negative edge detection. The first edge of the start bit, applied to pin P0.1, sets the output of the edge detect latch. It will be set until another reset of RXACT bit is performed.

Once the edge detect latch is set the timer starts operation. The first time interval is started and with the elapse of the programmed time the logical level of pin P0.1 is latched into the RXD latch. After that activity the interrupt is requested.

The interrupt routine for the first bit is optional and can test the presence of a start bit. In the absence of the start bit, the receive cycle is stopped. When the receive cycle is continued the next timing should be prepared by loading the pre-load register with proper data.

All further bits follow nearly the same process in the interrupt routine.

The interrupt routine should handle:

- Store RXD bit information
- Prepare next timing
- Optional: update parity information
  decide program flow on parity error
  look for address bit information
- Test stop bit if received bit should be stop bit.

> **Note: UART protocol, LSB/MSB sequence**
>
> UART protocol shifts the LSB of the data first. In order to collect the data properly, use the RRC instruction, to have the correct order of bits.

```
; ---------------- Receive interrupt handler ----------------------------------------------------------
RCINTRPT    PUSH     R5                  ; Receiver interrupt routine
                                         ; R5 is used temporary as pointer
                                         ; of receive bit
            MOV.B    &RCstatus,R5 ;
            BR       RCTAB(R5)           ;
;
RCTAB       .Word    RCstat0             ; Receive start bit
                                         ; set receive time bit0/LSB, 13ACLK
            .Word    RCstat1             ; Receive bit 0   ; set receive time bit1,
                                         ; 14ACLK
            .Word    RCstat1             ; Receive bit 1   ; set receive time bit2,
            .Word    RCstat2             ; Receive bit 2   ; set receive time bit 3
            .Word    RCstat1             ; Receive bit 3   ; set receive time bit 4
            .Word    RCstat1             ; Receive bit 4   ; set receive time bit 5
            .Word    RCstat2             ; Receive bit 5   ; set receive time bit 6
            .Word    RCstat1             ; Receive bit 6   ; set receive time bit 7: MSB
            .Word    RCstat1             ; Receive bit 7   ; set receive time parity bit
            .Word    RCstat2             ; Receive parity bit
                                         ; set receive time stop bit 1
            .Word    RCstop1             ; Receive stop bit 1
                                         ; set receive time stop bit 2
            .Word    RCstop2             ; Receive stop bit 2
                                         ; set receive time stop bit 2
            .Word    RCCmpl              ; Frame received
; ---------------- Receive start bit: Test for space -------------------------------------------------
RCstat0     BIT.B    #RXD,&TCCTL ; Check start bit
            JC       RCError             ; Error: start bit is Mark not Space
            MOV.B    #BitTime2,&TCPLD ; Start bit fine, load pre-load
                                         ; register
            JMP      RCRET
;
```

```
RCstat2      MOV.B    #BitTime2,&TCPLD
                                          ; Load pre-load register with bit time 2
             JMP      RCBit
RCstat1      MOV.B    #BitTime1,&TCPLD
                                          ; Load pre-load register with bit time 1
RCBit        BIT.B    #RXD,&TCCTL ; RXD bit -> Carry bit
             JNC      RCRET               ; RXD bit = Carry bit = 0 ?, Yes, jump
             RRC      RCData              ; RXD bit -> MSB, Negative bit
             INC.B    &Parity             ; RXD bit = 1, increment '1'-counter
JMP          RCRET1
RCRET        RRC      RCData              ; RXD bit -> MSB, Negative bit
RCRET1       INCD.B   &RCstatus
RCCmpl       POP      R5
             RETI
;
; Parity bit was received just like all other bits. During first stop bit parity is tested
;
RCstop1      BIT.B    #1,&Parity          ; Check parity bit. Bit must be zero.
             JNZ      RCError             ; Parity bit false.
;
RCstop2      MOV.B    #BitTime1,&TCPLD ; Load pre-load register with bit time 1
             BIT.B    #RXD,&TCCTL         ; Check stop bit for Mark
             JNZ      RCRET               ; Stop bit is Mark -> Ok
;
; Error handling: a new start is tried
             RCError POP R5
             CALL     RCINIT              ; Initialize receive routine again
             RETI
```

10.3 The Watchdog Timer

The primary function of the Watchdog Timer module (WDT) is to perform a controlled system restart after a software problem has occurred. If the selected time interval expires, a system reset is generated. If this watchdog function is not needed in an application, the module can work as an interval timer, which generates an interrupt after the selected time interval.

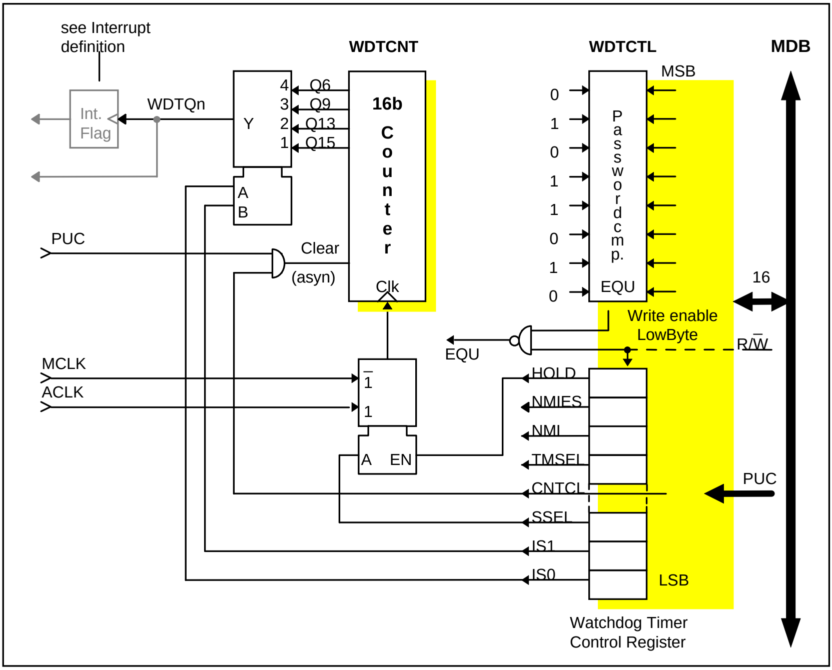


Figure 10.16: Schematic of Watchdog Timer

Features:

- eight software selectable time intervals
- two operating modes: as watchdog or interval timer
- expiration of time interval in watchdog mode generates a system reset; in timer mode, it generates an interrupt request
- for safety reasons, writing to the WDT control register is only possible using a password
- supports ultra-low power feature using hold mode

10.3.1 Watchdog Timer Register

The watchdog timer counter WDTCNT is a 16-bit up-counter which is not directly accessible by software. The WDTCNT is controlled through the watchdog timer control register WDTCTL, which is a 16-bit read/write-register located at the low byte of word address 0120h. Any read or write access should be done with word instructions, using no suffix or suffix '.w'. Writing to WDTCTL is, in both operating modes (watchdog or timer), only possible in conjunction with the correct password.

	7							0
WDTCTL 0120h	HOLD	NMIES	NMI	TMSEL	CNTCL	SSEL	IS1	IS0
	rw-0	rw-0	rw-0	rw-0	r0(w)	rw-0	rw-0	rw-0

Figure 10.17: Watchdog Timer Control Register

Bits 0,1: The bits IS0,IS1 select one of four taps from the WDTCNT.
Assuming fcrystal = 32,768 Hz and fSystem = 1 MHz, the following intervals are possible:

SSEL	IS1	IS0	interval [ms]	
0	1	1	0.064	t_{MCLK} x 2^6
0	1	0	0.5	t_{MCLK} x 2^9
1	1	1	1.9	t_{ACLK} x 2^6
0	0	1	8	t_{MCLK} x 2^{13}
1	1	0	16.0	t_{ACLK} x 2^9
0	0	0	32	t_{MCLK} x 2^{15} <- Value after PUC (Reset)
1	0	1	250	t_{ACLK} x 2^{13}
1	0	0	1000	t_{ACLK} x 2^{15}

Bit 2: The SSEL bit selects the clock source for WDTCNT.
SSEL = 0: WDTCNT is clocked by the system frequency
SSEL = 1: WDTCNT is clocked by ACLK, the crystal frequency (32,768 Hz)

Bit 3: CNTCL bit: In both operating modes writing a '1' to this bit restarts the WDTCNT at 00000h. The read value is not defined.

Bit 4: The bit TMSEL selects the operating mode: watchdog or timer.
TMSEL = 0: Watchdog mode
TMSEL = 1: Interval timer mode

BIT 5: The NMI-Bit selects the function of the RST/NMI-input pin. It is cleared after PUC.
NMI = 0: The RST/NMI input works as Reset input.
As long as the RST/NMI-pin is held 'low', the internal PUC-signal is active (level sensitive).
NMI = 1: The RST/NMI input works as edge sensitive non-maskable interrupt input.

BIT 6: This bit selects the activating edge of the RST/NMI input if NMI function is selected. It is cleared after PUC.
NMIES = 0: A rising edge triggers a NMI-interrupt.
NMIES = 1: A falling edge triggers a NMI-interrupt.

Bit 7: This stops the complete operation of the watchdog counter. It is mandatory to support the ultra-low power features. The clock multiplexer is disabled and the counter stops incrementing. It holds the actual state until the HOLD bit is reset and the operation continues. It is cleared after PUC.
HOLD = 0: Function is fully active.
HOLD = 1: Clock and counter are stopped

Accessing WDTCTL Watchdog Timer Control Register

- Read access:
  WDTCTL can be read without restriction by a password. A read is performed by simply accessing word address 0120h. The Lowbyte contains the value of WDTCTL. The value of the Highbyte is 069h. The value of the Highbyte is selected to 069h and limits the effect of instructions that can alter the WDTCTL register.

  Reading WDTCTL:

	15							8	7 0
WDTCTL 0120h	0	1	1	0	1	0	0	1	Read Data
	r	r	r	r	r	r	r	r	rw-x, (w)
			6			9			

- Write access:
  A write access to WDTCTL is only possible using the correct password in the high-byte. Changing the WDTCTL-register is performed by writing to word address 0120h. The low-byte contains the data to be written to WDTCTL and the high-byte has to be the password which is 05Ah. If any other value than 05Ah is written to the high-byte of address 0120h a system reset is generated.

Writing WDTCTL:

	15							8	7 0
WDTCTL 0120h	0	1	0	1	1	0	1	0	Write Data
	(w)	(w)	(w)	(w)	(w)	(w)	(w)	(w)	rw-x, (w)
			5			A			

10.3.2 Watchdog Timer interrupt control functions

The Watchdog Timer uses two bits in the SFR address range:

- WDT Interrupt Flag WDTIFG (located in IFG1.0, initial state is reset)
- WDT Interrupt Enable WDTIE (located in IE1.0, initial state is reset)

The WDT interrupt flag is reset when power is applied or a reset from the , RST/NMI pin is performed. The signal is called POR. The watchdog interrupt flag indicates whether the watchdog was the reason for a PUC or a power/reset. The vector address is in address 0FFFEh. The enable bit is not relevant.

The Watchdog Timer operates in two different modes. When the WDT is configured to operate in watchdog mode both a watchdog overflow and a security violation trigger the PUC signal which automatically clears the appropriate register bits in the entire system. For the bits in the WDTCTL register, it results in a system configuration where the WDT is set into the watchdog mode and the , RST/NMI pin is switched to reset configuration.

When the WDT is configured to operate in timer mode, the WDTIFG flag is set after the selected time interval, and it requests a standard interrupt service. The WDT interrupt flag is a single source interrupt flag and is automatically reset when the processor system serves it. The enable bit remains unchanged. The WDT interrupt enable bit and the GIE bit should be set to allow an interrupt request situation. The vector address of the interrupt in timer mode is different from that in watchdog mode.

10.3.3 Watchdog Timer Operation

The Watchdog Timer module can be configured in two modes, the watchdog mode and the interval timer mode.

Watchdog mode

After power-on reset or a system reset, the Watchdog Timer module automatically enters the watchdog mode with all bits in watchdog control register WDTCTL and watchdog counter WDTCNT cleared. The initial conditions at the WDTCTL register result in a time interval of 32 ms @ f_{SYS}=1 MHz. Since also the digital controlled oscillator DCO in the system clock generator is set to its lowest frequency, about 32,600 cycles are available for the software to react to such a drastic event. The initial conditions were selected to run the WDCNT with the system frequency f_{SYS} and to allow the application software to start operation with a compromise of the watchdog time in the middle of the time frame.

When the module is used in watchdog mode, the software must periodically reset WDTCNT by writing a '1' to bit CNTCL of WDTCTL to prevent expiry of the selected time interval. If a software problem occurs and the time interval expires because the counter is not reset anymore, a reset is generated and system power-up clear PUC is activated. The system restarts at the same program address as after power-up. The cause of reset can be determined by testing bit0 in the Interrupt Flag Register 1 in the SFR block. The appropriate time interval is selected by setting the bits SSEL, IS0 and IS1 accordingly.

Timer mode

Setting bit TMSEL in the WDTCTL register to '1' selects the timer mode. This mode provides periodic interrupts at the selected time interval. A time interval can also be started under software control by writing a '1' to bit CNTCL in the WDTCTL register.

> **Note: Watchdog Timer, changing the time interval**
>
> Changing the time interval without clearing the WDTCNT may result in an unexpected immediate system reset or interrupt. The time interval should be changed together with a counter clear in one instruction e.g. MOV #05A0Ah, &WDTCTL. Sequential clear and interval select may result in an unexpected immediate system reset or interrupt.
>
> Changing the clock source during normal operation may result in additional clocks for the WDTCNT.

Operation in low- power modes

The system check generator can run in five different modes. With three of them the MCLK and ACLK signals are active. During one mode only the ACLK signal is active and during the other remaining mode neither MCLK nor ACLK signal is active.

The application requirements set the handling of the watchdog timer in combination with the hardware reaction to the different operating conditions of ACLK and MCLK.

CPUOFF mode: Program execution is stopped. The software should define the operating conditions during this operating mode.

MCLKOFF mode/LPM2..3: The ACLK signal is active and MCLK is inactive. When ACLK (32,768 Hz) is selected, the watchdog timer continues operation and will awake the CPU through a system reset or a timer interrupt (if enabled) depending on the selected operating mode. When MCLK is selected the WDT halts operation until MCLK is restarted.

OSCOFF mode/LPM4: The MCLK and ACLK signal are inactive and the watchdog timer counter halts until the system is restarted. The software can reset the counter before entering the OscOff mode depending on the application needs.

The provision of a hold function supports ultra-low power operation. Where an application uses various low power modes, the watchdog timer may be held.

Software example

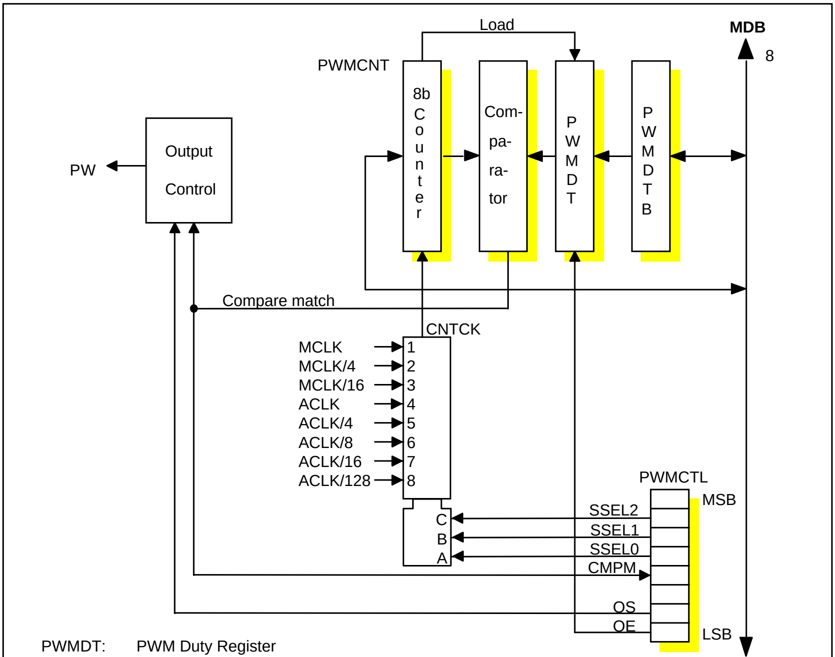
```
; After RESET or power-up, the WDTCTL register and WDTCNT are cleared and the
initial
; operating conditions are watchdog mode with a time interval of 32 ms.
;
;Constant definitions:
;
WDTCTL     .EQU      0120h          ; Address of Watchdog timer
WDTPW      .EQU      05A00h         ; Password
T250MS     .EQU      5              ; SSEL, IS0, IS1 set to 250 ms
T05MS      .EQU      2              ; SSEL, IS0, IS1 set to 0.5 ms
CNTCL      .EQU      8              ; Bit position to reset WDTCNT
TMSEL      .EQU      010h           ; Bit position to select timer mode
;
; As long as watchdog mode is selected, watchdog reset has to be done periodically
; through a instruction e.g.:
;
           ........
           ........
           MOV       #WDTPW+CNTCL,&WDTCTL
    .
    .
;
;To change to timer mode and a time interval of 250 ms, the following instruction
sequence
; can be used:
;
           MOV       #WDTPW+CNTCL+TMSEL+T250MS,&WDTCTL
                                                 ; Clear WDTCNT and
                                                 ; select 250 ms and timer
                                                 ; mode
           ........
           ........
; Note:  The time interval and clear of WDTCNT should be modified within one
         instruction to avoid unexpected reset or interrupt
;
; Switching back to watchdog mode and a time interval of 0.5 ms is performed by:
;
           ........
           ........
           MOV       #WDTPW+CNTCL,&WDTCTL        ; Reset WDT counter
;
           MOV       #WDTPW+T05MS,&WDTCTL        ; Select watchdog mode
                                                 ; and 0.5 ms
           ........
```

10.4 8-bit PWM Timer

Using an 8-bit timer counter, PWM peripheral generates a rectangular output pulse with a duty factor of 0 to 100%. The duty factor is specified by an 8-bit duty control register PWMDT.

The PWM timer module has the following features:

- Selection of eight clock sources
- Duty factors from 0 to 100% with 1/254 resolution
- Output with positive or negative logic



PWMDT:	PWM Duty Register
PWMDTB:	PWM Duty Buffer
PWMCNT:	PWM Counter
PWMCTL:	PWM Control Register

Figure 10.18: Block Diagram of PWM Timer

10.4.1 Operation

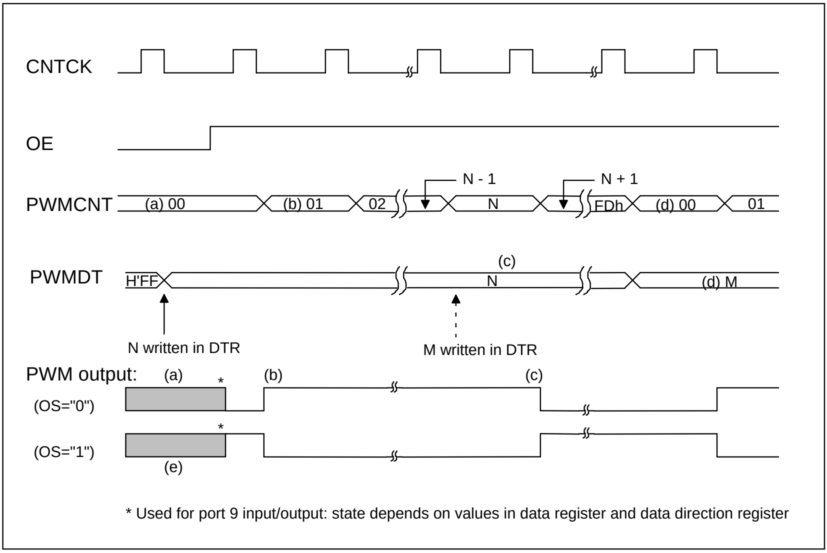
The operation of the PWM timer is described with the output polarity control signal OS reset. The value of the PWMDT register represents the number of clock pulses when PWM output is high.

When OE = 0, the timer count is held at 00h and the PWM output is reset. Any value written into the PWMDT becomes valid immediately.

When OE = 1, the timer counter begins incrementing, and the PWM output goes High (situation b in figure).

When the count reaches the PWMDT value, the PWM output goes Low (situation c in figure) with the next clock pulse.

If the PWMDT value is changed (by writing the data M in figure), the new value becomes valid after the timer count changes from FDh to 00h (situation d in figure)



* Used for port 9 input/output: state depends on values in data register and data direction register

Figure 10.19: PWM timing scheme

The control flag OS in the PWMCTL register defines the polarity of the PWM output.

When OS=0, value in the PWMDT register represents the number of PWM counter clock pulses where the PWM output is set.

When OS=1, value in the output polarity is inverted and the PWMDT register represents the number of PWM counter clock pulses where the PWM output is reset.

10.4.2 PWM Register Descriptions

The PWM timer module is controlled using access via the 8-bit MDB structure and MAB. It should be accessed using byte instructions.

Register	short form	Register type	Address	Initial State
PWM timer control register	PWMCTL.1	R/W	58h	reset
PWM duty buffer	PWMDTB.1	R/W	59h	reset
PWM duty register	PWMDTR.1	R/W	5Ah	reset
PWM timer counter	PWMCNT.1	R/(W)*	5Bh	reset
PWM timer control register	PWMCTL.2	R/W	5Ch	reset
PWM duty buffer	PWMDTB.2	R/W	5Dh	reset
PWM duty register	PWMDTR.2	R/W	5Eh	reset
PWM timer counter	PWMCNT.2	R/(W)*	5Fh	reset

Note: Changing the timer counter

The timer counters are read/write registers, but the write function is for test purposes only.
Application programs should never write to these registers.

Timer Counter PWMCNT

	7							0
PWMCNT 05Bh or 05Fh	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

The PWM timer counter PWMCNT is an 8-bit up-counter. When the output enable bit OE in the timer control register PWMCTL is set, the timer counter starts counting pulses of an internal clock source selected by clock select bits 2 to 0 (SSEL2 to SSEL0). After counting from 00h to FDh, the timer counter repeats from 00h.

The PWM timer counters can be read and written, but the write function is for test purposes only. Application software should never write to the PW timer counter, because this may have unpredictable effects.

The PWM timer counters are initialized to 00h at a PUC, and when the OE bit is cleared.

Duty Buffer Register PWMDTB

	7							0
PWMDTB 059h or 05Dh	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

The duty buffer register holds the value for the duty factor. This duty factor is written into the duty register when the timer counter changes from FDh to 00h.

The duty buffer register PWMDTB is initialized to 00h at a reset and in the OSCOff mode.

Note: Changing the PWM duty factor

Changing the duty factor of the PWM should be done only by writing the new value into the PWM duty buffer PWMDTB. Any write access directly to the duty register can result in a random duty cycle during the running period. The next period will run with the new duty factor now contained in the duty buffer.

Duty Register PWMDT

	7							0
PWMDT 05Ah or 05Eh	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	rw-1	rw-1	rw-1	rw-1	rw-1	rw-1	rw-1	rw-1

The duty register specifies the duty factor of the output pulse. Any duty factor from 0 to 100% can be selected, with a resolution of 1/254. Writing 0h in the PWMDT gives a 0% duty factor; writing 127 (07Fh) gives a 50% duty factor; writing 254 (0FEh) gives a 100% duty factor.

The timer count is continually compared with the duty register's contents. If the PWMDT value is not 0, the PWM output signal is set when the count increments from 00h to 01h. When the count increments to the PWMDT value, the PWM output returns to 0. If the PWMDT value is 0 (duty factor 0%), the PWM output remains constant at 0.

The PWMDT is double-buffered. A new value written in the PWMDTB while the timer counter is running does not become valid until after the count changes from FDh to 00h. After the PUC of the OE bit is reset the timer counter is stopped and new values become valid as soon as written. When the PWMDT is read, the value read is the currently valid value.

The duty register PWMDT is initialized to 0FFh at a reset and in the OSCOff mode.

PWM Timer Control Register PWMCTL

	7							0
PWMCTL 058h or 05Ch	—	SSEL2	SSEL1	SSEL0	CMPM	—	OS	OE
	rw-0	rw-0	rw-0	rw-0	r	rw-0	rw-0	rw-0

The PWM control register is an 8-bit readable/writeable register that selects the clock source and controls the PWM outputs.

Bit 0: Output Enable (OE): This bit enables the PWM counter and the PWM output.
OE = 0: PWM output is disabled. PWMTC is cleared to 00h and stopped.
OE = 1 PWM output is enabled. PWMTC runs.

Bit 1: Output Select (OS): This bit selects a true or inverted signal for the PWM output.
OS =0 Positive logic; positive going PWM pulse, 1 = High (Initial value)
OS =1 Negative logic; negative going PWM pulse, 1 = Low

Bits 2 and 7: Reserved: These bits cannot be modified and are always read as 0.

Bit 3: CMPM: This bit is of read-only type. The output of the compare match signal can be detected. It is read as '1' as long as the timer counter PWMCNT and the duty register PWMDT are identical.

Bits 4 - 6: Clock Select: These bits select one of eight clock sources obtained by dividing the system clock MCLK or the auxiliary clock ACLK.

Bit 6	Bit 5	Bit 4	
SSEL2	SSEL1	SSEL0	Clock source
0	0	0	MCLK
0	0	1	MCLK/4
0	1	0	MCLK/16
0	1	1	ACLK
1	0	0	ACLK/4
1	0	1	ACLK/8
1	1	0	ACLK/16
1	1	1	ACLK/128

From the clock source frequency, the resolution, period, and frequency of the PWM output can be calculated.

Resolution	=	1/clock source frequency	
PWM period	=	resolution x 254	= 254 / clock source frequency
PWM frequency	=	1/PWM period	= clock source frequency / 254
Duty cycle	=	PWMDT/254	

11 Timer_A

This section describes the basic functions of a the general purpose 16-bit Timer_A in MSP430 based system.

Topic **Page**

11.1 Operation of Timer_A

The major blocks of the 16-bit Timer_A are:

- a timer which can count continuously up to a predefined value, count up to a predefined value and down back to zero; the timer can also be stopped
- the clock source of the timer can be selected by software
- the selected clock source can be divided by one, two, four or eight
- five capture/compare registers, each with an individual capture event: two capture signals controlled by hardware or SW
- five output modules, supporting pulse-width modulation requirements.

The Timer_A is configured by means of the bits in the timer control register TACTL. This register defines the basic operation of the 16-bit timer. The input clock source - with its original frequency or pre-divided - and four different operating modes can be selected. Additionally, a clear function and the timer overflow interrupt control bits are included. A timer overflow is defined if the timer counts towards 0000h. This definition is independent of whether the timer counts up or down.

The five capture/compare registers operate identically, and are individually configurable with their control registers.

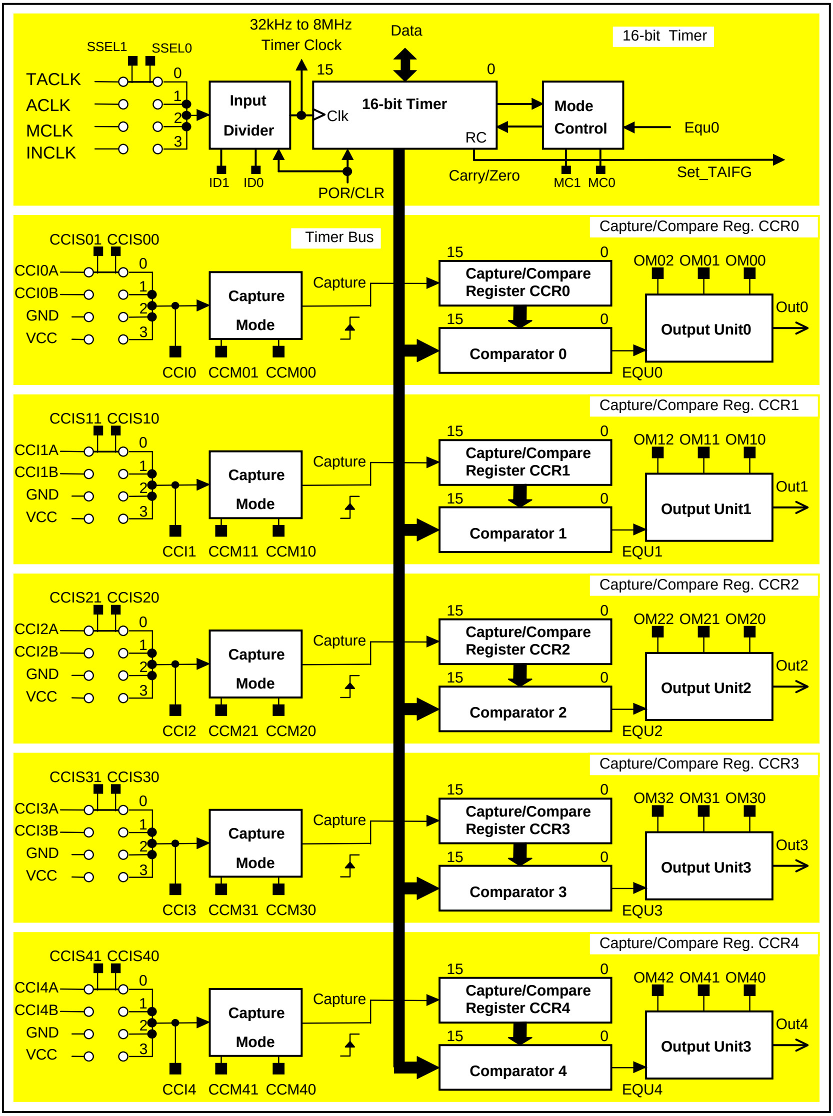
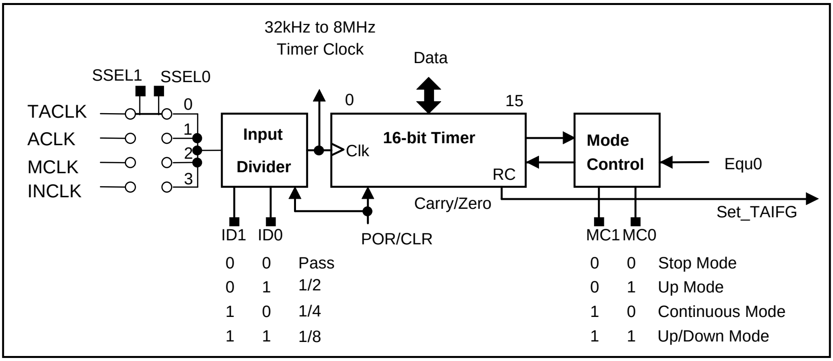


Figure 11.1: Schematic of Timer_A

11.1.1 Timer Operation

Four modes are provided to run the 16-bit timer and are defined with two control bits, MC1 and MC0, in the control register TACTL, plus the signal EQU0 which is the output of the comparator in the capture/compare 0 block. The clock source of the timer is selected via two bits - SSEL1 and SSEL0 - in the control register TACTL. The selected clock source is passed directly to the 16-bit timer or divided by 2, 4 or 8. The source signal can be supplied from internal clocks, or from outside.



ID1	ID0	
0	0	Pass
0	1	1/2
1	0	1/4
1	1	1/8

MC1	MC0	
0	0	Stop Mode
0	1	Up Mode
1	0	Continuous Mode
1	1	Up/Down Mode

Figure 11.2: Schematic of 16-bit Timer

Clock Source Select and Input Divider

The clock source is selected by two control bits, SSEL0 and SSEL1. The output of the multiplexer directly proceeds from the previous selected signal and its level, to the new selected signal and its level. Short intermediate states of the two control bits can select any of the sources applied to the multiplexer. The input divider can receive additional clocks when the clock source is changed. The input divider is reset with POR signal - when VCC is applied or a reset condition at RST/NMI pin is detected - or when the timer is reset via bit CLR. The CLR bit is located in the timer control register TACTL. The input divider remains in its existing state when the timer is modified - even if zero is written to the timer. In normal operation, the existing state of the input divider is not visible for software.

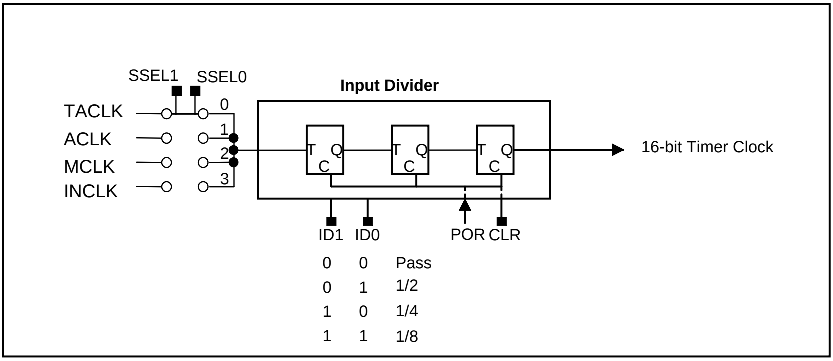


Figure 11.3: Schematic of Clock Source Select and Input Divider

Mode Control and 16-bit Timer

The 16-bit timer is incremented or decremented with each rising clock signal. It can be read and written directly from the software, via standard access to peripheral modules. The different modes are selected with bits MC1 and MC0.

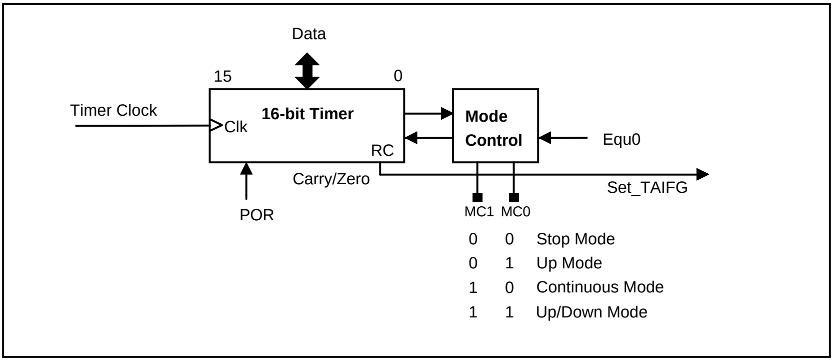


Figure 11.4: Schematic of Timer

During the low state of the timer clock, all operations are prepared which are executed with the following positive edge of the timer clock. Most of the special conditions that are discussed separately are based on this situation. An example of this feature is that a compare fails, if the counter has been already counted the value X and later the capture/compare register is also loaded with this data X.

Four timer operating modes are provided:

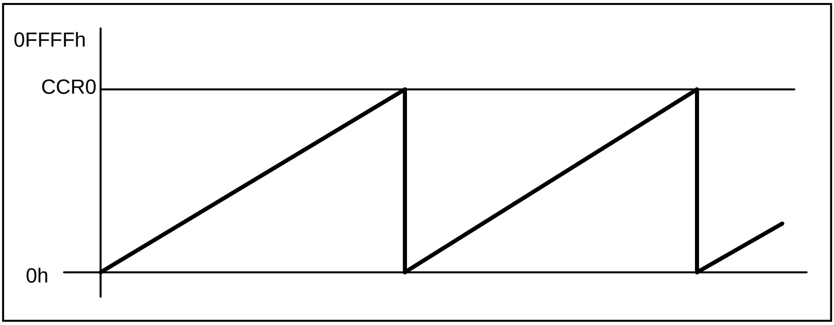
Mode Control:		Mode	Description
MC1	MC0		
0	0	Stop	Timer is halted
0	1	Up	Timer counts upwards until value is equal to value of Compare Register 0
1	0	Continuous	Timer counts upwards continuously
1	1	Up/Down	Timer counts up until the timer's value is equal to Compare Reg. 0 and then down to zero

Stop Mode

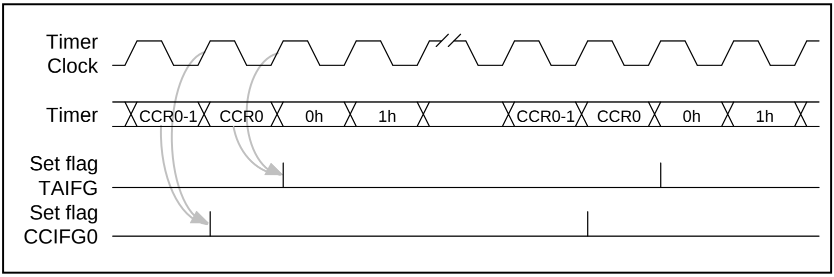
The timer is halted. When released, it counts according to the selected mode, starting from the actual content. The count direction is the same as when stopped. Nothing is reset, the present contents of all registers being used.

UP Mode

The counter counts up to the content of the compare register CCR0. The timer starts counting from the existing value in the timer register. When the timer value and the value of the compare register CCR0 are equal, the timer is reset and restarts counting from zero.

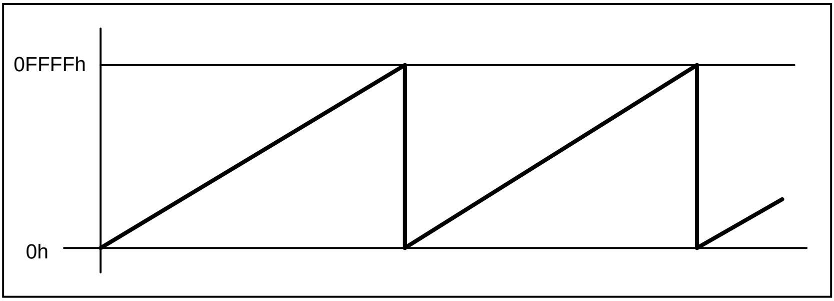


The compare register CCR0 works as the period register in the 'Up Mode'. The counter returns to zero with the next clock when the timer data are equal or greater than the CCR0 data.

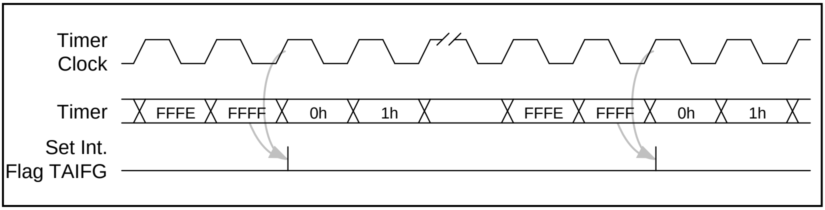


The flag CCIFG0 is set when the timer becomes equal to the CCR0 value. The TAIFG flag is set when the timer counts from CCR0 to zero. All interrupt flags are set independently of the corresponding interrupt enable bit.
An interrupt is requested if the corresponding interrupt enable bit is set and the general interrupt enable bit is set.

Continuous Mode
The timer starts counting from the present value in the timer register. The counter counts up to 0FFFFh and restarts counting from zero.



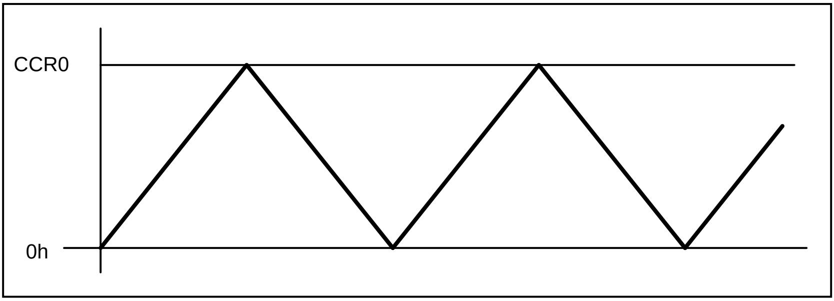
The Continuous Mode is used if more than one timing is needed. The interrupt handler adds to the corresponding compare register CCRx, the time difference from the present time (corresponding data in CCRx), to the time the next interrupt is needed.



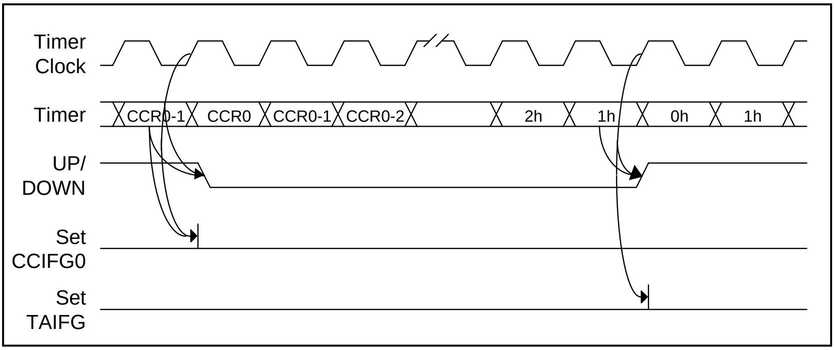
The TAIFG flag is set when the timer counts from 0FFFFh to zero. The interrupt flag is set independently of the corresponding interrupt enable bit. An interrupt is requested if the corresponding interrupt enable bit is set, and the general interrupt enable bit is set.
The capture/compare register CCR0 works the same way as the other compare registers in 'Continuous Mode'.

UP/DOWN Mode

The timer counts up to the content of the compare register CCR0. Then the count direction is reversed, and the timer counts down to zero.



The count direction is latched in a flip-flop FF. The FF is set at 0000h to have the UP condition for the timer, and is reset when the timer value is equal to CCR0, to have the DOWN condition for the timer latched.
The period is defined by the compare register CCR0, and is twice the value in the CCR0 register.



The interrupt flag CCIFG0 is set when the timer has counted up from 'CCR-1' to 'CCR0'. The interrupt flag TAIFG is set when the timer has counted down from 0001h to 0000h.

The Capture/Compare Block

Five identical blocks provide flexible control of real time processing. Any one of the block registers may be used to capture the timer data at the applied event, or for the generation of time intervals. Each time a capture is done or a time interval is completed, interrupts are generated from the appropriate capture/compare block - if the corresponding interrupt is enabled. The mode bit CAPx in the control word CCTLx selects compare (CAPx is reset) or capture (CAPx is set) operation. The capture mode bits CCMx1 and CCMx0 in the control word CCTLx define under which conditions the capture function is performed - if no capture, capture on the leading edge, the trailing edge or at both edges is executed.

Both the interrupt enable bit CCIEx and the interrupt flag CCIFGx are used for capture and compare modes. The CCIFG is set on a capture or compare event. The control bit CAPx defines if it is used for capture or compare.
The capture inputs CCIxA and CCIxB are connected to external pins or internal signals. Different MSP430 devices will have different signals connected to CCIxA and CCIxB.

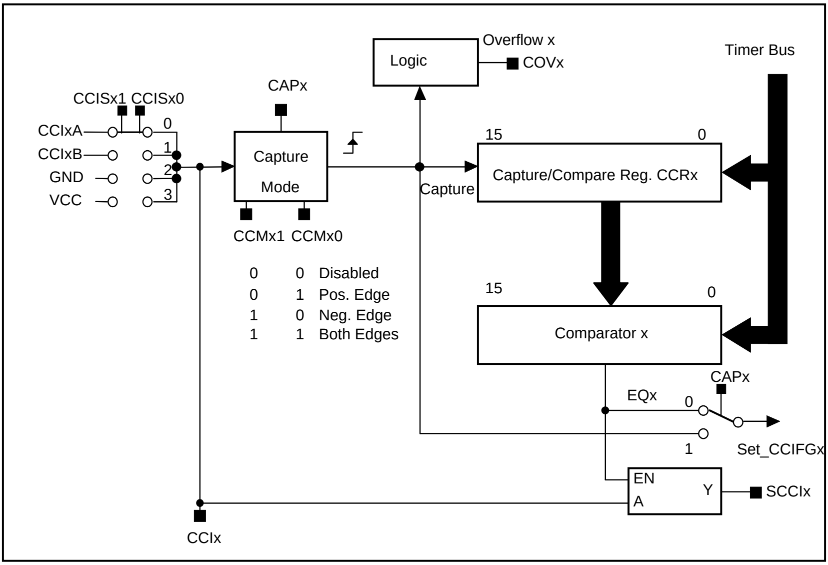
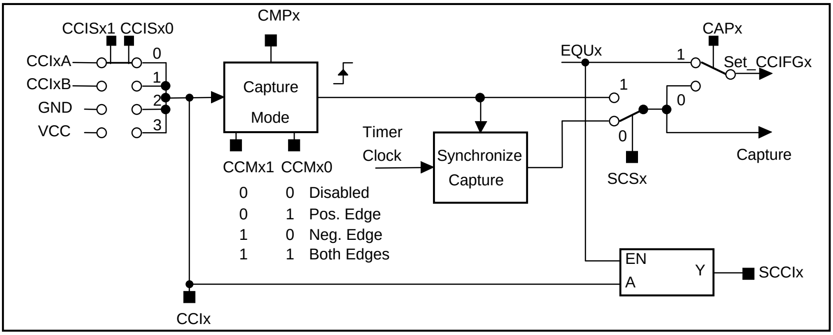
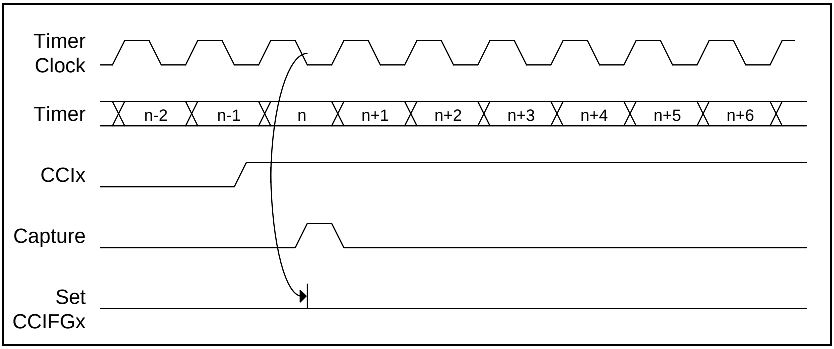


Figure 11.5: Capture/Compare Block

The source of the input signal to the capture logic can be selected by two control bits CCISx1 and CCISx0. It can be read directly by the software via bit CCIx or synchronized with the compare signal EQUx. The synchronized bit SCCIx supports serial protocol software handlers. The capture signal can be asynchronous related to the timer clock. Different application situations are supported by the possibility of using the non-synchronized or the synchronized capture signals.



The capture signal that sets the capture/compare interrupt flag, and stores the timer value into the capture register, is synchronized with the timer clock. It is synchronized to avoid race conditions between the timer data and the capture signal. The synchronized capture signal bit SCSx in the capture/compare control register CCTLx selects the mode of the capture signal.



Applications with slow timer clock are supported using the non-synchronized capture signal. A capture event can have race conditions versus the timer clock, and this results in invalid capture data. The software validates the data and corrects it.

```
; Software example for the handling of asynchronous capture signals
;
; The data of the capture/compare register CCRx are taken by the software
; in the according interrupt routine - they are taken only after a CCRIFG
; was set. The timer clock is much slower than the system clock MCLK
;
CCRx_Int_hand ...                   ; Start of interrupt handler
              ...
              ...
              CMP    &CCRx,&TAR     ; Test if the data CCRX = TAR
              JEQ    Data_Valid
              MOV    &TAR,&CCRx     ; The data in CCRx is wrong,
                                    ; use the timer data
Data_Valid    ...    ...            ; The data in CCRx are valid
              ...
              ...
              RETI
;
```

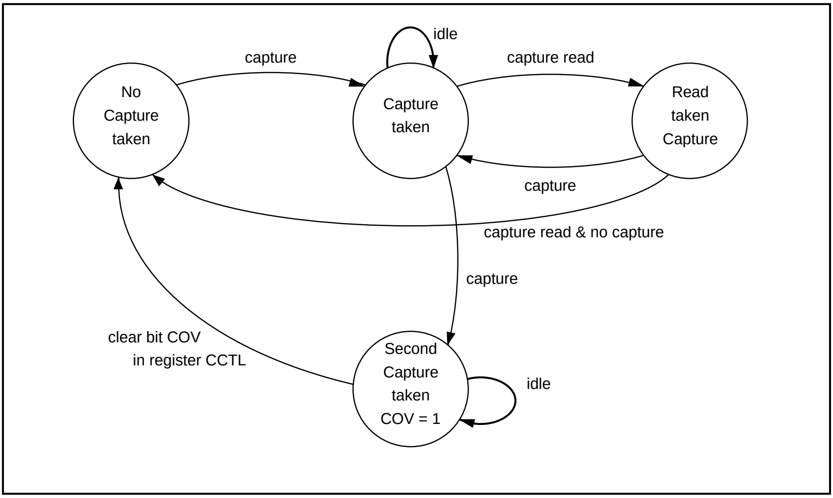
11.1.2 The Capture Mode

The capture mode is selected if the mode bit CAPx - located in the control word CCTLx - is set. The capture mode is used for the accurate fixing of time events. This may be used for speed computations or time measurements. The timer value is copied into the capture register CCRx if the selected edge (positive, negative or both) of the input signal occurs at the selected input pin. Three individual sources can be selected - CCIxA, CCIxB or from the CPU/software via the bits CCISx1/CCISx0 in the capture/compare control register CCTLx.

If a capture was done:

- the interrupt flag CCIFGx - located in the control word CCTLx - is set
- an interrupt is requested, if both interrupt enable bits CCIEx and GIE are set

The capture/compare register CCRx should be accessed with word instructions. It holds the last timer value that was copied to it. An overflow logic is provided. It indicates with its reset state that the capture data were taken before another sub-sequential capture was done. The overflow bit COVx in the register CCTLx is set when a second capture data is latched before the capture/compare register was read successfully. This allows activities for getting back into the lost synchronization.
The capture taken event is reset only if the captured data are completely read before another capture occurred. The overflow bit is set if the read operation is not completed.



The overflow bit COVx needs to be reset by software.

```
; Software example for the handling of captured data looking for overflow
; condition
;
; The data of the capture/compare register CCRx are taken by the software
; and immediately with the next instruction the overflow bit is tested
; and a decision is made to proceed regularly or with an error handler
;
CCRx_Int_hand ...                 ; Start of handler  Interrupt
              ...
              ...
              MOV   &CCRx,RAM_Buffer
              BIT   #COV,&CCTLx
              JNZ   Overflow_Hand
              ...
              ...
              ...
              RETI
Overflow_Hand BIC   #COV,&CCTLx   ; reset capture overflow flag
                                  ; get back to lost synchronization
              ...
              ...
;             RETI
```

Note: Capture with Timer halted

Capture should be stopped when the timer is halted. The sequence should be: stop capturing, and then stop the timer. When the capture function is restarted the sequence should be: start capturing, and then start the timer.

11.1.3 The Compare Mode

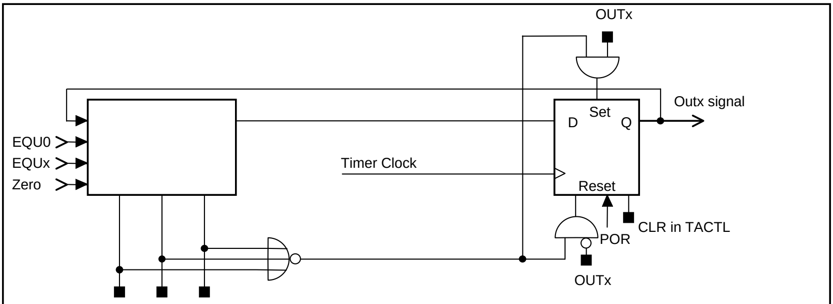
The compare mode is selected if bit CAPx is reset. The bit CAPx is located in the control word CCTLx. All circuitry of the capture hardware is inactive. If the timer becomes equal to the value in the Compare Register x then:

- the Interrupt Flag CCIFGx located in the control word CCTLx is set
- interrupt is requested if the Interrupt Enable Bit CCIEx and GIE bit are set
- the signal EQUx is output to the output unit OUx. Depending on the selected output mode this signal sets, resets or toggles the output OUTx (if OUTMODx > 0).

The capture/compare register CCRx should be accessed with word instructions. It holds the compare value that was written to it. The overflow logic provided for capture mode is inactive.
The EQU0 signal is true when the timer value is greater or equal to the CCR0 value. The EQU1 to EQU4 signals are true when the timer value is equal to the corresponding CCR1 to CCR4 value.

11.1.4 The Output Unit

The output unit supports applications that uses PWM or Digital-to-Analog conversion (DAC). The outputs EQU0 and EQUx of the capture/compare registers control the output logic according to the selected function by three control bits. Five output units OU0 to OU4 - one for each capture/compare block - are implemented. The control bits OMx0, OMx1 and OMx2 are located in the Control Register CCTLx.



OMx2	OMx1	OMx0	
0	0	0	Output Mode, Outx signal is set according to Outx bit
0	0	1	Set, EQUx sets Outx signal clock synchron with timer clock
0	1	0	PWM Toggle/Reset, EQUx toggles Outx signal, reset with EQU0, clock sync. with timer clock
0	1	1	PWM Set/Reset, EQUx sets Outx signal, reset with EQU0, clock synchron with timer clock
1	0	0	Toggle, EQUx toggles Outx signal, clock synchron with timer clock
1	0	1	Reset, EQUx resets Outx signal clock synchron with timer clock
1	1	0	PWM Toggle/Reset, EQUx toggles Outx signal, set with EQU0, clock synchron with timer clock
1	1	1	PWM set/Reset, EQUx reset Outx signal, set with EQU0, clock synchron with timer clock

Figure 11.6: Output Unit

The control bit OUTx determines the Outx signal if the output mode 0 is selected by OMx0, OMx1 and OMx2. The output signal starts with the actual level independent of the selected mode.

UP Mode
The Outx signal is changed when the timer counts up to CCRx, and when the timer counts from CCR0 to zero. The Outx signal is modified according to the selected output mode.

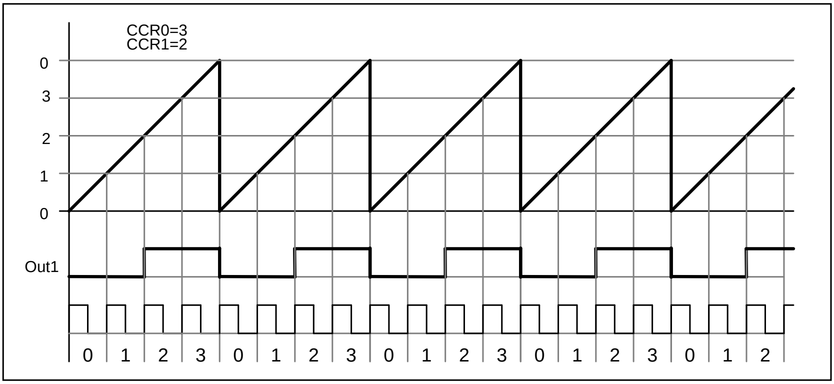


Figure 11.7: Output Unit: Example Up-Mode and Output Mode 3

Continuous Mode

The Outx signal is changed when the timer counts up to CCRx and when the timer counts up to CCR0. The Outx signal is modified according to the selected output mode.

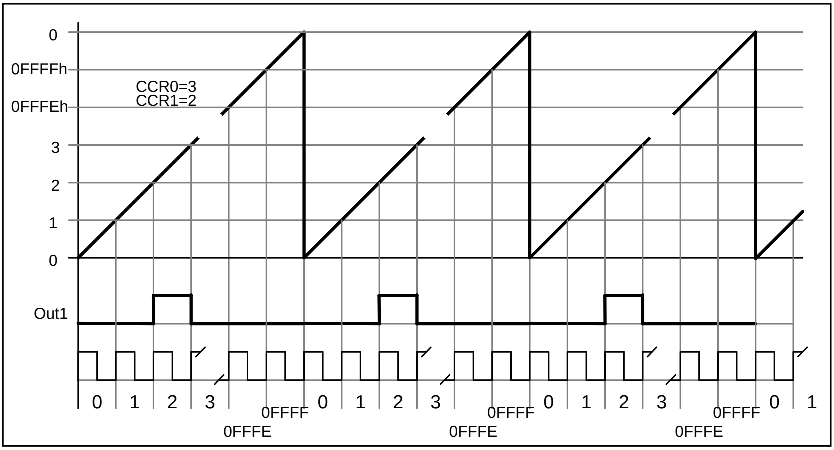


Figure 11.8: Output Unit: Example Continuous Mode and Output Mode 3

UP/DOWN Mode

The Outx signal is changed when the timer counts up to CCRx, and when the timer counts down to CCRx. The Outx signal is modified according to the selected output mode.

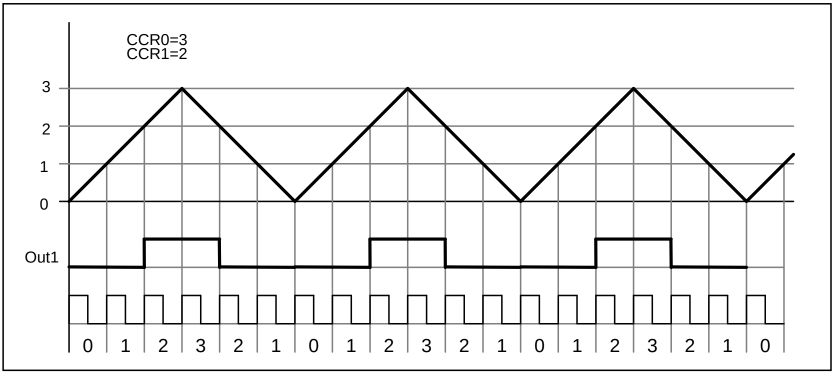


Figure 11.9: Output Unit: Example Up/Down Mode and Output Mode 4

11.2 Registers of Timer_A

The 16-bit Timer_A module hardware is word structured and should be accessed by word processing instructions.

Register	short form	Register type	Address	Initial state
● Timer_A control register	TACTL	Type of read/write	160h	POR reset
● Timer_A register	TAR	Type of read/write	170h	POR reset
● Cap/Com control register0	CCTL0	Type of read/write	162h	POR reset
● Capture/Compare register0	CCR0	Type of read/write	172h	POR reset
● Cap/Com control register1	CCTL1	Type of read/write	164h	POR reset
● Capture/Compare register1	CCR1	Type of read/write	174h	POR reset
● Cap/Com control register2	CCTL2	Type of read/write	166h	POR reset
● Capture/Compare register2	CCR2	Type of read/write	176h	POR reset
● Cap/Com control register3	CCTL3	Type of read/write	168h	POR reset
● Capture/Compare register3	CCR3	Type of read/write	178h	POR reset
● Cap/Com control register4	CCTL4	Type of read/write	16Ah	POR reset
● Capture/Compare register4	CCR4	Type of read/write	17Ah	POR reset
● Interrupt Vector register	TAIV	Type of read	12Eh	(POR reset)

The addresses 16Ch, 16Eh, 17Ch and 17Eh are reserved for future extensions.

11.2.1 Timer_A Control Register TACTL

All control bits regarding the timer and its operation are located in the timer control register TACTL. All control bits are reset automatically by the POR signal, but PUC does not affect them. The control register should be accessed with word instructions.

TACTL

160h

15															0
unused					Input Select			Input Divider		Mode Control		un-used	CLR	TA-IE	TA-IFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	(w)-(0)	rw-(0)	rw-(0)

Bit 0: TAIFG: This flag indicates a timer overflow event.

UP mode: TAIFG is set if the timer counts from CCR0 value to 0000h.

Continuous mode: TAIFG is set if the timer counts from 0FFFFh to 0000h.

UP/DOWN mode: TAIFG is set if the timer counts down to 0000h.

Bit 1: Timer Overflow Interrupt Enable TAIE bit. An interrupt request from the timer overflow bit is enabled if set, and it is disabled if reset.

Bit 2: Timer Clear CLR bit. The timer and the input divider are reset after POR, or if bit CLR is set. The CLR bit is automatically reset by the hardware and always read as zero. The timer starts operation with the next valid input edge. The timer starts in an upward direction if it is not halted by cleared mode control bits.

Bit 3: Not used

Bit 4 to 5: Mode Control Description

MC1	MC0	Count Mode	Comment, Timer ...
0	0	Stop	is halted
0	1	Up to CCR0	counts up to CCR0 and restarts at 0
1	0	Cont. Up	counts up cont. all 65536 steps
1	1	Up/Down	counts up to CCR0, down to 0,.....

Bit 6 to 7: Input Divider control bits

ID1	ID0	Operation	Comment
0	0	Pass	Input signal is passed to the timer
0	1	/2	Input signal is divided by two
1	0	/4	Input signal is divided by four
1	1	/8	Input signal is divided by eight

Bit 8 to 10: Select source of timer input clock signal - preprocessed in the Input Divider

SSEL2	SSEL1	SSEL0	O/P signal	Comment
0	0	1	TACLK	The signal at dedicated ext. pin is used
0	0	1	ACLK	Auxillary clock ACLK is used
0	1	0	MCLK	System clock MCLK is used
0	1	1	INCLK	See device description
1	X	X	-----	Reserved

Bit 11 to 15: Unused

Note: Modify Timer_A

Any write to the timer register TAR when it is operating and ACLK or external clock TACLK is selected can result in unpredictable results. The asynchronous clocks - MCLK used by the CPU and the timer clock can have critical race conditions.

Note: Changing of Timer_A Control bits

If the operation of the timer is modified by the control bits in the TACTL control register, the timer should be halted during this modification. The critical modifications are the input select bits, the input divider bits, and the timer clear bit. Asynchronous input clock situations and system clock (used by the software) can get into race conditions were the timer reacts falsely.
The recommended instruction flow is:

1. Modify the control register and stop the timer.
2. Start the timer operation.

```
E.G.:  MOV   #0286h,&TACTL       ; ACLK/8, timer stopped, timer cleared
       BIS   #10h,&TACTL         ; Start timer with continuous up mode
```

11.2.2 Capture/Compare Control Register CCTL

Each Capture/Compare block has its own control word CCTLx.

	15															0
CCTLx 162h to 16Eh	CAPTURE MODE		INPUT SELECT		SCS	SCCI	un-used	CAP	OUTMODx			CCIE	CCI	OUT	COV	CCIFG
	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

POR resets all bits of CCTLx, PUC does not affect them.

Bit 0: Capture/compare interrupt flag CCIFGx.

Capture Mode: If set, it indicates a timer value was captured in the register CCRx.

Compare Mode: If set, it indicates timer value was equal to the data in the compare register CCRx.

CCIFG0 flag:

CCIFG0 is automatically reset when the interrupt request was accepted according to the interrupt scheme of the MSP430 family.

CCIFG1 to CCIFG4 flags:

The flag which determines the actual interrupt vector word is automatically reset after the TAIV word is read. No vector word is generated if the interrupt enable bit is reset but the flag may be set independently. The flags CCIFG1 to CCIFG4 need to be reset by software.

Bit 1: Capture overflow flag COV.

Compare mode selected, CAP = 0:

The capture signal generation is reset. No capture event will set COV bit.

Capture mode selected, CAP = 1:

The overflow flag COV is set if a second capture is done before the capture register is read. The overflow bit supports software to detect a second capture operation before the previous data are read from capture register. The overflow flag is not reset by reading the capture register.

Bit 2: The OUTx bit is at the corresponding output if OUTMODx is 0 (output only mode).

Bit3: Capture/Compare Input Signal CCIx:
Capture Mode: The selected input signal (CCIxA, CCIxB, VCC or GND) can be read.
Compare Mode: CCI is reset

Bit 4: Interrupt Enable CCIEx: Enables or disables the interrupt request signal of capture/compare block x. Interrupt request is active if enable bit is set, the flag CCIFGx is set and GIE is set.
0: Interrupt disabled 1: Interrupt enabled

Bit 5 to 7:

	Output Mode	Description
0	Output only	Data of OUTx bit determines Outx signal.
1	Set	Comp. signal EQUx sets Outx signal
2	PWM Toggle/Reset	Comp. signal EQUx toggles Outx signal, EQU0 resets Outx signal
3	PWM Set/Reset	Comp. signal EQUx sets Outx signal, EQU0 resets Outx signal
4	Toggle	Comp. signal EQUx toggles Outx signal
5	Reset	Comp. signal EQUx resets Outx signal
6	PWM Toggle/Set	Comp. signal EQUx toggles Outx signal, EQU0 sets Outx signal
7	PWM Reset/Set	Comp. signal EQUx resets Outx signal, EQU0 sets Outx signal

Bit 8: CAP: Defines if the capture/compare block and associated interrupt block acts in capture or compare function.
0: Compare Mode 1: Capture Mode

Bit 9: read only, always read as 0.
Bit 10: Capture/Compare Input Signal SCCIx, synchronized with the compare output EQUx:
The selected input signal (CCIxA, CCIxB, VCC or GND) is stored into a transparent latch with the comparator’s equal signal EQUx and can be read.

Bit 11: The capture/compare signal can used in asynchronous mode or synchronized to the timer clock.
The asynchronous mode (SCS is reset) allows to set the CCIFG immediately on request and also capture the timer data immediately. It will be useful if the period of the capture source is far slower than the timer clock. The data in the capture register may be wrong if race conditions of timer clock and capture source occur.
The synchronous mode (SCS is set) is normally used and the capture data are always valid.
0: asynchronous capture 1: synchronous capture

Bit 12 to 13: Input Select, CCIS1 and CCIS0.

These two bits define the source which provides the capture event in capture mode. During compare mode there is no use of these control bits.

0	Input CCIxA is selected
1	Input CCIxB is selected
2	GND, Low
3	VCC, High

Bit 14 to 15:

Capture Mode		Description
0	Disabled	The capture mode is disabled
1	Pos. Edge	Capture is done with rising edge
2	Neg. Edge	Capture is done with falling edge
3	Both Edges	Capture is done with rising and falling edge

Note: Simultaneous capture and capture mode selection

If the operation of the capture/compare block is modified by the capture/compare bit CAP in the CCRx register from compare to capture mode, no capture should be done simultaneously. The result in the capture/compare register is unpredictable.

The recommended instruction flow is:

1. Modify the control register to switch from compare to capture.
2. Capture.

```
E.G.:   BIS     #CAP,&CCTL2         ; Select capture with register CCR2

        XOR     #CCIS1,&CCTL2       ; Software capture:  CCIS0 = 0

                                    ;                    Capture Mode = 3
```

11.2.3 Timer_A Interrupt Vector Register

Two interrupt vectors are associated with the 16-bit Timer_A module:

- The vector for the capture/compare register CCR0 has the highest priority of all Timer_A interrupts. The capture/compare register CCR0 can be used to define the period during the UP-Mode and the UP/DOWN-Mode. It therefore needs the fastest service.
- The multiplexed vector for the other capture/compare registers. A 16-bit vector word TAIV indicates the currently highest interrupt.

CCR0 Interrupt vector
The interrupt flag associated with the capture/compare register CCR0 is set if the timer value is equal to the compare register's value.

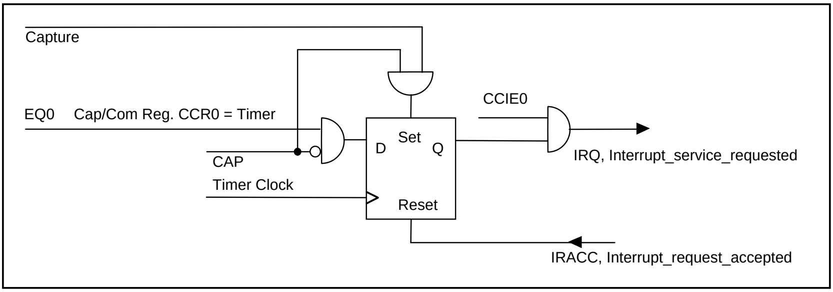

Capture
EQ0 Cap/Com Reg. CCR0 = Timer
CAP
Timer Clock
D
Set
Q
Reset
CCIE0
IRQ, Interrupt_service_requested
IRACC, Interrupt_request_accepted

Figure 11.10: Capture/Compare Interrupt Flag

The capture/compare register 0 has the highest interrupt priority, and uses its own interrupt vector to speed up the real time processing.

Vector word, TAIFG, CCIFG1 to CCIFG4 flags
A vector word is associated with the TAIFG flag and each of the other four capture/compare registers CCR1 to CCR4, and is additionally combined with a priority scheme: the flag CCIFGx with the highest priority generates a number from 0 (no flag set) to 12. This encoded number can be added to the program counter to enter the associated software according to the corresponding interrupt. The vector word TAIV is a 16-bit word to be added to the program counter (see also SW example).

Reading the actual vector word TAIV from the vector word register resets the flag CCIFGx that defines the current vector word.

	15															0
TAIV 12Eh	0	0	0	0	0	0	0	0	0	0	0	Interrupt vector				0
	r0	r0	r0	r0	r0	r0	r0	r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r-(0)	r0

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
	Capture/Compare 3	CCIFG3	Y	6
	Capture/Compare 4	CCIFG4	Y	8
	Timer Overflow	TAIFG	Y	10
Lowest	Reserved		Y	12
	No interrupt pending		Y	0

An interrupt from the timer is requested by setting of CCIFGx or TAIFG, if CCIEx or TAIE is set, and the general interrupt enable bit GIE is set. The bit with the highest priority is requesting the service. When the timer vector word TAIV was accessed the interrupt service requesting bit (CCIFGx or TAIFG) is reset automatically. The bit with the next lower priority now defines the timer vector word TAIV. An interrupt is also requested immediately if any interrupt enable bit (CCIEx or TAIE) is set and the corresponding interrupt flag was already set.
All interrupt flags CCIFGx and TAIFG are featured with full access by the CPU.

Note: Writing to read only register TAIV

When a write to the vector word register TAIV is done the actual interrupt flag that determines the vector word is reset. The requesting interrupt event is missed for later software handling. Additionally, writing to this read only register results in an increased current consumption as long as the write is active.

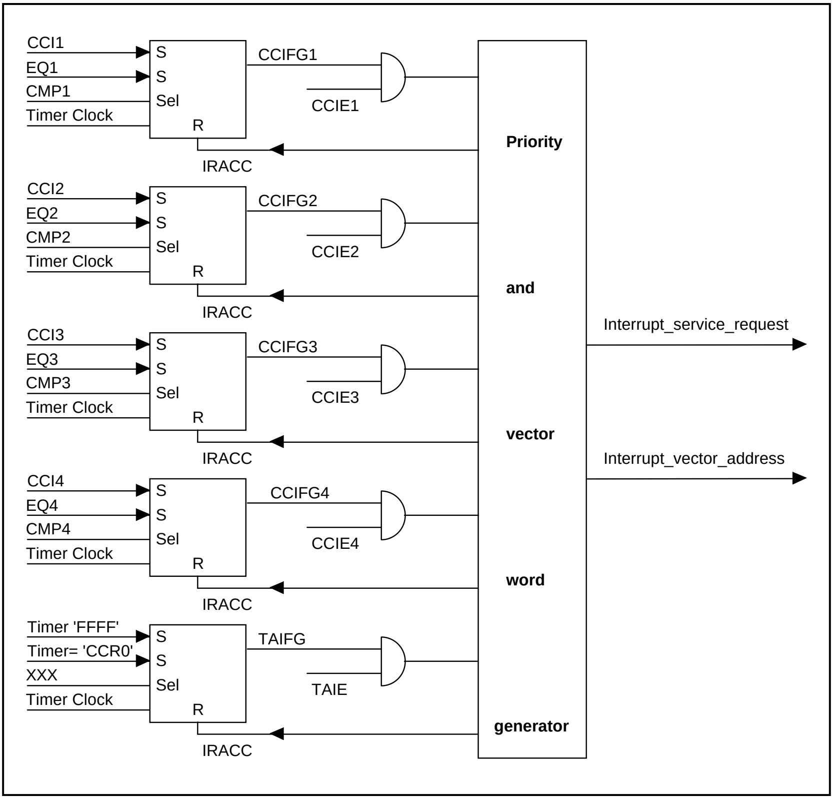


Figure 11.11: Schematic of Capture/Compare Interrupt Vector Word

Timer Interrupt Vector Register, Software Example

The software example shows the use of the vector word TAIV and the overhead of the handling. The numbers at the right margin show the necessary cycles for every instruction. The example is written for continuous mode: the time difference to the next interrupt is added to the corresponding compare register.

```
; Software example for the interrupt part                      Cycles
;
; Interrupt handler for Capture/Compare Module 0.
; The interrupt flag CCIFG0 is reset automatically
;
TIMMOD0  ...                ; Start of handler  Interrupt latency   6
             RETI                                                   5
```

```
;
; Interrupt handler for Capture/Compare Modules 1 to 4.
; The interrupt flags CCIFGx and TAIFG are reset by hardware
; Only the flag with the highest priority responsible for the
; interrupt vector word is reset.
TIM_HND       $                          ; Interrupt latency        6
              ADD    &TAIV,PC            ; Add offset to Jump table 3
              RETI                       ; Vector 0: No interrupt   5
              JMP    TIMMOD1             ; Vector 2: Module 1       2
              JMP    TIMMOD2             ; Vector 4: Module 2       2
              JMP    TIMMOD3             ; Vector 6: Module 3       2
              JMP    TIMMOD4             ; Vector 8: Module 4       2
;
; Module 5. Timer Overflow Handler: the Timer Register is
; expanded into the RAM location TIMEXT (MSBs)
;
TIMOVH                                   ; Vector 12: TIMOV Flag
              INC    TIMEXT       ; Handle Timer Overflow           4
              RETI                                                  5
;
TIMMOD2                                  ; Vector 4: Module 2
              ADD    #NN,&CCR2           ; Add time difference      5
              ...                        ; Task starts here
              RETI                       ; Back to main program     5
;
;
TIMMOD1                                  ; Vector 2: Module 1
              ADD    #MM,&CCR1           ; Add time difference      5
              ...                        ; Task starts here
              RETI                       ; Back to main program     5

; The Module 3 handler shows a way to look if any other interrupt is
; pending: 5 cycles have to be spent, but 9 cycles may be saved if
; another interrupt is pending
;
TIMMOD3                                  ; Vector 6: Module 3
              ADD    #PP,&CCR3           ; Add time difference      5
              ...                        ; Task starts here
              JMP    TIM_HND             ; Look for pending intrpts 2
;
              .SECT  "VECTORS",0FFF0h    ; Interrupt Vectors

              .WORD  TIM_HND      ; Vector for Capture/Compare Module 1..4
                                  ; and timer overflow TAIFG
              .WORD  TIMMOD0      ; Vector for Capture/Compare Module 0
```

If the FLL was turned off, then 2 additional cycles need to be added for synchronous start of CPU system and system clock MCLK.
The software overhead for the different interrupt sources includes the interrupt latency and return-from-interrupt cycles (but not the task handling itself):

• Capture/Compare block CCR0	11 cycles
• Capture/Compare blocks CCR1 to CCR4	16 cycles
• Timer Overflow TAIFG	14 cycles

Timing Limits

With the TAIV register and the above software, the shortest repetitive time distance t_{CRmin} between two events using a Compare Register is:

$$t_{CRmin} = t_{taskmax} + 16 \text{ x } t_{cycle}$$

with: $t_{taskmax}$ Maximum (worst case) time for the task to be done during the interrupt routine (e.g. incrementing of a counter)

t_{tcycle} Cycle time of the used system frequency MCLK

The shortest repetitive time distance t_{CLmin} between two events using a capture register is:

$$t_{CLmin} = t_{taskmax} + 16 \text{ x } t_{cycle}$$

11.3 Timer_A in Applications

11.3.1 Timer_A - Use of the UP-Mode

The UP-Mode is used if the period of the timer should be different to 65,536 clock cycles, which is the period in continuous mode. The capture/compare register CCR0 data is used to define the period of the timer.

Capabilities of output unit OU0
The output unit OU0 works usefully with four modes since CCR0 is also used to define the period of the timer. The four modes are output mode 0, output mode1, output mode 4 and output mode 5. The other four modes can not be used, since they use the EQU0 signal simultaneously in different ways.

Capabilities of output units OU1 to OU4
The output units OU1 to OU4 and its driving circuits are fully identical - all four have the same characteristics. Each can operate in the same or a different way.

The mix - to generate signals or to capture timer data - is selected and controlled by the application software. Examples of the different output mode basic functions are illustrated in the figure. The examples use output OUT1 for demonstration purpose.

Timer: The timer repeatedly runs from 0 up to the value of CCR0.

Output mode 0: The output signal OUTx is defined by the OUTx bit in the control register CCTLx of each capture/compare block, independently of any timing function and completely under software control.

Output mode 1: The output is set when the timer value becomes equal to the capture/compare data CCR1. The interrupt caused by the EQU0 signal (CCIFG0) may be used for modifications of the Compare Registers x.

Output mode 2: The output is toggled when the timer value becomes equal to the capture/compare data CCR1. It is reset when timer value is equal to CCR0 - timer is reset too. This is basically used for PWM functions or together with other outputs to generate phase relations.

Output mode 3: The output is set when the timer value becomes equal to the capture/compare data CCR1. It is reset when timer value is equal to CCR0 - timer is reset too. This is basically used for PWM functions or together with other outputs to generate phase relations.

Output mode 4: The output is toggled when the timer value becomes equal to the capture/compare data CCR1. The output period is double the period of the timer’s period. The phase relation to any other output is determined by selecting the CCRx data.

Output mode 5: The output is reset when the timer value becomes equal to the capture/compare data CCR1. The interrupt caused by the EQU0 signal (CINT0) may be used for modifications of the Compare Registers x.

Output mode 6: The output is toggled when the timer value becomes equal to the capture/compare data CCR1. It is set when timer value becomes equal to CCR0. This is basically used for PWM functions or together with other outputs to generate phase relations.

Output mode 7: The output is reset when the timer value becomes equal to the capture/compare data CCR1. It is set when timer value becomes equal to CCR0 - timer is reset. This is basically used for PWM functions, or together with other outputs to generate phase relations.

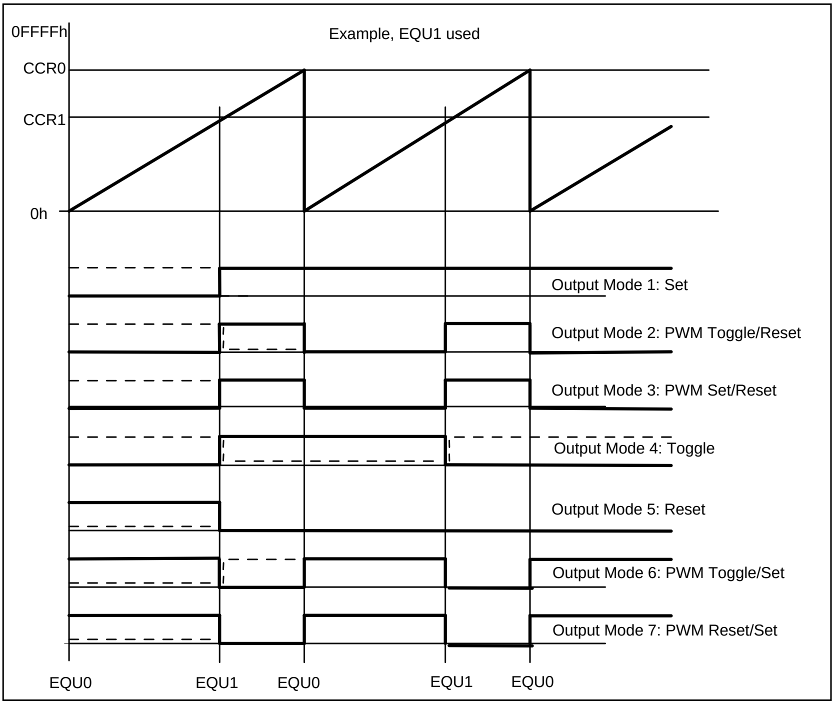

0FFFFh
Example, EQU1 used
CCR0
CCR1
0h
Output Mode 1: Set
Output Mode 2: PWM Toggle/Reset
Output Mode 3: PWM Set/Reset
Output Mode 4: Toggle
Output Mode 5: Reset
Output Mode 6: PWM Toggle/Set
Output Mode 7: PWM Reset/Set
EQU0
EQU1
EQU0
EQU1
EQU0

Figure 11.12: Output Unit in Up Mode

11.3.2 Timer_A - Use of the Continuous Mode

The continuous mode is used if the period of the timer of 65,536 clock cycles is insignificant for the application. A main application of the continuous mode is the generation oft independent software timings. The capture/compare register CCR0 data is used the same way like the other four capture/compare registers CCRx.

All output modes will be useful for various kinds of applications. The feasible output signals for the output modes are chosen by the output mode bits OMx2 to OMx0 in the CCTLx register.

The mix - to generate signals or to capture timer data - is selected and controlled by the application software. Examples of the different output mode basic functions are illustrated in the succeeding figure. The outputs OUT0 and OUT1 are used for demonstration purposes only. The data in CCR0 are greater than the data in CCR1.

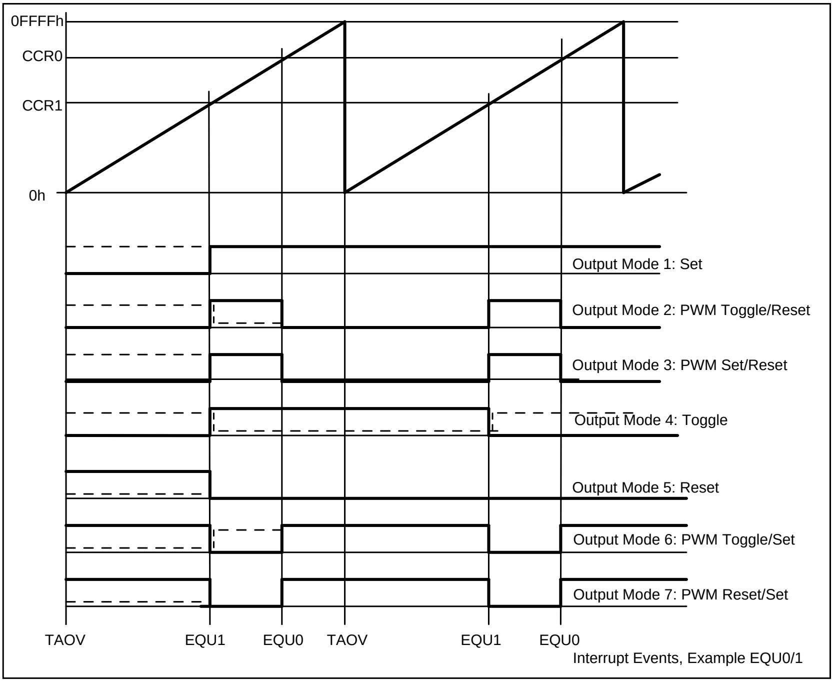

0FFFFh
CCR0
CCR1
0h
Output Mode 1: Set
Output Mode 2: PWM Toggle/Reset
Output Mode 3: PWM Set/Reset
Output Mode 4: Toggle
Output Mode 5: Reset
Output Mode 6: PWM Toggle/Set
Output Mode 7: PWM Reset/Set
TAOV
EQU1
EQU0
TAOV
EQU1
EQU0
Interrupt Events, Example EQU0/1

Figure 11.13: Output Unit in Continuous Mode

Timer: The timer repeatedly runs from 0 up to FFFF.

Output mode 0: The output signal OUTx is defined by the OUTx bit in the control register CCTLx of each capture/compare block, independently of any timing function, and completely under software control.

Output mode 1: The output is set when the timer value becomes equal to the capture/compare data CCR1. The interrupt caused by the EQU0 signal (CCIFG0) may be used for modifications of the Compare Registers x.

Output mode 2: The output is toggled when the timer value becomes equal to the capture/compare data CCR1. It is reset when the timer value is equal to CCR0. This is basically used for pulse generation.

Output mode 3: The output is set when the timer value becomes equal to the capture/compare data CCR1. It is reset when the timer value is equal to CCR0. This is basically used for pulse generation.

Output mode 4: The output is toggled when the timer value becomes equal to the capture/compare data CCR1. The output period is double the period of the timer's period. The phase relation to any other output is determined by selecting the CCRx data.

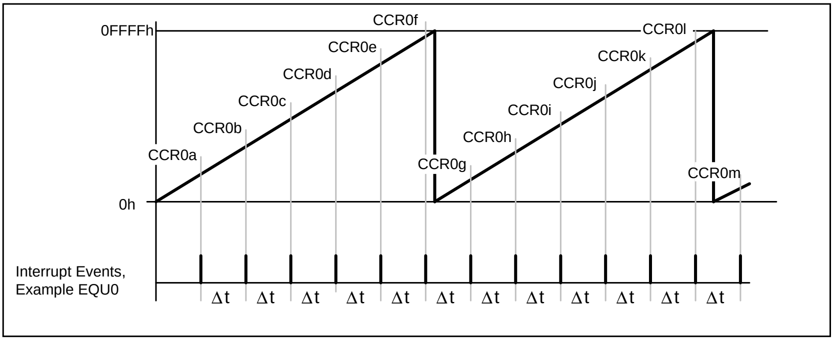
Output mode 5: The output is reset when the timer value becomes equal to the capture/compare data CCR1. The interrupt (CCIFG0) caused by the EQU0 signal may be used for modifications of the Compare Registers x.

Output mode 6: The output is toggled when the timer value becomes equal to the capture/compare data CCR1. It is set when the timer value is equal to CCR0. This is basically used for pulse generation.

Output mode 7: The output is reset when the timer value becomes equal to the capture/compare data CCR1. It is set when the timer value is equal to CCR0. This is basically used for pulse generation.

Continuous Mode - used for time intervals

The continuous mode can be used to generate easily time intervals for the application software. Each time the interval is completed, an interrupt is generated if enabled. In the interrupt routine of this event, the time distance to the next event is added to the capture/compare register CCRx used for this function. Up to five completely independent time events can be generated using all five capture/compare blocks.



Time intervals can be done also with the other modes were CCR0 is used as the period register. There handling is more complex since the sum of the old CCRx data and the new period can be higher than the CCR0 register. When the sum CCRxold plus Δt is greater than CRR0 data, the sum must be reduced by CCR0 data for correct time interval.

11.3.3 Timer_A - Use of the UP/DOWN Mode

The UP/DOWN mode is used if the period of the timer should be different to 65,536 clock cycles and symmetrical pulse waveform generation is needed. The capture/compare register CCR0 data is used to define the period of the timer. The period of the timer is twice the data contained in the CCR0.

Capabilities of output unit OU0
The capture/compare register is used to define the period of the timer. The output unit OU0 only operates effectively in the output mode 0, 1, 4 and 5. All other modes fail, since the timer is already controlled by the CCR0 equal signal EQU0.

Capabilities of output units OU1 to OU4
The output units OU1 to OU4 and its driving circuits are fully identical - all four have the same functions and can operate in different modes.

The mix - to generate signals or to capture timer data - is selected and controlled by the application software. Examples of the different output mode basic functions are illustrated in the succeeding figure. Output OUT3 is used for demonstration purpose only.

Two interrupts are generated during continuous running in the UP/DOWN mode - the interrupt from the capture/compare block CCR0 and the interrupt from the timer, when timer is in down phase and reaches zero. Both interrupts can be used to run proper output pulse modification.

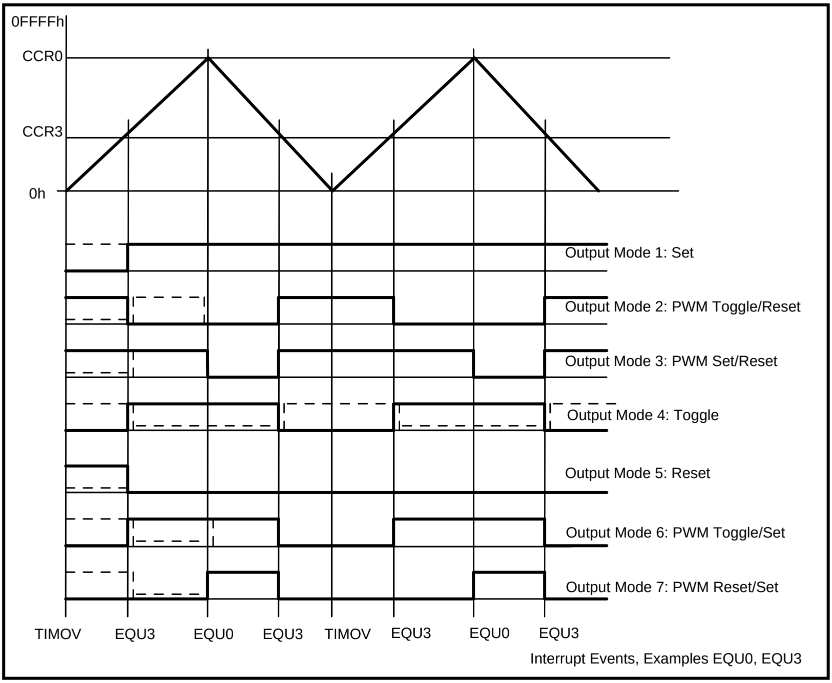

0FFFFh
CCR0
CCR3
0h
Output Mode 1: Set
Output Mode 2: PWM Toggle/Reset
Output Mode 3: PWM Set/Reset
Output Mode 4: Toggle
Output Mode 5: Reset
Output Mode 6: PWM Toggle/Set
Output Mode 7: PWM Reset/Set
TIMOV
EQU3
EQU0
EQU3
TIMOV
EQU3
EQU0
EQU3
Interrupt Events, Examples EQU0, EQU3

Figure 11.14: Output Unit in UP/DOWN Mode(I)

The UP/DOWN mode makes applications possible that enforce the use of "Dead Times" between the output signals. For example, two outputs driving an H-bridge must never be high simultaneously to avoid overload conditions. For a short programmable time - the dead time - both outputs are switched to low. Also the reverse situation is applicable - if necessary the two outputs may be programmed to be never low simultaneously. In the example the t_{dead} is:

$$\mathbf{t_{dead} = t_{timer} \times (CCR1 - CCR3)}$$

with:		
	t_{dead}	Time that both outputs need to be low
	t_{timer}	Cycle length of the Timer Register input frequency
	CCRx	Content of the Compare Register x

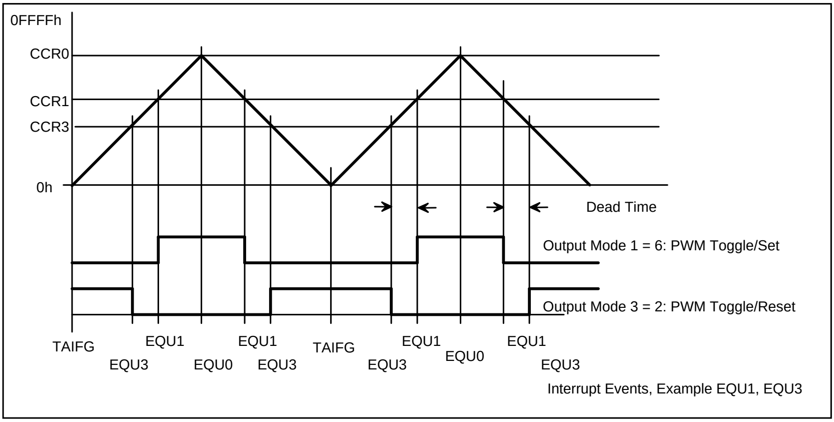

0FFFFh
CCR0
CCR1
CCR3
0h
Dead Time
Output Mode 1 = 6: PWM Toggle/Set
Output Mode 3 = 2: PWM Toggle/Reset
TAIFG
EQU3
EQU1
EQU0
EQU1
EQU3
TAIFG
EQU3
EQU1
EQU0
EQU1
EQU3
Interrupt Events, Example EQU1, EQU3

Figure 11.15: Output Unit in UP/DOWN Mode (II)

11.3.4 Timer_A - Capture via Software

Each of the capture/compare registers can be used by the software to get a time stamp. It can be used for various purposes:

- measure time used by software routines
- measure time between hardware events
- measure the system frequency
- ....

The two bits CCISx1 and CCISx0 and the capture mode selected by the two bits CCMx1 and CCMx0 are use to realize the capture performed by software. The capture mode can be selected to act on the positive edge, negative edge or both edges of the capture signal CCIx. The simplest realization is done when the capture mode is selected to capture on both edges. The capture input signal is selected to be VCC/high or GND/Low. The bit CCISx1 is set and with the bit CCISx0 the capture signal VCC/high or GND/Low is selected.

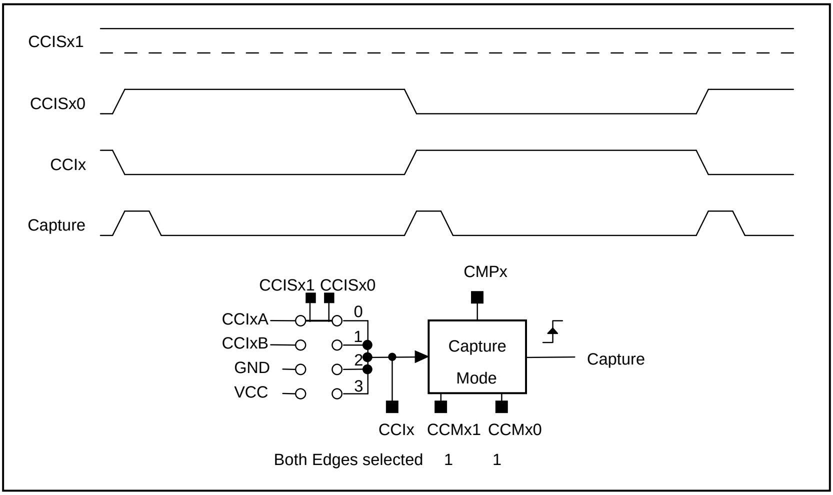


Figure 11.16: Software Capture Example

```
; Software example to capture data performed by software
;
; The data of the capture/compare register CCRx are taken by the software
; It is assumed that CCMx1, CCMxO and CCISx1 bits are set.
; The bit CCISx0 selects the CCIx signal to be high or low
;
;
              ...
              ...
              XOR    #CCISx0,&CCTLx
              ...
              ...
              ...
```

11.3.5 Timer_A - Handle asynchronous serial protocol

The serial asynchronous protocol transmits and receives the data with a defined baudrate. A few different baudrates are defined in the industry. The receive uses the same or another baudrate as that in transmit. The receive starts with a negative edge of the signal. The receiver synchronizes itself with this negative edge, and the following bits are of the selected baudrate.

The transmit feature can be realized by using one compare function to shift data via the output unit to the selected pin. The baudrate is ensured by reconfiguring the compare data along with each interrupt. The output unit sets or resets the pin using the mode 1 for set and mode 5 for reset.

The receive feature can be realized by using one capture/compare function to shift data applied to a pin via the control register’s bit SCCIx into a memory. The receive start time is recognized by capturing the timer data with the negative edge of the receive signal.

The same capture/compare block is then selected to compare. The data for compare is the captured time plus half bit time determined by the baudrate. The first bit is latched with the first compare event EQUx. The scanning of the following bits is done the same way with a timing accordingly to the selected baud rate. The interrupt routine associated with the bit scanning collects all bits of one character for later processing by software.

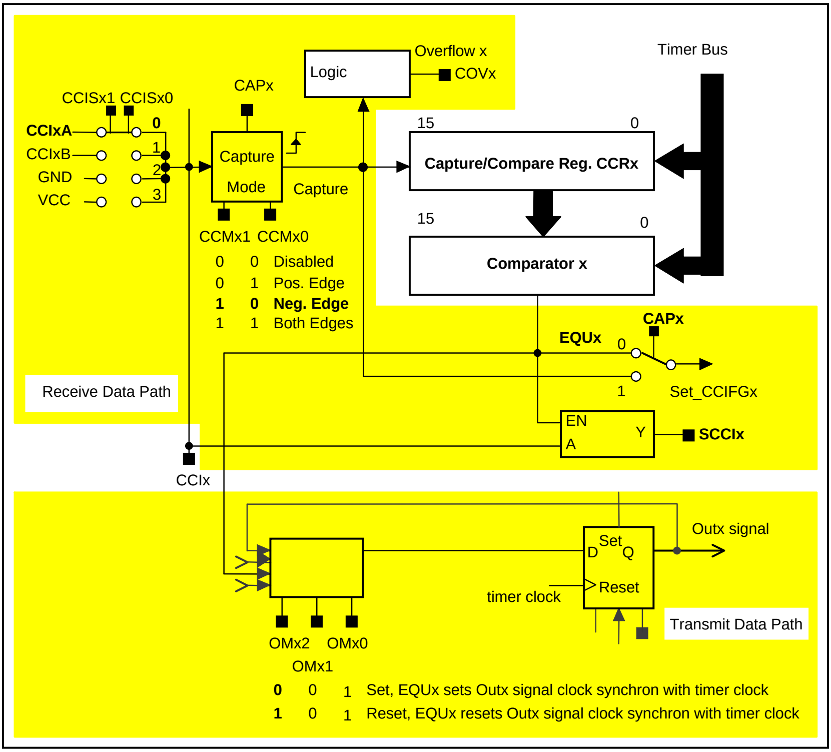


Figure 11.17: Timer_A used to handle asynchronous protocol

One capture/compare block is used when half duplex communication is selected. Two capture compare blocks are used to perform full duplex mode. In half duplex mode, receive and transmit should be sequential and use only one data line. In full duplex mode receive and transmit can be executed in parallel.

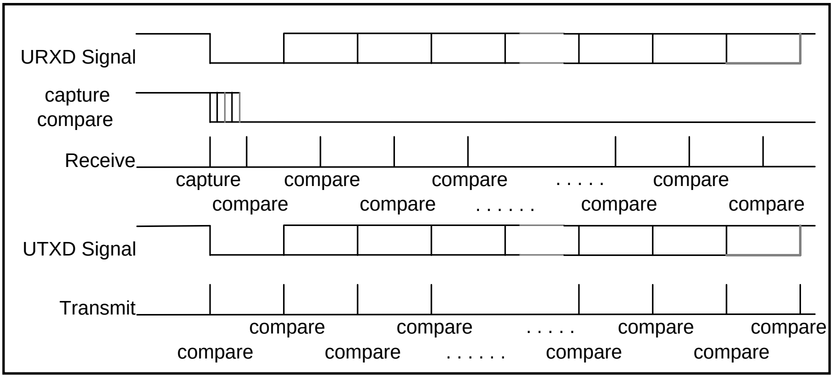


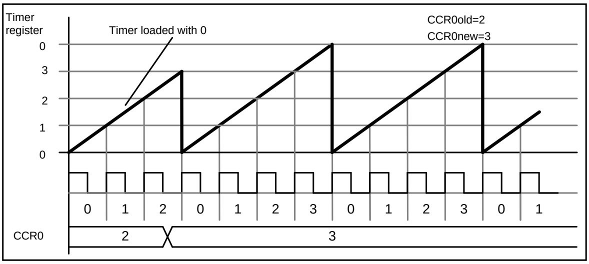
Figure 11.18: Timer_A, timing for asynchronous protocol handling

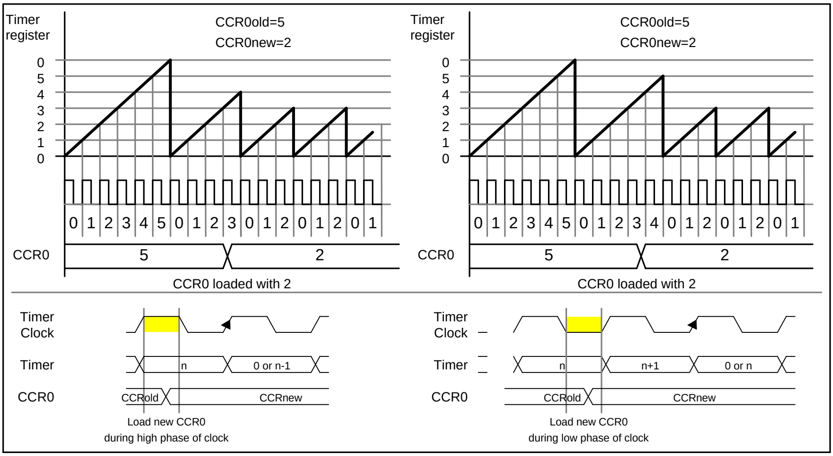
11.4 Timer_A special conditions

There are some special conditions possible, and these will be discussed in this section. A basic principle that follows all the timer and compare functions, is that increment or decrement from the timer register (by a timer clock) is needed to execute the selected function.

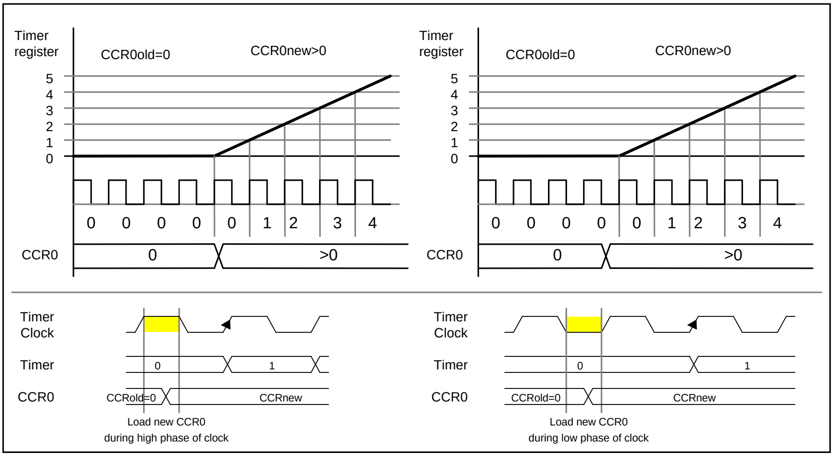
11.4.1 CCR0, used for period register

The compare registers are used for matching with the timer register 180$^{\text{o}}$ before the timer register increments. When the CCR0 is used as a period register, and a new period is the same as or greater than the old period, the timer runs up to the new data and needs no special attention. When the CCR0 is used as the period register, and a new period is less than the old period, the timer is affected with the next positive edge if the new data was written to the CCR0 during the high phase of the timer clock. The timer continues to increment for one further leading edge of the timer clock, and is affected with the second leading timer clock edge if the CCR0 data was written during the low phase of the timer clock.





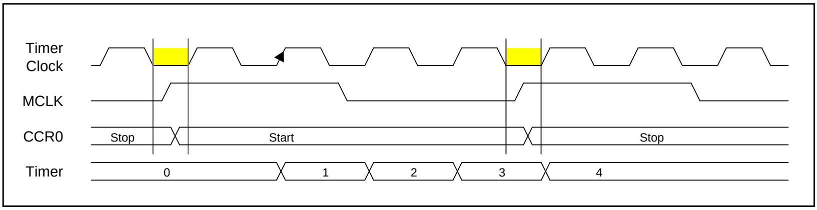
The previous examples demonstrate the different situations in the UP-Mode. The same reaction happens in the UP/DOWN-Mode when the timer operates in up-direction. The timer decrements continuously towards 0 if the period register CCR0 is altered when direction down is active.

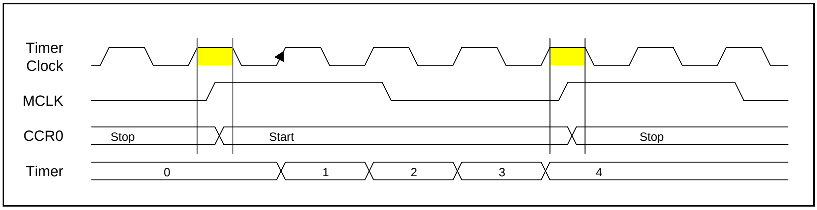


The counter starts this way in Up-Mode and UP/DOWN-Mode.

11.4.2 Start/Stop of the Timer Register

The start of the timer register, and also the stop of the timer register, follow the same basic rules as the period register CCR0.

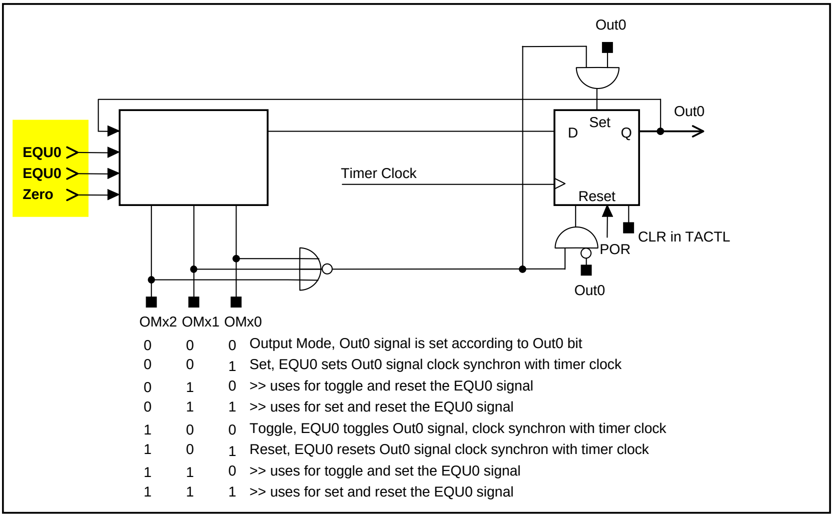




The selected count mode is loaded during the trailing edge of the timer clock. The following leading edge increments the timer register, if one of the three run modes is selected. The following leading edge does not further increment the timer register if the timer register is stopped.

11.4.3 Output Unit0

All output units have identical structures. The inputs use various control signals to define the specific operation. Two of the control signals are the comparator output timer-equal-compare register of the related module x (CCRx), and the comparator output timer-equal-compare register of the module 0 (CCR0). When the module x is the output unit 0, then not all of the possible operating conditions should be used:



OMx2	OMx1	OMx0	
0	0	0	Output Mode, Out0 signal is set according to Out0 bit
0	0	1	Set, EQU0 sets Out0 signal clock synchron with timer clock
0	1	0	>> uses for toggle and reset the EQU0 signal
0	1	1	>> uses for set and reset the EQU0 signal
1	0	0	Toggle, EQU0 toggles Out0 signal, clock synchron with timer clock
1	0	1	Reset, EQU0 resets Out0 signal clock synchron with timer clock
1	1	0	>> uses for toggle and set the EQU0 signal
1	1	1	>> uses for set and reset the EQU0 signal

The modes 0, 1, 4 and 5 are recommended.

Universal Synchronous Asynchronous Receive/Transmit USART

This section describes the serial communication interface USART. It has two functions implemented, to allow serial communication working in different ways. The first function is the well-known asynchronous communication protocol UART; the second function is the serial peripheral interface function SPI, which is also widely used. Even if all the hardware is used in common for both functions, it is described specifically for the function finally chosen, in the application environment which is normally defined to be UART or SPI. Nevertheless, with proper software and hardware design, both functions can be used, one after the other. One bit in the control register defines if the module operates as UART or SPI.

Topic **Page**

USART Peripheral Interface

The universal synchronous/asynchronous interface is a serial channel which allows a serial bit stream of 7 or 8 bits to be shifted into and out of the MSP430, at a programmed rate, or at a rate defined by an external clock. The USART peripheral interface is built to support, with one hardware configuration, two different serial protocols: the universal asynchronous protocol - often simply called RS232 - and the synchronous serial protocol - usually known as the SPI protocol.

The control bit SYNC in control register UCTL is used to select the required mode:

SYNC = 0:	asynchronous - UART - mode selected
SYNC = 1:	synchronous - SPI - mode selected.

The USART is connected to the CPU as a byte peripheral module. It connects the controller to the external system environment by three or four external pins.

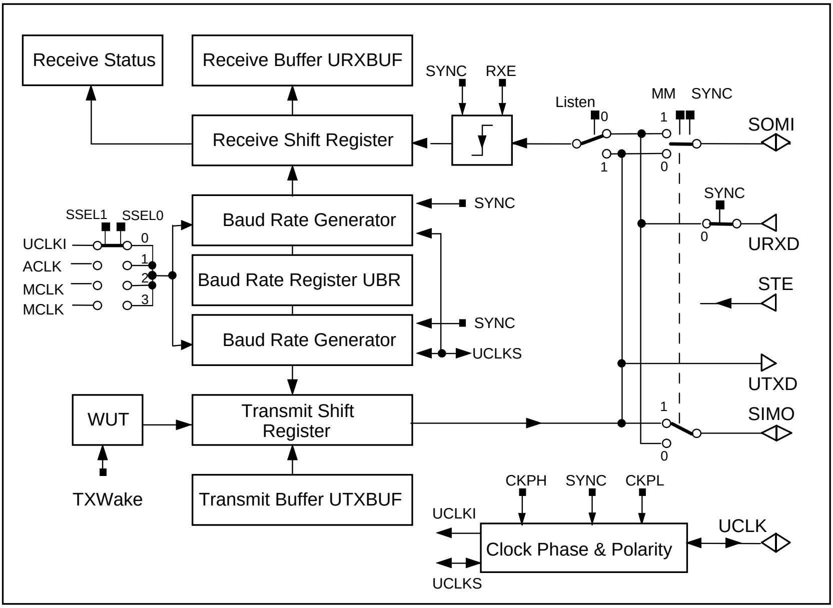


Figure 12.1: Block diagram of USART

12 USART Peripheral Interface, UART Mode

The universal synchronous/asynchronous interface is a serial channel which allows a serial bit stream of 7 or 8 bits to be shifted into and out of the MSP430 at a programmed rate. The asynchronous mode is selected when the control bit SYNC in the USART control register UCTL is reset. The USART is connected to the CPU as a byte peripheral. It connects the controller to the external system environment by three external pins.

USART's serial asynchronous communication feature:

- Asynchronous modes, including Idle line/Address bit communication protocols
- Two shift registers shift serial data stream into URXD, and out on UTXD
- Data transmitted/received with LSB first
- Programmable transmit and receive bit rate
- Status flags

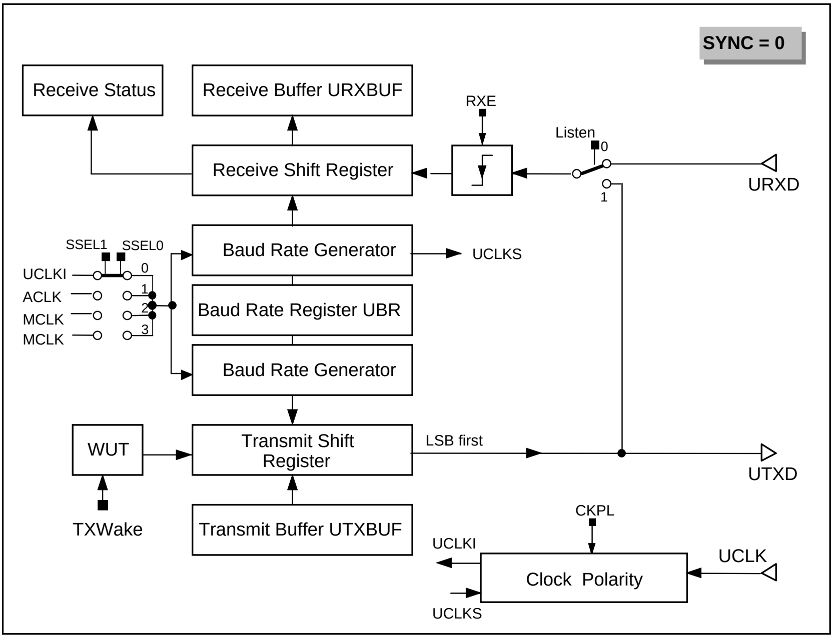


Figure 12.1: Block diagram of USART - UART mode

12.1 Asynchronous Operation

In the asynchronous mode, the receiver synchronizes itself to frames, but the external transmitting and receiving devices do not use the same clock source; the baud rate is generated locally.

12.1.1 Asynchronous Frame Format

The asynchronous frame format consists of a start bit, seven or eight data bits, even/odd/no parity bit, an address bit in Address bit mode, and one or two stop bits. The bit period is defined by the selected clock source and the data in the baud rate registers.

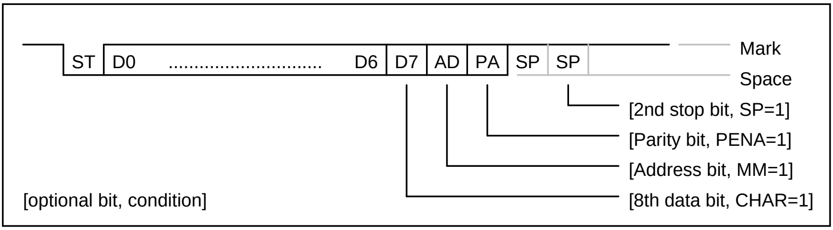


Figure 12.2: Asynchronous frame format

The receive (RX) operation is initiated by the receipt of a valid start bit. It consists of a negative edge at URXD, followed by the taking of a majority vote from three samples, where 2 of the samples must be zero. These samples occur at n/2-x, n/2 and n/2+x of BRCLK periods after the negative edge. This sequence provides false start bit rejection, and also locates the center of bits in the frame, where the bits will be read on a majority basis. The timing of x is $^{1}/_{32}$ to $^{1}/_{63}$ times of BRCLK, but at least BRCLK, depending on the division rate of the baud rate generator.

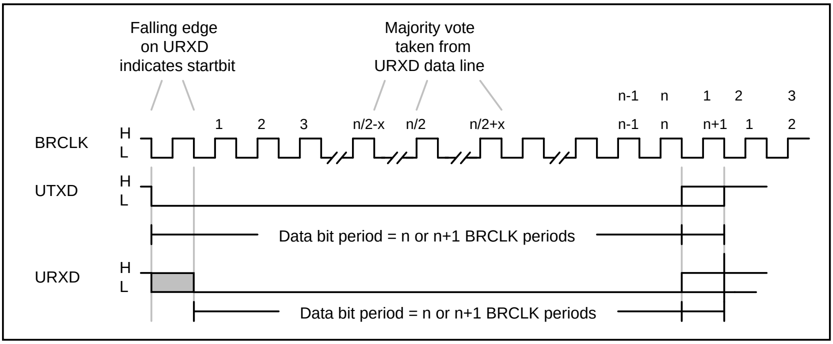


Figure 12.3: Asynchronous bit format. Example for n or n+1 clock periods

12.1.2 Baud rate generation in asynchronous communication format

The baud rate generation in the MSP430 differs from other standard serial communication interface implementations.

Standard Baud Rate Generation
The standard implementation uses a prescaler from any clock source and a fixed second clock divider which is usually a divide by 16.

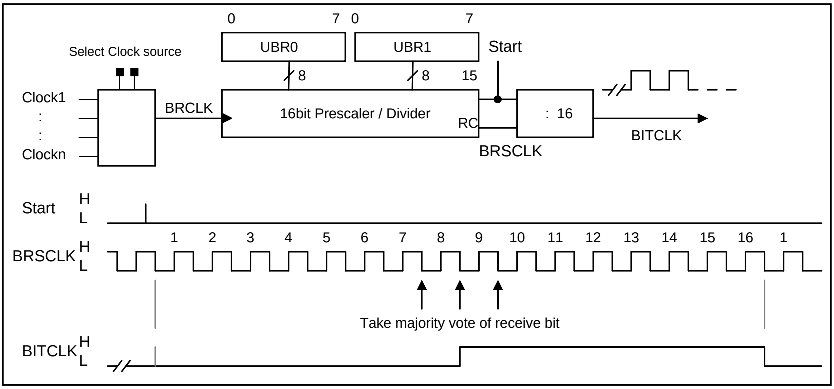


Figure 12.4: Standard baudrate generation - other than MSP430

Baudrate = **Error!**

Using this common scheme to generate the baud rate can not generate baud rates that are chosen close to the frequency of the prescaler's input frequency BRCLK. Division factors of e.g. 18 are not possible, as well as non-integer factors - for example 13.67.

Example 1
Assuming a clock frequency of 32,768Hz for the BRCLK signal, and a required baudrate of 4800 Baud, the division factor is 6.83. In a standard baud rate generator the minimum factor is 16 - the crystal's frequency and the baud rate generation can not meet the requirements.

Example 2
Assuming a clock frequency of 1.04MHz (32 x 32,768Hz) for BRCLK signal and a required baudrate of 19 200 Baud, the division factor is 54.61. In a standard baud rate generator the next factors are 48 (3x16) or 64 (4x16) - the crystal's frequency and the baud rate generation can not meet the requirements. The crystal frequency needs to be selected to meet the communication requirements. Other criteria like current consumption, simple real-time clock function or system cost constraints can not be considered to be favorable.

MSP430 Baud Rate Generation

The baud rate generator of the MSP430 uses one prescaler/divider and a modulator. This combination is used to work properly with crystals whose frequency is not a multiple of the standard baud rates, but allows the protocol to run at maximum baud rate. Using this technique, even with a watch crystal (32,768Hz) baudrates up to 4800 (9600) baud are possible. This gives power advantages, since the selection of sophisticated MSP430 operation in low power mode is possible.

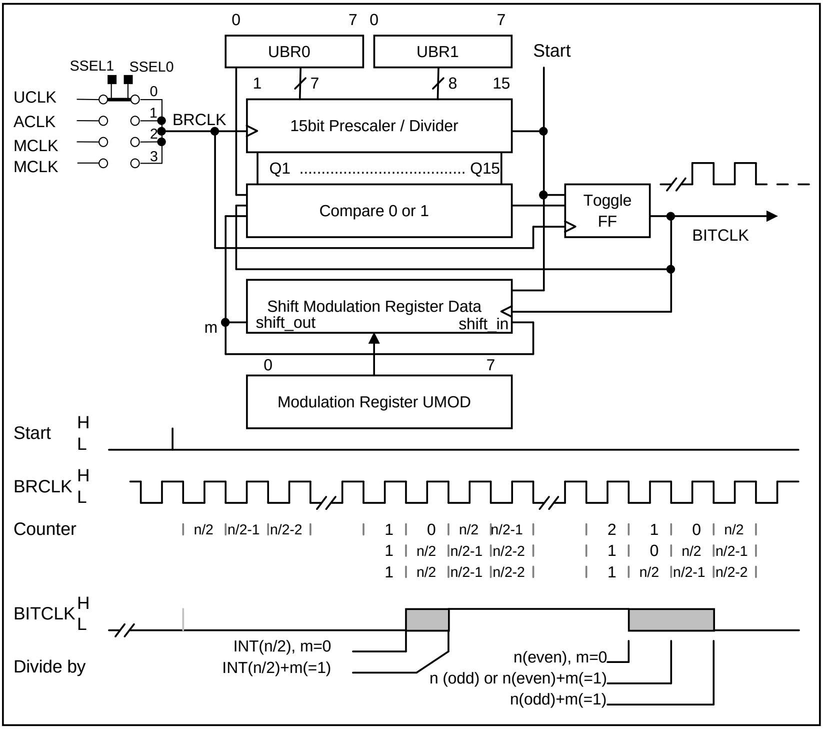


Figure 12.5: MSP430 Baud Rate Generation. Example for n or n+1 clock periods

The LSB of the modulation register is used first for modulation - it starts with the start bit. A set modulation bit increases the division factor by one.

Example 1

Assuming a clock frequency of 32,768Hz for BRCLK signal and a required baudrate of 4800 Baud, the division factor is 6.83. The baud rate generation in the MSP430's USART uses a factor of 6 plus the modulation register loaded with 6Fh (0110 1111). This means the divider runs the following sequence: 7 - 7 - 7 - 7 - 6 - 7 - 7 -6 - .... The sequence repeats after all eight bits of the modulator are used.

Example 2

Assuming a clock frequency of 1.04MHz (32 x 32,768Hz) for BRCLK signal, and a required baudrate of 19 200 Baud, the division factor is 54.61 The baud rate generation in the MSP430's USART uses a factor of 54 (36h) plus the modulation register loaded with 0D5h. This means the divider runs the following sequence: 55 - 54 - 55 - 54 - 55 - 54 - 55 -55 - .... The sequence repeats after all eight bits of the modulator are used.

The standard baud rate data needed for the baud rate registers and the modulation register are listed for the watch crystal 32,768Hz (ACLK) and MCLK, assumed to be 32 times the ACLK frequency. The error listed is calculated for the receive path. In addition to this error, the synchronization error should also be considered.

Baud rate	Divide by		ACLK			max. error %	MCLK (= 32 x ACLK)			max. error %
	ACLK	MCLK	UBR1	UBR0	UMOD		UBR1	UBR0	UMOD	
75	436.91	13981	1	B4	FF	-.1/.3	36	9D	FF	0/.1
110	297.89	9532.51	1	29	FF	0/.5	25	3C	FF	0/.1
150	218.45	6990.5	0	DA	55	0/.4	1B	4E	FF	0/.1
300	109.23	3495.25	0	6D	22	-.3/.7	0D	A7	00	-.1/0
600	54.61	1747.63	0	36	D5	-1/1	06	D3	FF	0/.3
1200	27.31	873.81	0	1B	03	-4/3	03	69	FF	0/.3
2400	13.65	436.91	0	0D	6B	-6/3	01	B4	FF	0/.3
4800	6.83	218.45	0	06	6F	-9/11	0	DA	55	0/.4
9600	3.41	109.23	0	03	4A	-21/12	0	6D	03	-.4/1
19 200		54.61					0	36	6B	-.2/2
38 400		27.31					0	1B	03	-4/3
76 800		13.65					0	0D	6B	-6/3
115 200		9.10					0	09	08	-5/7

Table 12.1: Commonly used Baud Rates, Baudrate data and errorsCommonly

The maximum error is calculated for the receive mode and the transmit mode. The error in the receive mode is the accumulating timing error versus the ideal scanning time in the middle of each bit. The transmit error is the accumulating timing error versus the ideal time of the bit period.

The maximum frequency of MCLK is noted in the device data sheet and can exceed the example frequency.

12.1.3 Asynchronous Communication Formats

The USART module supports two multiprocessor communication modes when the asynchronous mode is used. These formats can be used to transfer information between many microcomputers on the same serial link. Information is transferred as a block of frames from a particular source to one or more destinations. The USART has features to identify the start of blocks, and to suppress interrupts and status information from the receiver, until a block start is identified. In both multiprocessor modes, the sequence of data exchange with the USART module could be based on polling of data, or using the receive interrupt features.

Both asynchronous multiprocessor protocols, the idle line and the address bit multiprocessor mode allow efficient data transfer between multiple communication systems. They also can be used to minimize activity of the system, whether to save current consumption or processing resources. The MM bit in the control register defines the address bit or idle line multiprocessor protocol mode. Both formats use the wake up on transmitting, using the address feature function (TXWake bit), and on activating the RXWake bit. The URXWIE and URXIE bits control the transmit and receive features of these modes.

12.1.4 Idle line multiprocessor mode

In this mode, blocks of data are separated by an idle time between them. An idle receive line is detected when 10 or more 1s in a row are received after the first stop bit of a character.

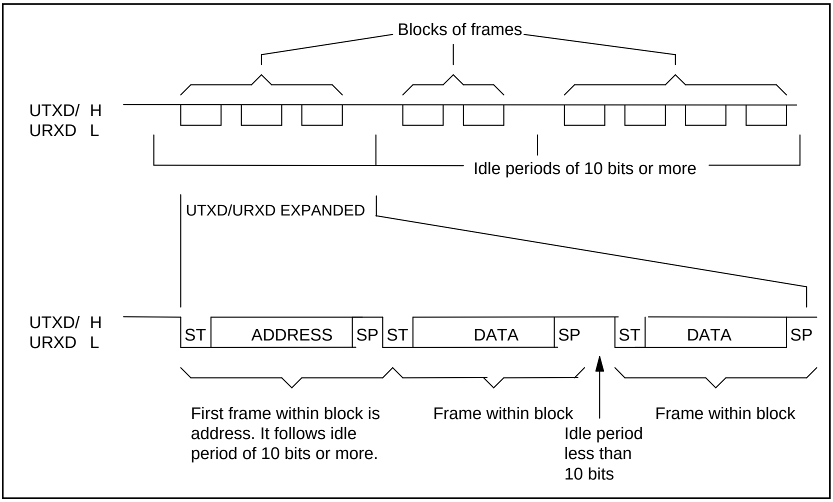

Blocks of frames
UTXD/ H
URXD L
Idle periods of 10 bits or more
UTXD/URXD EXPANDED
UTXD/ H
URXD L
ST
ADDRESS
SP
ST
DATA
SP
ST
DATA
SP
First frame within block is address. It follows idle period of 10 bits or more.
Frame within block
Idle period less than 10 bits
Frame within block

Figure 12.6: Idle line multiprocessor protocol

When two stop bits are used, the second one is counted as the first 'Mark' bit of the idle period. The first character received after an idle period is an address character. The RXWake bit can be used as an address tag for the character. In idle line multiprocessor format, RXWake bit is set when a received character is an address character and is transferred into the receive buffer.

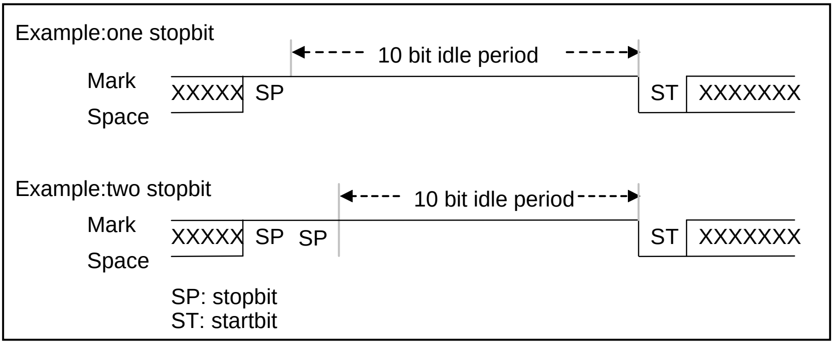


Figure 12.7: USART Receiver Idle Detect

Normally, if the USART’s URXWIE bit in the receive control register is set, characters will be assembled as usual by the receiver, but they will not be transferred to the receiver buffer, URXBUF, nor will interrupts be generated. When an address character is received, the receiver is temporarily activated to transfer the character to URXBUF and set the URXIFG interrupt flag. Appropriate error status flags will be set. The application software can validate the received address. If there is a match, the application software will handle the further data processing and execute proper operation. If not, the processor waits for the next address character to arrive. The URXWIE bit itself is not modified by the USART: it should be modified by the user in order to receive non-address characters or address characters.

In idle line multiprocessor mode, a precise idle period can be generated to create efficient address character identifiers. Associated with the TXWake bit is the wake-up temporary (WUT) flag. WUT is an internal flag, double buffered with TXWake. When the transmitter is loaded from UTXBUF, WUT is loaded from TXWake, and TXWake bit is reset.

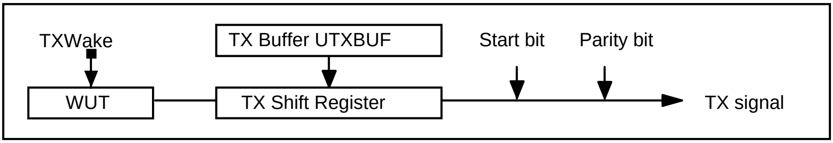


Figure 12.8: Double-Buffered WUT and TX Shift Register

Sending out an idle frame to identify an address character is accomplished as follows:

The TXWake bit should be set, and then any word (don't care) must be written to the UTXBUF (UTXIFG should be set). When the transmitter shift register is empty (TXEPT is set), the contents of the UTXBUF are shifted to the transmit shift register, and the TXWake value is shifted to WUT. When the WUT bit has been set, the start, data, and parity bits will be suppressed and an idle period of exactly 11 bits will be transmitted. The next data word, shifted out of the serial port after the address character identifying idle period, will be the second word written to the UTXBUF after TXWake bit was set. The first data word written is suppressed while the address identifier is sent out, and ignored after that. Writing the first don't care word to UTXBUF is necessary so that the TXWake bit value can be shifted to WUT.

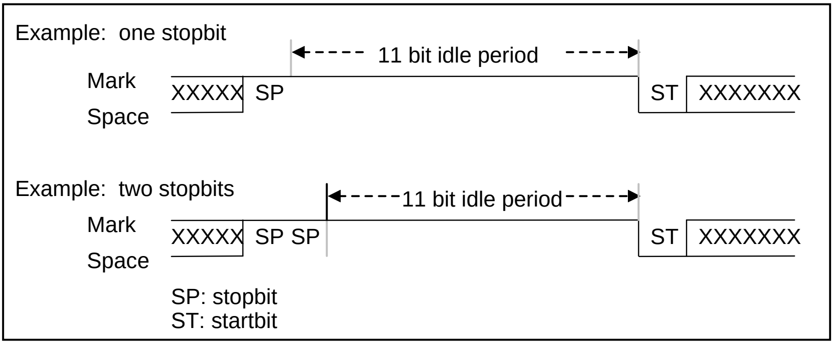


Figure 12.9: USART Transmitter Idle Generation

12.1.5 Address bit Format

In this mode, characters contain an extra bit that is used as an address indicator. The first character in a block of data carries an address bit that is set to indicate that the character is an address. The RXWake bit is set when a received character is an address character, and is transferred into the receive buffer (receive conditions are true).

Normally, if the USART's URXWIE bit is set, data characters will be assembled as usual by the receiver, but they will not be transferred to the receiver buffer URXBUF nor will interrupts be generated. When a character is received that has an address bit set, the receiver is temporarily activated to transfer the character to URXBUF and set the URXIFG. Error status flags will be set as appropriate. The application SW handles the succeeding operation for the best benefit in processing resource handling or current consumption reduction. The application software can validate the received address. If there is a match, the processor can read the remainder of the data block. If not, the processor waits for the next address character to arrive.

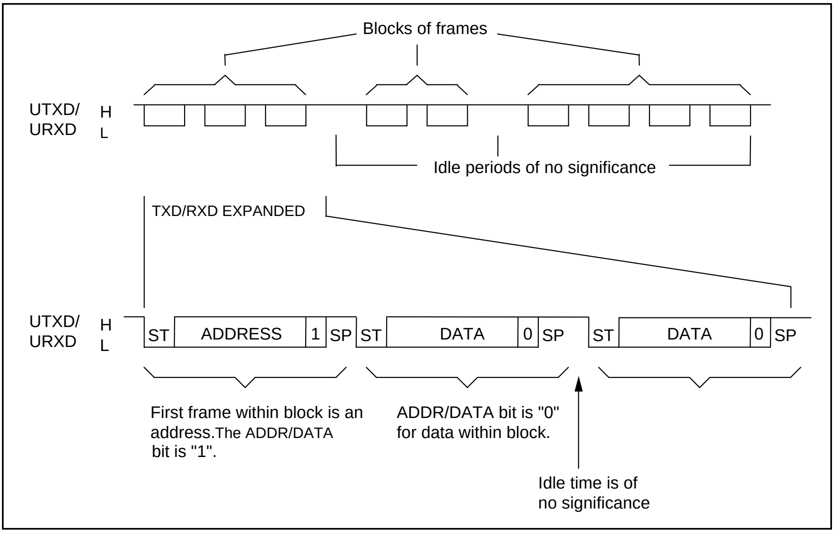

Blocks of frames
UTXD/
URXD
H
L
Idle periods of no significance
TXD/RXD EXPANDED
UTXD/
URXD
H
L
ST
ADDRESS
1
SP
ST
DATA
0
SP
ST
DATA
0
SP
First frame within block is an address.The ADDR/DATA bit is "1".
ADDR/DATA bit is "0" for data within block.
Idle time is of no significance

Figure 12.10: Address bit multiprocessor protocol

In address bit multiprocessor mode, the address bit of a character can be controlled by writing to the TXWake bit. The value of the TXWake bit is loaded into the address bit of that character each time a character is transferred from the transmit buffer UTXBUF to the transmitter. The TXWake bit is then cleared by the USART.

12.2 Interrupt and Control Function

The USART peripheral serves two main interrupt sources, for transmission and reception. Two individual interrupt vectors are available, one for receive and one for transmit interrupt events.

The control bits of the USART are located in the SFR address range:

• Receive Interrupt Flag	URXIFG	initial state reset (by PUC/SWRST)
• Receive Interrupt Enable	URXIE	initial state reset (by PUC/SWRST)
• Receive Enable	URXE	initial state reset (by PUC)
• Transmit Interrupt Flag	UTXIFG	initial state set (by PUC/SWRST)
• Transmit Interrupt Enable	UTXIE	initial state reset (by PUC/SWRST)
• Transmit Enable	UTXE	initial state reset (by PUC)

The receiver and transmitter of the USART operate fully independently, but use the same baud rate generator. Transmit and receive use the same baud rate.

12.2.1 USART Receive Enable

The Receiver Enable bit URXE enables or disables the receiver from collecting the bit stream on the URXD data line. Disabling the USART receiver will stop the receive operation after completing a receive operation which has been started, or stop immediately if no receive operation is active. The start bit detection is disabled.

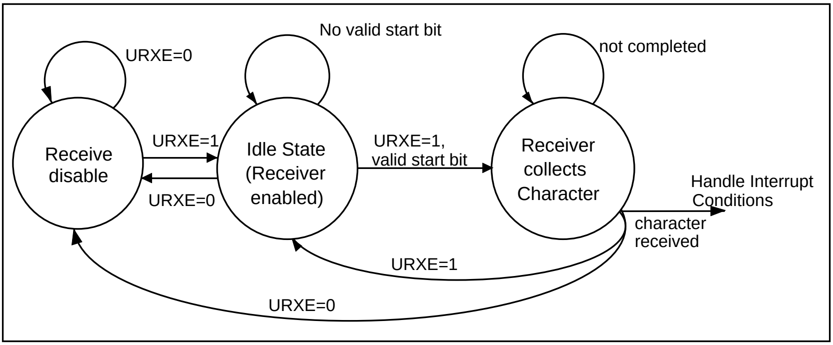


Figure 12.11: State diagram on Receiver enable URXE

Note: URXE re-enable, UART Mode

Since the receiver is completely disabled a re-enable of the receiver is asynchronous to any data stream on the communication line. Synchronization can be done by looking for an idle line condition before accepting any received character.

12.2.2 USART Transmit Enable

The transmit enable bit UTXE enables or disables a character transmission on the serial data line. If this bit is reset, the transmitter is disabled but any active transmission is not halted until all data previously written into the transmit buffer has been sent. If the transmission is completed, any further write to the transmitter buffer will not result in a data transmission.

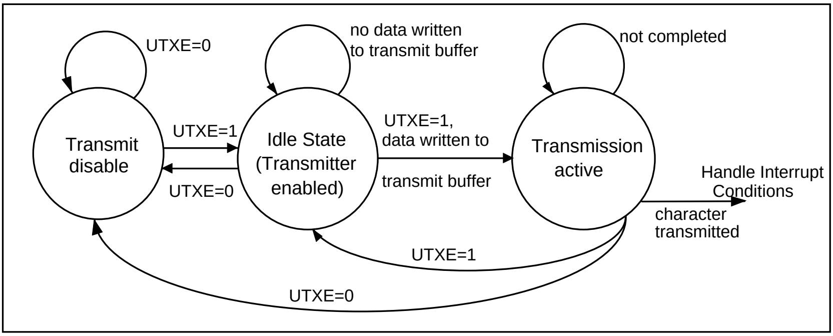


Figure 12.12: State diagram on Transmitter enable

When UTXE is reset any data can be written regularly into the transmit buffer, but no transmission is started. Once the UTXE bit is set, an immediate start of transmission of the character presently in the buffer is initiated. This character is transmitted correctly.

Note: Write to UTXBUF, UART Mode

Data should never be written into the transmit buffer UTXBUF when it is not ready and the transmitter is enabled (UTXE is set). If it is, the character shifted out can be random.

12.2.3 USART Receive Interrupt Operation

The receive interrupt flag URXIFG is set or is unchanged each time a character is received and loaded into the receive buffer:

- Erroneous characters (parity, frame or break error) will not set interrupt flag URXIFG when URXEIE is reset: URXIFG is unchanged.
- All type of characters (URXWIE=0) or only address characters (URXWIE=1) will set the interrupt flag URXIFG pending on the bit URXWIE. When URXEIE is also set, erroneous character will set the interrupt flag URXIFG.

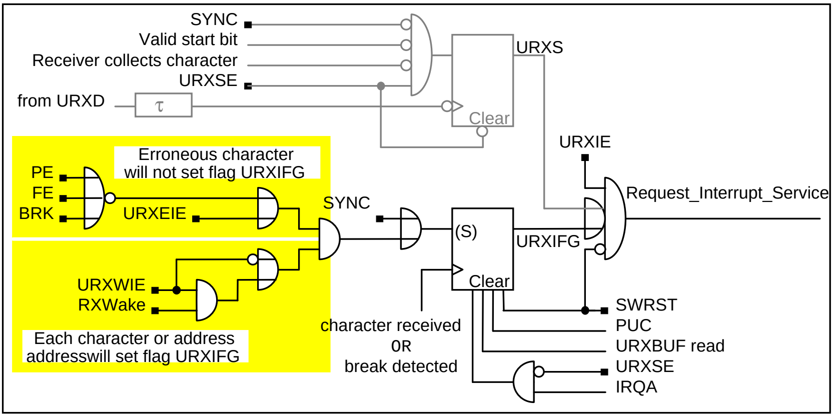

SYNC
Valid start bit
Receiver collects character
URXSE
from URXD
τ
URXS
Clear
URXIE
Erroneous character will not set flag URXIFG
PE
FE
BRK
URXEIE
SYNC
Request_Interrupt_Service
(S)
URXIFG
Clear
URXWIE
RXWake
Each character or address addresswill set flag URXIFG
character received 0R break detected
SWRST
PUC
URXBUF read
URXSE
IRQA

Figure 12.13: Receive Interrupt Conditions

URXIFG is reset at system reset PUC, or at a software reset SWRST. URXIFG is reset automatically if the interrupt is served (URXSE=0) or the receive buffer URXBUF is read. The Receive Interrupt Flag URXIFG indicates, if set, an interrupt event waiting to be served. The Receive Interrupt Enable bit URXIE enables, if set, serving of a waiting interrupt request. Both the receive interrupt flag URXIFG and the receive interrupt enable bit URXIE are reset with PUC and SWRST.

The signal URXIFG can be accessed by software. Signal URXS can not be accessed by software. When both interrupt events - receive start detection and character receive action - are enabled by software, the flag URXIFG indicates that a character was received and not the start detect request interrupt service. This works, since the interrupt software handler for the receive start detection will reset the URXSE bit. This clears the URXS bit and prevents further interrupt requests from URXS. The URXIFG should be already reset since no set condition was at this time at URXIFG latch.

12.2.4 USART Transmit Interrupt Operation

The transmit interrupt flag UTXIFG is set by the transmitter to indicate that the transmitter buffer UTXBUF is ready to accept another character. This bit is automatically reset if the interrupt request service is started or a character is written into the UTXBUF. This flag will assert a transmitter interrupt if the local (UTXIE) and general (GIE) interrupt enable bit are set. The UTXIFG is set after system reset PUC or SWRST are removed.

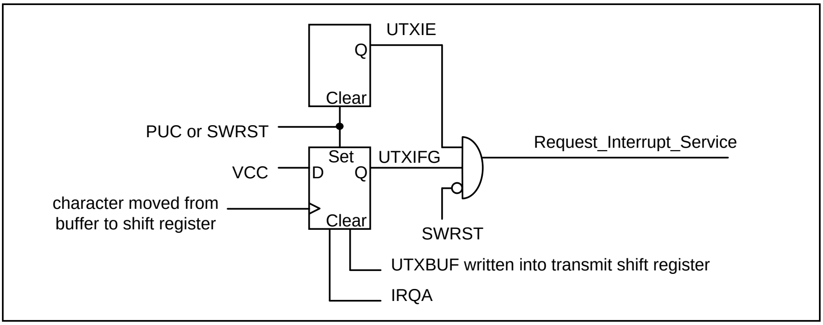


Figure 12.14: Transmit Interrupt Condition

The transmit interrupt enable UTXIE bit controls the ability of the UTXIFG to request an interrupt but does not prevent the flag UTXIFG from being set. The UTXIE is reset with PUC or software reset bit SWRST. The UTXIFG bit is set after system reset PUC or software reset SWRST, but the UTXIE bit is reset to ensure full interrupt control capability.

12.3 Control and Status Register

The USART module hardware is byte structured and should be accessed by byte processing instructions (suffix 'B').

Register	short form	Register type	Address	Initial state
● USART Control register	UCTL	Type of read/write	070h	See ....
● Transmit Control register	UTCTL	Type of read/write	071h	individual ...
● Receive Control register	URCTL	Type of read/write	072h	bit description
● Modulation Control reg.	UMCTL	Type of read/write	073h	unchanged
● Baud Rate register 0	UBR0	Type of read/write	074h	unchanged
● Baud Rate register 1	UBR1	Type of read/write	075h	unchanged
● Receive Buffer	URXBUF	Type of read/write	076h	unchanged
● Transmit Buffer	UTXBUF	Type of read	077h	unchanged

All bits are random after PUC, unless noted otherwise by the detailed functional description.
Reset of the USART is performed by PUC or SWRST bit. After power-up clear (PUC) the SWRST bit remains set and the USART remains in this condition until the reset is disabled by resetting the SWRST bit.
The USART module operates in asynchronous or in synchronous mode defined by the SYNC bit. The bits in the control registers may have different functions in the two modes. All bits in this section are described with their functions in the asynchronous mode - SYNC=0. Their functions in the synchronous mode are described in the USART's serial peripheral interface section.

12.3.1 USART Control register UCTL

The information stored in the control register determines the basic operation of the USART module. The register bits select the communications protocol, communication mode and parity bit. All bits should be programmed according to the selected mode before reset is disabled by resetting bit SWRST.

	7							0
UCTL 070h	PENA	PEV	SP	CHAR	Listen	SYNC	MM	SWRST
	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

Figure 12.15: USART Control Register UCTL

Bit 0: The USART state machines and operating flags are initialized to the reset condition if the software reset bit is set. Until the SWRST bit is reset, all affected logic is held in the reset state. This implies that after a system reset the USART must be re-enabled by resetting this bit. The receive and transmit enable flags URXE and UTXE are not altered by SWRST.

Bit 1: Multiprocessor mode (address/idle line wake up).
Two multiprocessor protocols, idle line and address bit, are supported by the USART module. The choice of multiprocessor mode affects the operation of the automatic address decoding functions.
MM = "0" : Idle line multiprocessor protocol
MM = "1" : Address bit multiprocessor protocol
The conventional asynchronous protocol uses MM bit reset

Bit 2: Mode or function of USART module selected.
The SYNC bit selects the function of the USART peripheral interface module. Some of the USART control bits will have different functions in UART and SPI mode.
SYNC = 0 : UART function is selected.
SYNC = 1 : SPI function is selected.

Bit 3: The Listen bit selects if the transmitted data is fed back internally to the receiver.
Listen = 0 : No feed back.
Listen = 1 : Transmit signal is internally fed back to the receiver. Each transmission from the MSP430's USART is received parallel and no external signal is received anymore.

Bit 4: Character length.
This register bit selects the length of the character to be transmitted as 7 or 8 bits. Characters of 7 bits do not use the eighth bit in URXBUF and UTXBUF and this bit is padded with "0".
CHAR = 0 : 7 bit data.
CHAR = 1 : 8 bit data.

Bit 5: Number of stop bits.
This bit determines the number of stop bits transmitted. The receiver checks for one stop bit only.
SP = 0 : one stop bit.
SP = 1 : two stop bits.

Bit 6: Parity odd/even.
If PENA bit is set (parity bit is enabled), the PEV bit defines odd or even parity according to the number of odd or even "1" bits in both transmitted and received characters, address bit (address bit multiprocessor mode) and parity bit.
PEV = 0 : Odd parity
PEV = 1 : Even parity.

Bit 7: Parity enable.
If parity is disabled no parity bit is generated during transmission or expected during reception. A received parity bit is not transferred to the URXBUF with the received data as it is not considered as one of the data bits. During address bit multiprocessor mode, the address bit is included in the parity calculation.
PEN = 0 : Parity disable
PEN = 1 : Parity enable

Note: MARK, SPACE definition

The MARK condition is identically to the signal level in the idle state. SPACE is the opposite signal level: the start bit is always SPACE.

12.3.2 Transmit Control Register UTCTL

The register UTCTL controls the USART hardware associated with transmit operation.

UTCTL
071h

7							0
unused	CKPL	SSEL1	SSEL0	URXSE	TXWake	unused	TXEPT
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

Figure 12.16: USART Transmitter Control Register

Bit 0: The transmitter empty TXEPT flag is set when the transmitter shift register and UTXBUF are empty, and reset when data is written to UTXBUF. It is set on SWRST.

Bit 1: unused

Bit 2: The TXWake bit is used to control the transmit features of the multiprocessor communication modes. Each transmission - started by loading the UTXBUF - uses the state of the TXWake bit to initialize the feature of address identification. It should not be cleared - the USART hardware clears this bit once it has been transferred to "Wake Up Temporary", WUT; SWRST also clears TXWake bit.

Bit 3: The receive start edge control bit requests - if set - a receive interrupt service. For a successful interrupt service the corresponding enable bits URXIE and GIE should be set. The advantage of this bit is to start the controller's clock system including MCLK along with the interrupt service, and keep it running by modifying the mode control bits. The USART is working with selected MCLK properly, even if the system is switched to a low power mode with disabled MCLK.

Bit 4,5: Source Select 0 and 1.
The source select bit defines which clock source is used for the baud rate generation:

SSEL1,SSEL0	0	external clock selected, UCLKI
	1	auxiliary clock selected, ACLK
	2, 3	main system clock selected, MCLK

Bit 6: Clock polarity CKPL.
The CKPL bit controls the polarity of the UCLKI signal.
CKPL = 0: the UCLKI signal has same polarity than UCLK signal.
CKPL = 1: the UCLKI signal has inverted polarity of UCLK signal.

Bit 7: Unused

12.3.3 Receive Control Register URCTL

The register URCTL controls the USART hardware associated with the receiver operation and holds error and wakeup conditions modified by the latest character written to the receive buffer URXBUF. Once any of the bits FE, PE, OE, BRK, RXERR or RXWake is set, they are not reset by receiving another character. They are reset by accessing the receive buffer URXBUF, by a USART SW reset SWRST, a system reset PUC or by instruction.

	7							0
URCTL 072h	FE	PE	OE	BRK	URXEIE	URXWIE	RXWake	RXERR
	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Figure 12.17: USART Receiver Control Register

Bit 0: The receive error bit RXERR indicates that one or more error flags (FE, PE, OE or BRK) are set. It is not reset when the error bits are cleared by instruction.

Bit 1: Receiver Wake-up Detect

RXWake bit is set when a received character is an address character and is transferred into the receive buffer.

Address bit multiprocessor mode: RXWake is set when the address bit is set in the character received.

Idle line multiprocessor mode: RXWake is set if an idle URXD line was detected (11 bits of Mark level) in front of the received character.

RXWake is reset by accessing the receive buffer URXBUF, by a USART SW reset SWRST or a system reset PUC.

Bit 2: The receive wake-up interrupt enable bit URXWIE selects the type of character that will set the interrupt flag URXIFG:

URXWIE=0: each character received will set the URXIFG

URXWIE=1: only characters that are marked as address characters will set the interrupt flag URXIFG. It operates identically in both multiprocessor modes.

The wake-up interrupt enable feature depends on the receive erroneous character feature. See also URXEIE bit.

Bit 3: The receive erroneous character interrupt enable bit URXEIE selects if an erroneous character will set the interrupt flag URXIFG.

URXEIE=0: each erroneous character received will not alter the interrupt flag URXIFG

URXEIE=1: all characters can set the interrupt flag URXIFG depending on the conditions set by URXWIE bit.

URXEIE	URXWIE	Char. w/ Error	Char. address	Description Flag URXIFG after a character was received
0	x	1	x	unchanged
0	0	0	x	set
0	1	0	0	unchanged
0	1	0	1	set
1	0	x	x	set (will receive all characters)
1	1	x	0	unchanged
1	1	x	1	set

Bit 4: The break detect bit BRK is set when a break condition occurs and URXEIE bit is set. The break condition is recognized if the RXD line remains continuously low for at least 10 bits, beginning after a missing first stop bit. It is not cleared by receipt of a character after the break is detected - but reset by SWRST, system reset, and by reading the URXBUF.

Bit 5: The overrun error flag bit OE is set when a character is transferred into the URXBUF before the previous character has been read out. The previous character is overwritten and lost. OE is reset by SWRST, system reset, and by reading the URXBUF.

Bit 6: The parity error bit PE is set when a character is received with a mismatch between the number of "1's" and its parity bit and is loaded into the receive buffer. The parity checker includes the address bit - used with the address bit multiprocessor mode - in the calculation. The flag is disabled if parity generation and detection is not enabled. In such a case, it is read as "0". It is reset by SWRST, system reset, and by reading the URXBUF.

Bit 7: The framing error flag bit FE is set when a character is received with a "0" stop bit and is loaded into the receive buffer. Only the first stop bit is checked when more than one is used. The missing stop bit indicates that synchronization with the start bit has been lost and the character is incorrectly framed. FE is reset by SWRST, system reset, and reading URXBUF.

Note: Receive Status Control bits

The receive status control bits FE, PE, OE, BRK and RXWake are set conditionally by the hardware according to the conditions of the characters received. Once bits are set they remain set until the software will reset them directly or by reading the receive buffer. False character interpretation or missing interrupt capability can be the result of non-cleared error bits.

12.3.4 Baud Rate Select and Modulation Control Registers

The baud rate generator uses the content of both baud rate select registers UBR1 and UBR0 together with the modulation control register to generate the bit timing for the serial data stream.

UBR0 074h	7							0
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	rw	rw	rw	rw	rw	rw	rw	rw

UBR1 075h	7							0
	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8
	rw	rw	rw	rw	rw	rw	rw	rw

Figure 12.18: USART Baud Rate Select Register

$$\text{Baudrate} = \frac{\text{BRCLK}}{\text{UBR} + \frac{1}{n}\sum_{i=0}^{n-1} mi} \qquad \text{with UBR= [UBR1,UBR0]}$$

The baud rate control register range is: $3 \leq \text{UBR} < \text{0FFFFh}$

The modulation control register ensures a proper timing generation together with UBR0/1, even with crystal frequencies that are not integer multiples of the required baud rate.

UMCTL 073h	7							0
	m7	m6	m5	m4	m3	m2	m1	m0
	rw	rw	rw	rw	rw	rw	rw	rw

Figure 12.19: USART Modulation Control Register

The timing of the running bit is expanded by one clock cycle of the input clock of the baud rate divider if the actual bit m_i is set.

Each time a bit is received or transmitted the next bit in the modulation control register is used to determine the present bit timing. The first bit time in the protocol - the start bit time - is determined by UBR plus m0; the next bit by UBR plus m1,...
The modulation sequence is:

m0 - m1 - m2 - m3 - m4 - m5 - m6 - m7 - m0 - m1 - m2 - .....

12.3.5 USART Receiver Data Buffer URXBUF

The receiver buffer URXBUF contains previous data from the receiver shift register. Reading URXBUF resets the receive error bits, RXWake bit and interrupt flag URXIFG.

	7							0
URXBUF 076h	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	r	r	r	r	r	r	r	r

Figure 12.20: USART Receive Buffer

In 7-bit length mode, the MSB of the URXBUF is always reset.

The receive buffer is loaded with the recently received character when receive and control conditions are true:

URXEIE	URXWIE	Load URXBUF by	PE	FE	BRK
0	1	error-free address characters	0	0	0
1	1	all address characters	x	x	x
0	0	error-free characters	0	0	0
1	0	all characters	x	x	x

12.3.6 USART Transmit Data Buffer UTXBUF

The transmit buffer contains current data to be transmitted by the transmitter.

	7							0
TXBUF 077h	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	rw	rw	rw	rw	rw	rw	rw	rw

Figure 12.21: USART Transmit Buffer

The UTXIFG flag indicates that UTXBUF is ready to accept another character for transmission.
The transmission will be initialized by writing data to UTXBUF. The transmission of this data is started immediately, if the transmitter shift register is empty or is going to be empty.
Writing data to the transmit buffer should be done only if the buffer UTXBUF is empty, otherwise an unpredictable character can be transmitted.

12.4 UART Mode, Utilizing Features of low power Modes

There are several functions or operational features implemented that support the basic ultra-low power system of the MSP430 architecture:

- System start from any processor mode through sensing of UART frame start condition
- Use lowest input clock frequency for required baud rate
- Support of multiprocessor modes for reduced use of MSP430 resources.

12.4.1 Start Receive Operation from UART Frame

The most effective use of the start detection in the receive path is reached when the baudrate requires to run the system main clock MCLK, but the entire system can operate without MCLK. The receive start condition is the negative edge from the signal at URXD pin. Each time when it triggers the interrupt flag URXS, it requests a service when URXIE and GIE enable bits are set. The MSP430 system returns to the active mode and full system performance with MCLK and ACLK active.

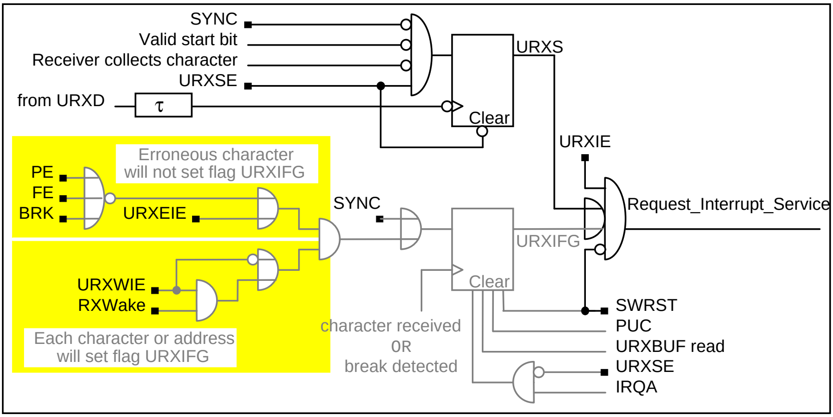

SYNC
Valid start bit
Receiver collects character
URXSE
from URXD
τ
URXS
Clear
URXIE
Erroneous character will not set flag URXIFG
PE
FE
BRK
URXEIE
SYNC
Request_Interrupt_Service
URXIFG
Clear
URXWIE
RXWake
Each character or address will set flag URXIFG
character received 0R break detected
SWRST
PUC
URXBUF read
URXSE
IRQA

Figure 12.22: Receive Start Conditions

Three character streams will not set the interrupt flag URXIFG:

- erroneous characters (URXEIE=0)
- address characters (URXWIE=1)
- and invalid start bit detect.

The interrupt software should handle these conditions. The interrupt handler must configure the correct clock system condition and the clock system will continue operation - and current consumption - until it is modified by the software. Whenever the CPU operates in the active mode the clock system is operating normally and start condition detection should not be used.

Start conditions

The URXD signal feed into the USART module is going first into a deglitch circuit. Glitches can not trigger the receive start condition flag URXS. This prevents the module from being started from small glitches on the URXD line. In noisy environments the current consumption is reduced, since glitches does not start system and USART.

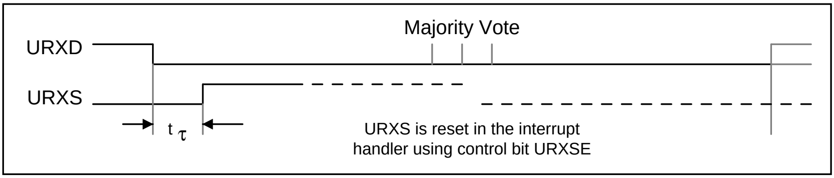

Majority Vote
URXD
URXS
t τ
URXS is reset in the interrupt handler using control bit URXSE

Figure 12.23: Receive Start Timing using URXS flag, start bit accepted

The UART stops receiving a character when the URXD signal exceeds the deglitch time t_τ but then the majority vote of the signal fails to start bit detection. The software should handle this condition and hold the system in the appropriate low power mode. The interrupt flag URXIFG is not set.

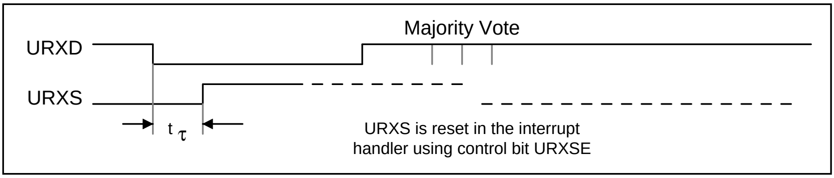

Majority Vote
URXD
URXS
t τ
URXS is reset in the interrupt handler using control bit URXSE

Figure 12.24: Receive Start Timing using URXS flag, start bit not accepted

Glitches at the URXD line are suppressed automatically and no further activity in the MSP430 is started. The data for the deglitch time t_τ is noted in the corresponding device specification.

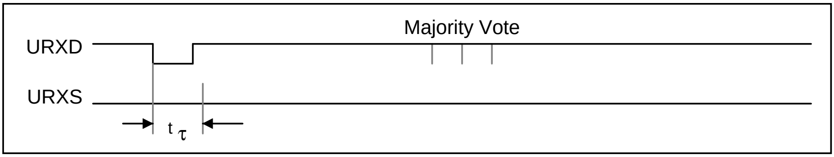


Figure 12.25: Receive Start Timing using URXS flag, glitch suppression

The interrupt handler should reset the URXSE bit in the control register UCTL to prevent further interrupt service requests from URXS signal and to enable the basic function of receive interrupt flag URXIFG.

```
*********************************************************************
*      INTERRUPT HANDLER FOR FRAME START CONDITION AND              *
*      CHARACTER RECEIVE                                            *
*********************************************************************
IFG2      .EQU      3               ; URXIFG AND UTXIFG IN ADDRESS 3
UTCTL     .EQU      71H             ;
UTXIFG    .EQU      0               ;
URXSE     .EQU      8               ;
                                    ;

URX_INT   BIT.B     #URXIFG,&IFG2   ; TEST URXIFG SIGNAL TO CHAECK
          JNE       ST_COND         ; IF FRAME START CONDITION
          .....
          .....
ST_COND   BIC.B     #URXSE,&UTCTL   ; CLEAR FF/SIGNAL URXS, STOP
                                    ; FURTHER INTERRUPT REQUESTS
          BIS.B     #URXSE,&UTCTL   ; PREPARE FF_URXS FOR NEXT FRAME
          .....                     ;.START CONDITION
          .....                     ; AND SET THE CONDITIONS TO RUN
          .....                     ; THE CLOCK NEEDED FOR UART RX
```

Note: Break detect BRK bit with halted UART clock

If the UART is operating with the feature of wake-up with a start condition, and to switch off the UCLK whenever a character is completely received, the break of the communication line can not be detected automatically by the UART hardware. The break detect needs the clock BRSCLK out of the baud rate generator to detect this conditions, but it is stopped upon the missing UCLK.

12.4.2 Maximum Utilization of Clock Frequency vs. Baud Rate UART Mode

The current consumption depends linearly on the clock frequency. It should be kept to the minimum required to meet the application conditions. Fast communication speed is needed due to various reason - calibration and test in manufacturing processes, alarm situations in critical applications, response time to human requests for information,......
The baud rate generator in the MSP430 USART is realized to meet baud rates up to $^1/_3$ of the clock frequency. An additional modulation of the baud rate timing gives extra benefit since the timing for the single bit in a frame can be adjusted. The timing is adjusted from bit to bit to meet the requirements even when a non-integer division is needed. Baud rates can be done from a 32,768Hz crystal up to 4800 Baud with errors of max. 11%. Standard UART’s can - with the worse maximum error (-14.6%) reach maximum baud rates of 75 Baud.

12.4.3 Support of multiprocessor modes for reduced use of MSP430 resources

Communication systems with multiple character protocols can use the features of multiprocessor modes - whether the idle line or the address bit protocol. The first character can be a target address, a message identifier or can have another definition. This character is interpreted by software, and if there is any significance for the application the succeeding characters are collected and further activities defined. No significance of the first character would stop any activity for the processing device. The application of this feature is supported by the wake-up interrupt feature in receive situation, and to send wake-up conditions along with transmission. Avoiding activity on characters without any significance reduces the use of MSP430 resources and the system can remain in the most efficient power conserving mode.
Additional to the multiprocessor modes, rejection of erroneous characters avoids interrupt handling of these characters. This is useful whenever erroneous characters will not be processed anyway. The processor waits in the most efficient power conserving mode until a character can be processed.

12.5 Baud Rate Considerations

The baud rate generator of the MSP430 uses one divider and a modulator. A given crystal’s frequency and a required baud rate will determine the needed division factor N:

N = **Error!**

The necessary division factor N usually has an integer part and a fraction. The divider in the baudrate generator realizes the integer portion of the division factor N and the modulator is responsible for meeting the fractional part as close as possible. The factor N is defined:

$$N = UBR + \frac{1}{n} \sum_{i=0}^{n-1} m_i$$

where N is the target division factor
UBR is the 16-bit representative of register UBR1 and UBR0
i is the actual bit in the frame
n is the number bits in the frame
m_i is the data of the actual modulation bit.

$$\text{Baudrate} = \frac{BRCLK}{N} = \frac{BRCLK}{UBR + \frac{1}{n} \sum_{i=0}^{n-1} mi}$$

Bit Timing in Transmit Operation

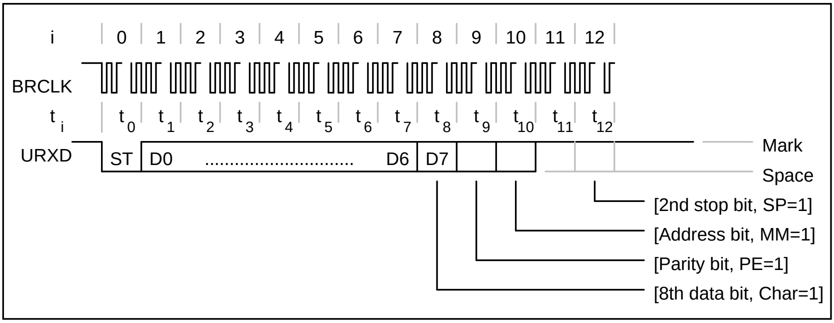


Figure 12.26: MSP430 Transmit Bit Timing

The timing for each individual bit in one frame or character is the sum of the actual bit timings. The error of the baud rate generation in respect to the required ideal timing is calculated for each individual bit. The relevant information is the error relative to the actual bit, not the overall relative error.

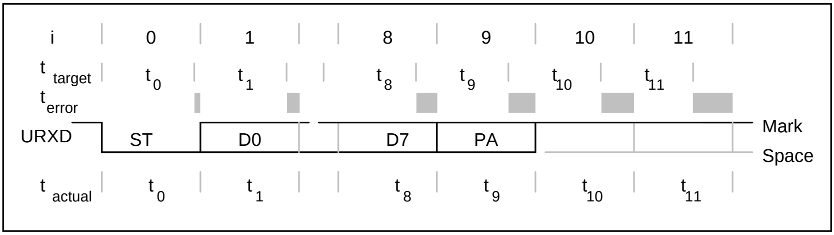


Figure 12.27: MSP430 Transmit Bit Timing Errors

Even small errors per bit (relative errors) end up in larger errors - they should be considered to be accumulative, not relative. The error of an individual bit can be calculated by:

$$\text{Error [\%]} = \frac{\sum_{i=0}^{n-1} t_{actual_i} - \sum_{i=0}^{n-1} t_{t\,arg\,et_i}}{t_{baud\ rate}} \times 100\%$$

OR

$$\text{Error [\%]} = ((\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } ((i+1) \text{ x UBR } + \sum_{i=0}^{n-1} m_i) - (i+1)) \text{ x } 100\%$$

with baud rate is the required baud rate
BRCLK is the input frequency - selected for UCLK, ACLK or MClK
i=0 for the start bit, 1 for data bit D0, ...
UBR is division factor in registers UBR1 and UBR0

Example 1

The following data are assumed:

baud rate =	2400 Baud
BRCLK =	32,768Hz (ACLK)
UBR =	13, since the ideal division factor should be 13.67
m = 6Bh:	m7=0, m6=1, m5=1, m4=0, m3=1, m2=0, m1=1 and m0=1 The LSB (m0) of the modulation register is used first.

Start bit $\text{Error [\%]} = (\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } ((0+1) \text{ x UBR } + 1) - 1) \text{ x } 100\% = 2.54\ \%$

Data bit D0 $\text{Error [\%]} = (\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } ((1+1) \text{ x UBR } + 2) - 2) \text{ x } 100\% = 5.08\ \%$

Data bit D1 $\text{Error [\%]} = (\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } ((2+1) \text{ x UBR } + 2) - 3) \text{ x } 100\% = 0.29\ \%$

Data bit D2 $\text{Error [\%]} = (\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } ((3+1) \text{ x UBR } + 3) - 4) \text{ x } 100\% = 2.83\ \%$

Data bit D3 $\text{Error [\%]} = (\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } ((4+1) \text{ x UBR } + 3) - 5) \text{ x } 100\% = -1.95\ \%$

Data bit D4 $\text{Error [\%]} = (\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } ((5+1) \text{ x UBR } + 4) - 6) \text{ x } 100\% = 0.59\ \%$

Data bit D5 $\text{Error [\%]} = (\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } ((6+1) \text{ x UBR } + 5) - 7) \text{ x } 100\% = 3.13\ \%$

Data bit D6 $\text{Error [\%]} = (\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } ((7+1) \text{ x UBR } + 5) - 8) \text{ x } 100\% = -1.66\ \%$

Data bit D7 $\text{Error [\%]} = (\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } ((8+1) \text{ x UBR } + 6) - 9) \text{ x } 100\% = 0.88\ \%$

Parity bit $\text{Error [\%]} = (\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } ((9+1) \text{ x UBR } + 7) - 10) \text{ x } 100\% = 3.42\ \%$

Stop bit 1 $\text{Error [\%]} = (\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } ((10+1) \text{ x UBR } + 7) - 11) \text{ x } 100\% = -1.37\ \%$

Stop bit 2 $\text{Error [\%]} = (\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } ((11+1) \text{ x UBR } + 8) - 12) \text{ x } 100\% = 1.17\ \%$

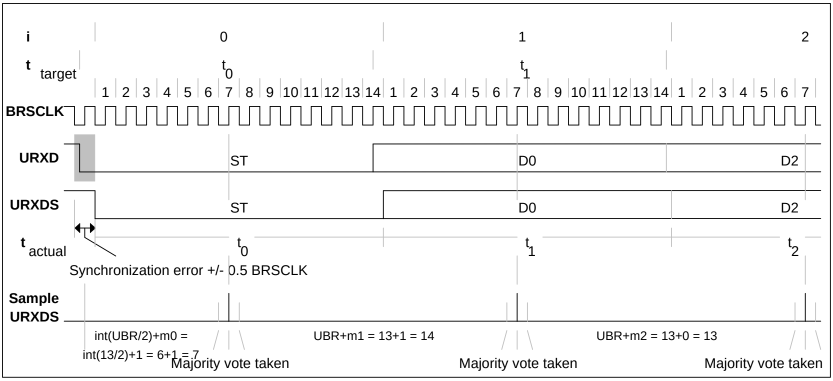
The standard baud rate data needed for the baud rate registers and the modulation register are listed for the watch crystal 32,768Hz (ACLK) and MCLK assumed to be 32-times the ACLK frequency. The error listed is calculated for the transmit and receive path. Additionally to this error for the receive situation, the synchronization error should also be considered.

Baud rate	Divide by		ACLK (32 768Hz)			max. TX error %	max. RX error %	Synchr. RX error %	MCLK (1 048 576Hz)			max. TX error %	max. RX error %
	ACLK	MCLK	UBR1	UBR0	UMOD				UBR1	UBR0	UMOD		
75	436.91	13981	1	B4	FF	-.1/.3	-.1/.3	+/-2	36	9D	FF	0/.1	+/-2
110	297.89	9532.51	1	29	FF	0/.5	0/.5	+/-3	25	3C	FF	0/.1	+/-3
150	218.45	6990.5	0	DA	55	0/.4	0/.4	+/-2	1B	4E	FF	0/.1	+/-2
300	109.23	3495.25	0	6D	22	-.3/.7	-.3/.7	+/-2	0D	A7	00	-.1/0	+/-2
600	54.61	1747.63	0	36	D5	-1/1	-1/1	+/-2	06	D3	FF	0/.3	+/-2
1200	27.31	873.81	0	1B	03	-4/3	-4/3	+/-2	03	69	FF	0/.3	+/-2
2400	13.65	436.91	0	0D	6B	6/3	-6/3	+/-4	01	B4	FF	0/.3	+/-2
4800	6.83	218.45	0	06	6F	-9/11	-9/11	+/-7	0	DA	55	0/.4	+/-2
9600	3.41	109.23	0	03	4A	-21/12	-21/12	+/-15	0	6D	03	-.4/1	+/-2
19 200		54.61							0	36	6B	-.2/2	+/-2
38 400		27.31							0	1B	03	-4/3	+/-2
76 800		13.65							0	0D	6B	-6/3	+/-4
115 200		9.10							0	09	08	-5/7	+/-7

Table 12.2: Mostly used Baud Rates, Baudrate data and errors

The synchronization error results from the asynchronous timing between the data signal at the URXD pin and the internal clock system. The receive signal is synchronized with the BRSCLK clock. The BRSCLK clock is sixteen to thirty-one times faster than the bit timing:

BRSCLK = BRCLK	for		N	≤ 1F
BRSCLK = BRCLK/2	for	20h	≤ N	≤ 3Fh
BRSCLK = BRCLK/4	for	40h	≤ N	≤ 7Fh
BRSCLK = BRCLK/8	for	80h	≤ N	≤ FFh
BRSCLK = BRCLK/16	for	100	≤ N	≤ 1FF
BRSCLK = BRCLK/32	for	200	≤ N	≤ 3FFh
BRSCLK = BRCLK/64	for	400	≤ N	≤ 7FFh
BRSCLK = BRCLK/128	for	800h	≤ N	≤ FFFh
BRSCLK = BRCLK/256	for	1000h	≤ N	≤ 1FFFh
BRSCLK = BRCLK/512	for	2000h	≤ N	≤ 3FFFh
BRSCLK = BRCLK/1024	for	4000h	≤ N	≤ 7FFFh
BRSCLK = BRCLK/2048	for	8000h	≤ N	≤ FFFFh



The target baud rate timing t_{target_0} for the start bit detection is half the baud rate timing $t_{baud\ rate}$ since the bit is tested in the middle of its period. The target baud rate timing t_{target_i} for the all other succeeding bits is the baud rate timing $t_{baud\ rate}$.

$$\text{Error [\%]} = \frac{t_{actual_0} + t_{target_0}}{0.5 \times t_{target_0}} + \frac{\sum_{i=1}^{n-1} t_{actual_i} - \sum_{i=1}^{n-1} t_{target_i}}{t_{target_i}} \times 100\%$$

OR

$$\text{Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times \{2 \times [m0 + \text{int}(UBR / 2)] + (i \times UBR + \sum_{i=1}^{n-1} m_i)\} - 1 - i\right) \times 100\%$$

where baud rate is the required baud rate
BRCLK is the input frequency - selected for UCLK, ACLK or MClK
i=0 for the start bit, 1 for data bit D0, ...
UBR is division factor in registers UBR1 and UBR0

Example 2

The following data are assumed:

baud rate =	2400 Baud
BRCLK =	32,768Hz (ACLK)
UBR =	13, since the ideal division factor should be 13.67
m = 6Bh:	m7=0, m6=1, m5=1, m4=0, m3=1, m2=0, m1=1 and m0=1 The LSB (m0) of the modulation register is used first.

$$\text{Start bit} \quad \text{Error [\%]} = \{\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } [2\text{x}(1+6)+(0\text{xUBR}+0-0)]-1\} \text{ x } 100\% = 2.54\ \%$$

$$\text{Data bit D0} \quad \text{Error [\%]} = \{\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } [2\text{x}(1+6)+(1\text{xUBR}+1)]-1-1\} \text{ x } 100\% = 5.08\ \%$$

$$\text{Data bit D1} \quad \text{Error [\%]} = \{\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } [2\text{x}(1+6)+(2\text{xUBR}+1)]-1-2\} \text{ x } 100\% = 0.29\ \%$$

$$\text{Data bit D2} \quad \text{Error [\%]} = \{\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } [2\text{x}(1+6)+(3\text{xUBR}+2)]-1-3\} \text{ x } 100\% = 2.83\ \%$$

$$\text{Data bit D3} \quad \text{Error [\%]} = \{\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } [2\text{x}(1+6)+(4\text{xUBR}+2)]-1-4\} \text{ x } 100\% = -1.95\ \%$$

$$\text{Data bit D4} \quad \text{Error [\%]} = \{\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } [2\text{x}(1+6)+(5\text{xUBR}+3)]-1-5\} \text{ x } 100\% = 0.59\ \%$$

$$\text{Data bit D5} \quad \text{Error [\%]} = \{\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } [2\text{x}(1+6)+(6\text{xUBR}+4)]-1-6\} \text{ x } 100\% = 3.13\ \%$$

$$\text{Data bit D6} \quad \text{Error [\%]} = \{\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } [2\text{x}(1+6)+(7\text{xUBR}+4)]-1-7\} \text{ x } 100\% = -1.66\ \%$$

$$\text{Data bit D7} \quad \text{Error [\%]} = \{\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } [2\text{x}(1+6)+(8\text{xUBR}+5)]-1-8\} \text{ x } 100\% = 0.88\ \%$$

$$\text{Parity bit} \quad \text{Error [\%]} = \{\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } [2\text{x}(1+6)+(9\text{xUBR}+6)]-1-9\} \text{ x } 100\% = 3.42\ \%$$

$$\text{Stop bit 1} \quad \text{Error [\%]} = \{\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } [2\text{x}(1+6)+(10\text{xUBR}+6)]-1-10\} \text{ x } 100\% = -1.37\ \%$$

$$\text{Stop bit 2} \quad \text{Error [\%]} = \{\frac{\text{baud rate}}{\text{BRCLK}} \text{ x } [2\text{x}(1+6)+(11\text{xUBR}+7)]-1-11\} \text{ x } 100\% = 1.17\ \%$$

Baud Rate Considerations - Conclusion

The system chosen to generate a proper serial communication bit stream allows baud rates up to nearly the clock rate fed into the USART. It enables low accumulating errors through modulation of the individual bit timing. In practice an error margin of 20% to 30% should make proper serial communication possible.

13 USART Peripheral Interface, SPI Mode

The synchronous interface is a serial channel which allows a serial bit stream of 7 or 8 bits to be shifted into and out of the MSP430, at an externally determined rate or at an internally programmed rate. The USART module is connected to the CPU as a byte peripheral. It connects the controller to the external system environment by three or four external pins.

USART's serial synchronous communication features:

- Control bit SYNC in control register UCTL is set to select synchronous mode
- Supports 3 pin and 4 pin SPI operation via SOMI, SIMO, UCLK and STE
- Select master or slave mode
- Separate shift registers for receive (URXBUF) and transmit (UTXBUF)
- Double buffering for receive and transmit
- Clock polarity and clock phase control
- Clock frequency control in master mode
- Character length 7 or 8 bits/character

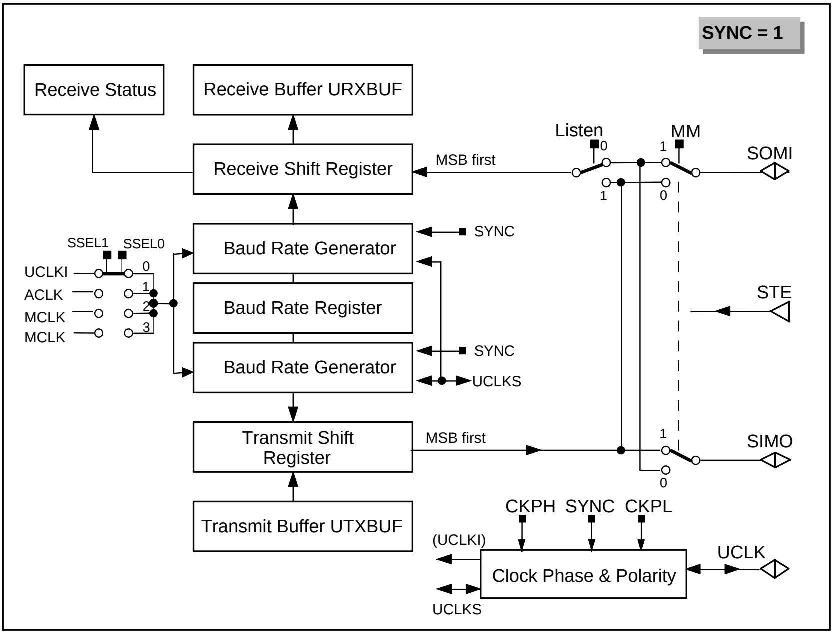


Figure 13.1: Block diagram of USART - SPI mode

13.1 USART's Synchronous Operation

In the synchronous mode, data and clock signals are used to transmit and receive serial data. The master supplies the clock and data. The slave(s) use this clock to shift the serial information in and out. The 4 pin SPI mode uses a control line additionally, to enable a slave to receive and transmit data. It is controlled by the master.

Three or four signals are used for the data exchange:

- SIMO Slave in, master out
- SOMI Slave out, master in
- UCLK USART clock, the master drives this signal and the slave uses it to receive and transmit data
- STE Slave transmit enable, used in 4-pin mode to control more than one slave in a multiple master and slave system.

The interconnection of the USART in synchronous mode to another device's serial port with one common transmit receive shift register is shown when MSP430 is master or slave. The operation will remain identical. The master initiates the transfer by sending the UCLK signal. For the master, data is shifted out of the transmit shift register on one clock edge and shifted into the receive shift register on the opposite edge. For the slave, the data shifting operation is the same, using one common register shift for transmitting and receiving data. Master and slave send and receive data at the same time.

Whether or not the data is meaningful or dummy data depends upon the application software:

1. Master sends data and Slave sends dummy data
2. Master sends data and Slave sends data
3. Master sends dummy data and Slave sends data.

The master can initiate data transfer at any time, and controls the UCLK. The software protocol determines the way in which the master knows when the slave wishes to broadcast data.

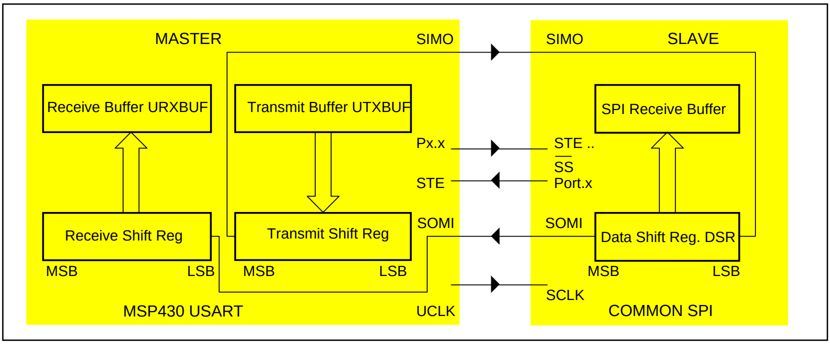
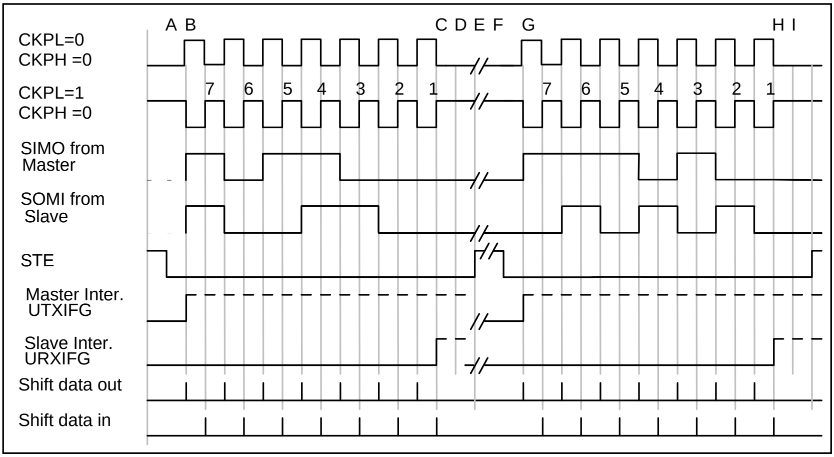
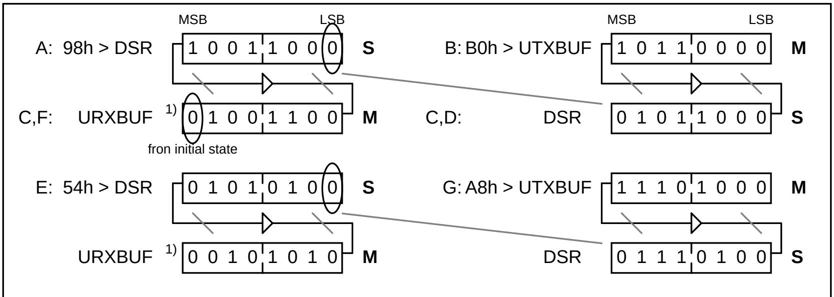


Figure 13.2: MSP430 USART as Master, external device with SPI as slave

There follows an example of serial synchronous data transfer for a character length of seven bits. The initial content of receive shift register is 00.



A: Slave writes 98h to the DSR and waits for the master to shift out data.
B: Master writes B0h to UTXBUF which is immediately transferred to the Transmit Shift Register and starts the transmission.
C: First character is finished and sets the interrupt flags.
D: Slave reads 58h from its receive buffer (right justified).
E: Slave writes 54h to its DSR and waits for the master to shift out data.
F: Master reads 4Ch from receive buffer URXBUF (right justified).
G: Master writes E8h to the transmit buffer UTXBUF and starts the transmission.
H: Second character is finished and sets the interrupt flag.
I: Master receives 2Ah and slave receives 74h (right justified).



[1] In 7bit Mode, the MSB of RXBUF is always read 0.

S: Slave **M:** Master

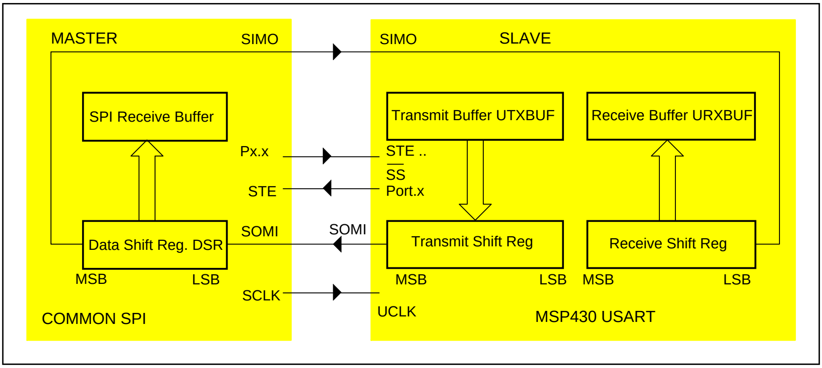


Figure 13.3: MSP430 USART as Slave in 3 pin or 4 pin configuration

13.1.1 Master Mode in Synchronous USART Mode, MM=1, SYNC=1

The master mode is selected when master mode bit MM in the control register UCTL is set. The USART controls the serial communication network by providing UCLK at the UCLK pin. Data is output on the SIMO pin on the first UCLK period and latched from the SOMI pin in the middle of the corresponding UCLK period.
The data written to the transmit buffer UTXBUF is moved to the transmit shift register as soon as it is empty and this initiates the data transfer on the SIMO pin, with the most significant bit first. At the same time, received data is shifted into the receive shift register, and upon completing of receiving the selected number of bits, the received data is transferred to the receive buffer URXBUF and the receive interrupt flag URXIFG is set. Data is shifted into the receive shift register, with the most significant bit first. It is stored right justified in receive buffer URXBUF. When previous data was not read from the receive buffer URXBUF the overrun error bit OE is set.

Note: USART Synchronous Master Mode, Receive initiation

The master should write data to the transmit buffer UTXBUF to receive a character. The receive starts when the transmit shift register is empty and the data is transferred into it. Receive and transmit always take place together, at opposite clock edges.

The control of the protocol can be done by using the transmit interrupt flag UTXIFG or the receive interrupt flag URXIFG. Using the UTXIFG immediately after sending the data from the shift register to the slave the data from the buffer is transferred to the shift register and the transmission starts. The slave receive timing needs to ensure pick-up of the data in time. The URXIFG flag indicates when the data is shifted out and in completely. The master can use URXIFG to ensure that the slave should be ready to receive the next data properly.

Any standard digital output including STE in standard digital port function can be used to select a slave. The slave use the STE signal to enable its access to the SOMI data line and to enable to receive the clocks on UCLK.

4-pin SPI master mode, SYNC=1, STC=0, MM=1

The signal on STE is used by the active master to prevent bus conflicts with another master. The STE pin is input when the corresponding PnSEL bit selects the module function. The master operates normally while the STE signal is high. Whenever the STE signal is set to low - e.g. another device requests to become master - the actual master reacts with:

- the pins that drive the SPI bus lines SIMO and UCLK, are set to inputs
- the error bit FE and the interrupt flag URXIFG in the URCTL register are set.

The bus conflict is then removed - SIMO and UCLK do not drive the bus lines - and the error flag indicates to the software the violation of the system integrity. The pins SIMO and UCLK are forced to inputs while STE is low, and return to the conditions defined by the corresponding control bits when STE returns to high.

In the 3-pin mode the STE input signal is not relevant for the master.

13.1.2 Slave Mode in SPI Mode, MM=0, SYNC=1

The slave mode is selected when the master mode bit MM in the control register is reset and synchronous mode is selected.

The UCLK pin is used as the input for the serial shift clock supplied by an external master. The transfer rate is determined by this clock and not by the internal bit rate generator. The data, loaded into transmit shift register via transmit buffer UTXBUF before start of UCLK, is transmitted on SOMI pin using the UCLK applied by the master. Simultaneously the serial data applied to SIMO pin are shifted into the receive shift register on the opposite edge of the clock.

The receive interrupt flag URXIFG indicates when data is received and transferred into the receive buffer. The overrun error bit is set when previous received data is not read before the new data is written to the receive buffer.

4 pin SPI slave mode, SYNC=1, MM=0, STC=0

In the 4 pin SPI mode the signal STE is used by the slave to enable transmit and receive operation. The STE signal is used to enable the receive and transmit function of the slave. It is applied from the SPI master. The receive and transmit operation is disabled when the STE signal is high, and enabled when it is low. Whenever the STE signal becomes high any started receive operation is halted, and continues when the STE signal is low again. The STE signal is used to enable one slave to access the data lines. The SOMI is input if STE is high.

13.2 Interrupt and Control Function

The USART peripheral serves two main interrupt sources, the transmission and receive. Two individual interrupt vectors are available, one for receive and one for transmit interrupt events.

The control bits of the USART are located in the SFR address range:

•	Receive Interrupt Flag	URXIFG	initial state reset (by PUC/SWRST)
•	Receive Interrupt Enable	URXIE	initial state reset (by PUC/SWRST)
•	Receive Enable	URXE	initial state reset (by PUC)
•	Transmit Interrupt Flag	UTXIFG	initial state set (by PUC/SWRST)
•	Transmit Interrupt Enable	UTXIE	initial state reset (by PUC/SWRST)
•	Transmit Enable	UTXE	initial state reset (by PUC)

The receiver and transmitter of the USART operate in parallel and use the same baud rate generator in synchronous master mode. In synchronous slave mode the external clock - applied to UCLK - is used for receiver and transmitter.

13.2.1 USART Receive Enable

The Receiver Enable bit URXE enables or disables the receiver from collecting the bit stream on the URXD/SOMI data line. Disabling the USART receiver (URXE=0) will stop the receive operation after completing a started receive operation, or stop immediately if no receive operation is active. In synchronous mode the clock UCLK does not shift any data into the receiver shift register.

Receive when MSP430 is master
The receive operation is identical for 3-pin and 4-pin mode, when MSP430 USART is selected to be SPI master.

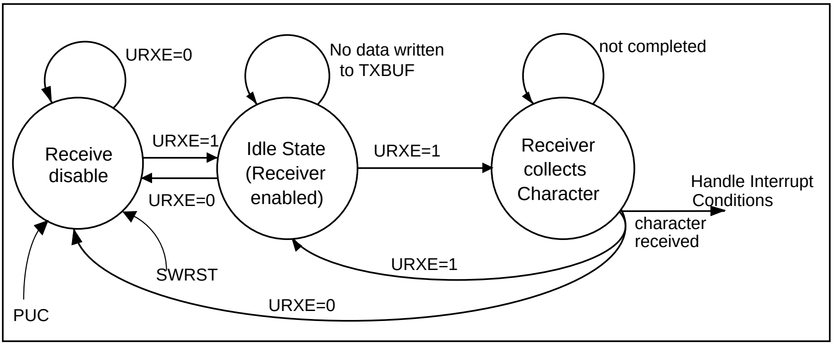


Figure 13.4: State diagram on Receiver enable URXE. MSP430 is master

Receive when MSP430 is slave, 3-pin mode

The receive operation is different for 3-pin and 4-pin mode when MSP430 USART is selected to be SPI slave. In the 3-pin mode no external SPI receive control signal stops a receive operation which has started. Power-up clear PUC, software reset SWRST or receive enable URXE can stop a receive operation and reset the USART.

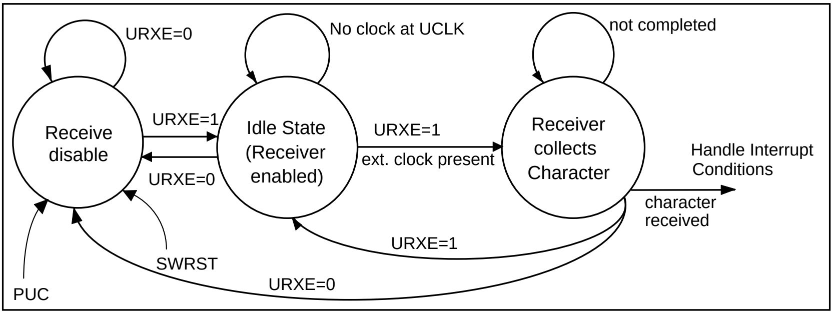


Figure 13.5: State diagram on Receiver enable URXE. MSP430 is slave/3-pin mode

Note: URXE re-enable, SPI Mode

Since the receiver is completely disabled a re-enable of the receiver is asynchronous to any data stream on the communication line. Synchronization to the data stream should be handled by the software protocol as usual in 3-pin SPI mode.

Receive when MSP430 is slave, 4-pin mode
In the 4-pin mode the external SPI receive control signal applied to pin STE stops a started receive operation. Power-up clear PUC, software reset SWRST or receive enable URXE can stop a receive operation and reset the operation control state machine. Whenever the STE signal is set to high, the receive operation is halted.

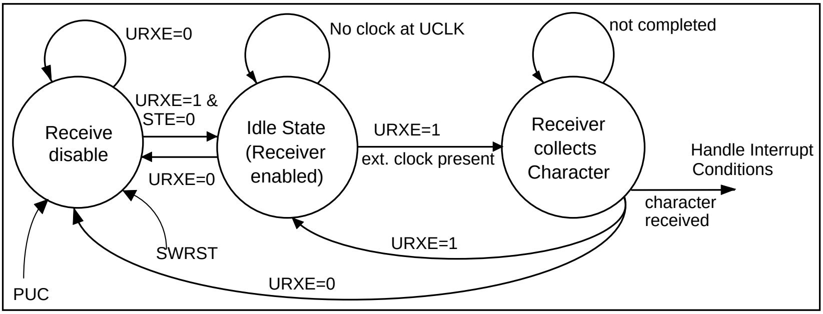


Figure 13.6: State diagram on Receiver enable URXE. MSP430 is slave/4-pin mode

13.2.2 USART Transmit Enable

The transmit enable bit UTXE enables or disables a character from being shifted onto the serial data line. If this bit is reset, the transmitter is disabled but any active transmission is not halted until all data previously written into the transmit buffer has been sent. If the transmission is completed any further write to the transmitter buffer will not result in a data transmission. When the UTXBUF was ready, a pending request for transmission will remain, and this results in an immediate start of transmission when UTXE is set and the transmitter is empty. A low signal on the STE signal removes the active master (4-pin mode) from the bus. Low at STE indicates that another master requests the active master function.

USART Transmit Enable, MSP430 is master

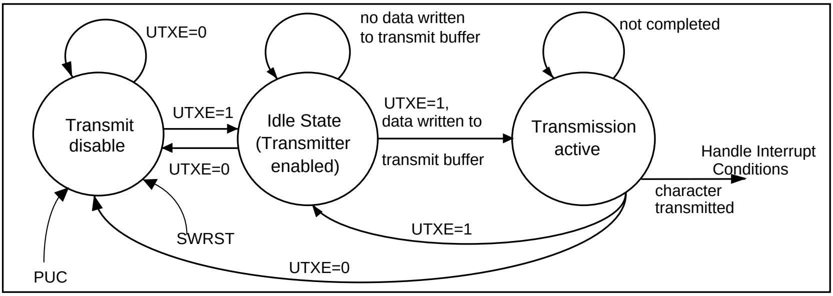


Figure 13.7: State diagram on Transmitter enable, MSP430 is master

USART Transmit Enable, MSP430 is slave

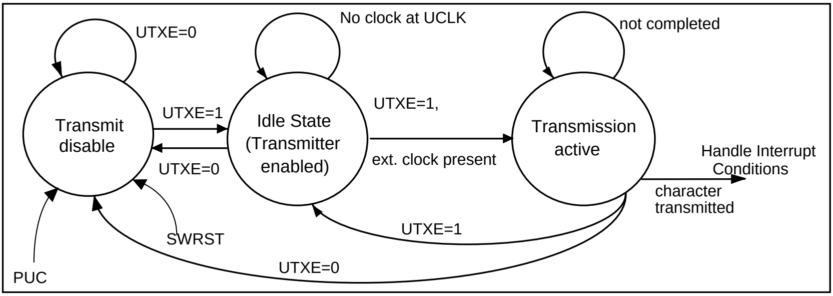


Figure 13.8: State diagram on Transmitter enable, MSP430 is slave

When UTXE is reset any data can be written regularly into the transmit buffer, but no transmission is started. Once the UTXE bit is set, the data in the transmit buffer are immediately loaded into the transmit shift register and the transmission of the character is started.

Note: Write to UTXBUF, SPI Mode

Data should never be written into the transmit buffer UTXBUF when it is not ready (UTXIFG=0) but the transmitter is enabled (UTXE=1). The character shifted out can be random.

13.2.3 USART Receive Interrupt Operation

The receive interrupt flag URXIFG is set each time a character is received and loaded into the receive buffer. Asynchronous conditions are not used.

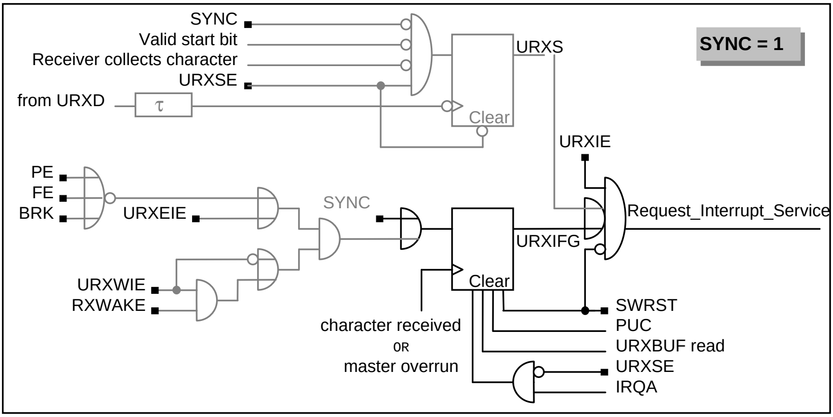


Figure 13.9: Receive Interrupt Conditions

URXIFG is reset at system reset PUC and at a software reset SWRST. URXIFG is reset automatically if the interrupt is served or the receive buffer URXBUF is read. The Receive Interrupt Enable bit URXIE enables, if set, serving of a pending interrupt request. Both, the receive interrupt flag URXIFG and the receive interrupt enable bit URXIE are reset with PUC and SWRST.

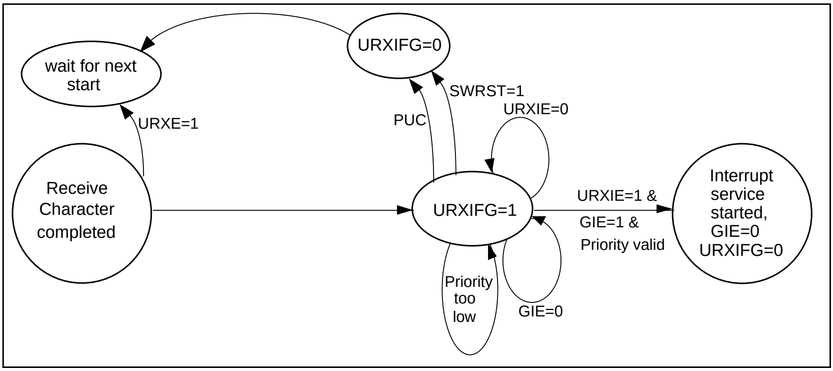


Figure 13.10: State diagrams on receive interrupt

13.2.4 USART Transmit Interrupt Operation

The transmit interrupt flag UTXIFG is set by the transmitter to indicate that the transmitter buffer UTXBUF is ready to accept another character. This bit is automatically reset if the interrupt request service is started or a character is written into the UTXBUF. This flag will assert a transmitter interrupt if the local (UTXIE) and general (GIE) interrupt enable bits are set. The UTXIFG is set after system reset PUC or SWRST are removed.

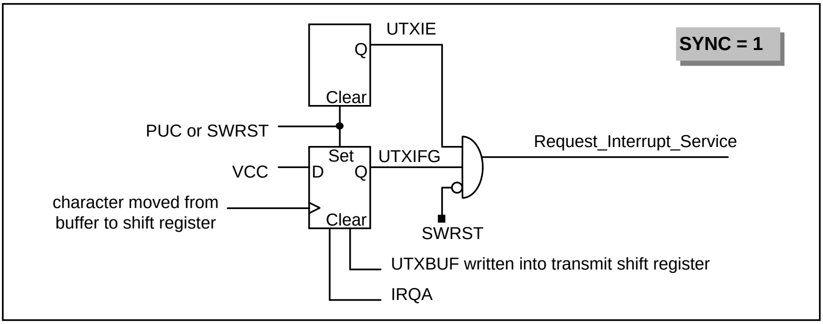

UTXIE
SYNC = 1
Q
Clear
PUC or SWRST
Request_Interrupt_Service
Set
UTXIFG
VCC
D
Q
character moved from buffer to shift register
Clear
SWRST
UTXBUF written into transmit shift register
IRQA

Figure 13.11: Transmit Interrupt Condition

The transmit interrupt enable UTXIE bit controls the ability of the UTXIFG to request an interrupt, but does not prevent the flag UTXIFG from being set. The UTXIE is reset with PUC or software reset bit SWRST. The UTXIFG bit is set after system reset PUC or software reset but the UTXIE bit is reset to ensure full interrupt control capability.

13.3 Control and Status Register

The USART module hardware is byte structured and should be accessed by byte processing instructions (suffix 'B').

Register	short form	Register type	Address	Initial state
● USART Control register	UCTL	Type of read/write	070h	See ....
● Transmit Control register	UTCTL	Type of read/write	071h	individual ...
● Receive Control register	URCTL	Type of read/write	072h	bit description
● Modulation Control reg.	UMCTL	Type of read/write	073h	unchanged
● Baud Rate register 0	UBR0	Type of read/write	074h	unchanged
● Baud Rate register 1	UBR1	Type of read/write	075h	unchanged
● Receive Buffer	URXBUF	Type of read/write	076h	unchanged
● Transmit Buffer	UTXBUF	Type of read	077h	unchanged

All bits are random after PUC unless otherwise noted by the detailed functional description.
Reset of the USART is performed by PUC or SWRST bit. After power-up (PUC) the SWRST bit remains set and the USART remains in this condition until the reset is disabled by resetting the SWRST bit. The SPI mode is disabled after PUC.
The USART module operates in asynchronous or in synchronous mode defined by the SYNC bit. The bits in the control registers may have different functions in the two modes. All bits are described with their function in the synchronous mode - SYNC=1. Their function in the asynchronous mode is described in the USART's serial interface UART mode section.

13.3.1 USART Control register

The information stored in the control register determines the basic operation of the USART module. The register bits select the communication mode and number of bits per character. All bits should be programmed according to the selected mode before reset is disabled by resetting bit SWRST.

	7							0
UCTL 070h	unused	unused	unused	CHAR	Listen	SYNC	MM	SWRST
	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

Figure 13.12: USART Control Register

Bit 0: The USART state machines and operating flags are initialized to the reset condition, if the software reset bit is set. Until the SWRST bit is reset, all affected logic is held in the reset state. This implies that after a system reset the USART must be re-enabled by resetting this bit.

Bit 1: Master mode is selected when the MM bit is set. The USART module slave mode is selected when the MM bit is reset.

Bit 2: Peripheral module mode select.
The SYNC bit selects the function of the USART peripheral interface module. Some of the USART control bits will have different functions in UART and SPI mode.
SYNC = 0 : UART function is selected.
SYNC = 1 : SPI function is selected.

Bit 3: The Listen bit selects if internally the transmitted data is fed back into the receiver

Bit 4: Character length.
This register bit selects the length of the character to be transmitted as 7 or 8 bits.
CHAR = 0 : 7 bit data.
CHAR = 1 : 8 bit data.

Bit 5: unused
Bit 6: unused
Bit 7: unused

13.3.2 Transmit Control Register UTCTL

The register controls the USART hardware associated with transmitter operation.

	7							0
UTCTL 071h	CKPH	CKPL	SSEL1	SSEL0	unused	unused	STC	TXEPT
	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

Figure 13.13: USART Transmitter Control Register

Bit 0: The transmitter empty TXEPT flag is set when the transmitter shift register and UTXBUF are empty, and reset, when data is written to UTXBUF. It is set on SWRST.

Bit 1: The slave transmit control bit STC selects if the signal at STE pin is used in the master and slave.
STC = 0: Four pin mode of SPI is selected. The STE signal is used by the master to avoid bus conflicts or it is used in slave mode to control transmit and receive enable.
STC = 1: Three pin SPI mode. STE is not used in master mode nor in slave mode.

Bit 2: unused

Bit 3: unused

Bit 4,5: Source Select 0 and 1.
The source select bits define - only when master mode is selected - which clock source is used for the baud rate generation:

SSEL1,SSEL0 0 external clock selected, UCLK
1 auxiliary clock selected, ACLK
2, 3 main system clock selected, MCLK

In master mode (MM=1) an external clock at UCLK can not be selected since the master applies the UCLK signal for any slave.
In the slave mode the bits SSEL1 and SSEL0 are not relevant. The external clock UCLK is always used.

Bit 6,7: Clock polarity CKPL and Clock Phase CKPH.
The CKPL bit controls the polarity of the SPICLK signal.

CKPL = 0: the inactive level is low; data is output with the rising edge of UCLK; input data is latched with the falling edge of UCLK.

CKPL = 1: the inactive level is high; data is output with the falling edge of UCLK; input data is latched with the rising edge of SPICLK.

The CKPH bit controls the polarity of the SPICLK signal.

CKPH = 0: normal UCLK clocking scheme.

CKPH = 1: UCLK is delayed by one half cycle.

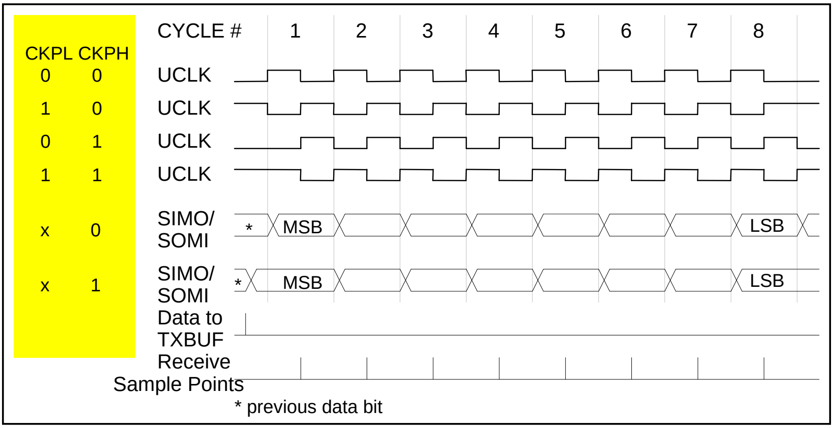


Figure 13.14: USART Clock Phase and Polarity

When operating with the CKPH bit set, the USART (synchronous mode) makes the first bit of data available after the transmit shift register is loaded and before the first edge of UCLK. Data is latched on the first edge of the UCLK and transmitted on the second edge in this mode.

13.3.3 Receive Control Register URCTL

The register URCTL controls the USART hardware associated with receiver operation and holds error conditions.

	7							0
URCTL 072h	FE	undef.	OE	undef.	unused	unused	undef.	undef.
	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Figure 13.15: USART Transmitter Control Register

Bit 0: undefined, driven by USART hardware
Bit 1: undefined, driven by USART hardware
Bit 2: unused
Bit 3: unused
Bit 4: undefined, driven by USART hardware
Bit 5: The overrun error flag bit OE is set when a character is transferred into URXBUF before the previous character has been read. The previous character is overwritten and lost. OE is reset by SWRST, system reset, by reading the URXBUF and by instruction.
Bit 6: undefined, driven by USART hardware
Bit 7: Frame error. The FE bit is set when a bus conflict stopped an active master with a negative transition of the signal applied to pin STE - only when 4-pin mode is selected. FE is reset by SWRST, system reset, by reading the URXBUF and by instruction.

13.3.4 Baud Rate Select and Modulation Control Registers

The baud rate generator uses the content of both baud rate select registers UBR1 and UBR0 to generate the bit timing for the serial data stream. The smallest division factor is two.

	7							0
UBR0 074h	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	rw	rw	rw	rw	rw	rw	rw	rw

	7							0
UBR1 075h	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8
	rw	rw	rw	rw	rw	rw	rw	rw

Figure 13.16: USART Baud Rate Select Register

$$\text{Baudrate} = \frac{\text{BRCLK}}{\text{UBR} + \frac{1}{n}\sum_{i}^{n} mi} \qquad \text{with UBR= [UBR1,UBR0]}$$

The maximum baud rate that can be selected for transmission in master mode is half of the clock input frequency of the baud rate generator. In slave mode, it is determined by the external clock applied to UCLK.

The modulation control is not used for serial synchronous communication. It is recommended to keep it reset (bits m0 to m7 are 0).

	7							0
UMCTL 073h	m7	m6	m5	m4	m3	m2	m1	m0
	rw	rw	rw	rw	rw	rw	rw	rw

Figure 13.17: USART Modulation Control Register

13.3.5 USART Receive Data Buffer URXBUF

The receiver buffer URXBUF contains previous data from the receiver shift register. URXBUF is cleared by SWRST or PUC. Reading URXBUF resets the receive error bits and receive interrupt flag URXIFG.

	7							0
URXBUF 076h	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	r	r	r	r	r	r	r	r

Figure 13.18: USART Receive Buffer

In 7-bit length mode the MSB of the URXBUF is always reset.

13.3.6 USART Transmit Data Buffer UTXBUF

The transmit buffer contains current data to be transmitted by the transmitter.

	7							0
UTXBUF 077h	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Figure 13.19: USART Transmit Buffer

The UTXIFG bit indicates that UTXBUF is ready to accept another character for transmission.

In master mode, the transmission will be initialized by writing data to UTXBUF. The transmission of this data is started immediately, if the transmitter shift register is empty or is going to be empty.

When seven bits/character is selected the data moved into the transmit buffer should be left adjusted since the MSB is shifted out first.

14 Liquid Crystal Display Drive

Topic **Page**

14.1 Basics of LCD Drive

Liquid crystal displays use ambient luminescence to display information and do not send out light actively. This results in low power consumption. The requirement for visible displayed information is sufficient ambient luminescence.

The liquid crystal must be driven with alternating voltage. DC drive would destroy the liquid crystal. This AC drive requirement is the main factor for any power consumption. The electrical equivalence for the driving stage is a capacitor. Its electrodes are the back plane or common plane, controlled by signal COMn and the segment driven by SEGn. The frequency of the AC drive is low - in the range of 1000 Hz to 30 Hz. The data sheets of the LCD manufacturer give defined ranges for this frequency.

Different methods of controlling LC displays were developed in the past. The different driving methods are applied as a compromise between number of segments, number of pins at display and driving source, LCD contrast, temperature range, ...
Multiplexing methods reduce the number of pins needed.

The MSP430 Family's LCD module supports four driving methods:

- Static
- 2MUX or 1/2 duty, 1/2 bias
- 3MUX or 1/3 duty, 1/3 bias
- 4MUX or 1/4 duty, 1/3 bias.

The static method needs one pin for common plane (COM0) and one pin for each segment:

#-of-pins = 1 + #-of-segments

The 2MUX method needs two pins for common plane (COM0, COM1) and one pin for two segments:

#-of-pins = Integer [2 + (#-of-segments/2)]

The 3MUX method needs three pins for common plane (COM0, COM1, COM2) and one pin for three segments:

14

#-of-pins = Integer [3 + (#-of-segments/3)]

The 4MUX method needs four pins for common plane (COM0, COM1, COM2, COM3) and one pin for four segments:

#-of-pins = Integer [4 + (#-of-segments/4)]

The increase of the multiplex rate reduces the number of pins required. The continuous reduction of pin counts is demonstrated by an application that uses 80 segments:

Static method:	#-of-pins = (1 + 80)	= 81
2MUX:	#-of-pins = (2 + 80/2)	= 42
3MUX:	#-of-pins = (3 + 80/3)	= 30
4MUX:	#-of-pins = (4 + 80/4)	= 24

Static Driving Method

In the static drive method each segment line drives one segment.
The example shows one digit of the liquid crystal displaying ‘5’, including an example of the connections together with the output wave forms.

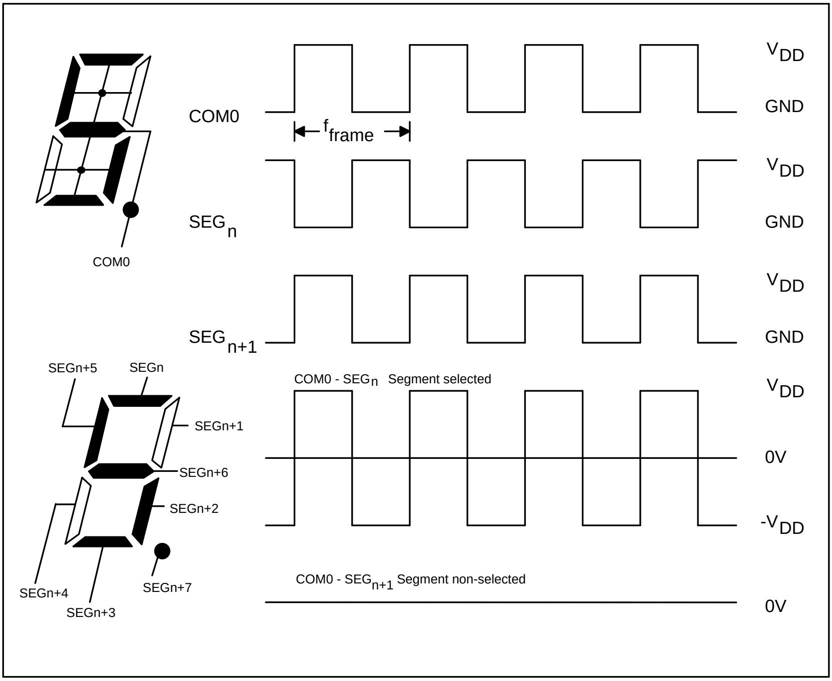


Figure 14.1: Example of static wave form drive

Two MUX, ½ Bias

In the 2MUX drive each segment line drives two segments.
The example shows one digit of the liquid crystal display displaying '5', including an example of the connections, together with the output wave forms.

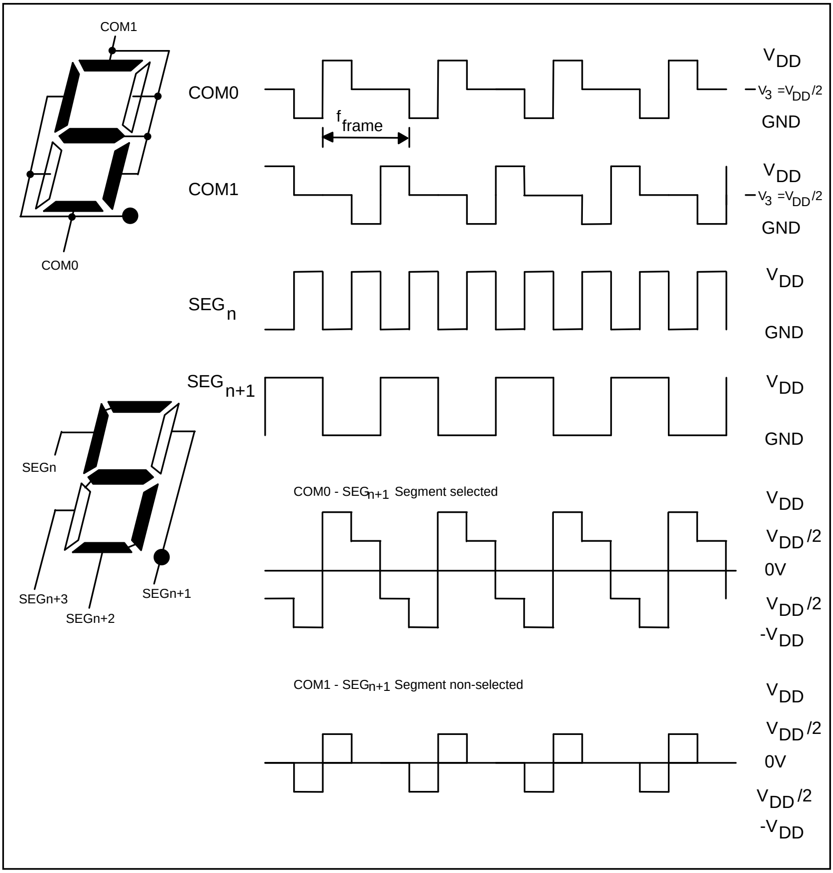


Figure 14.2: Example of 2MUX wave form drive

Three MUX, $^1/_3$ Bias

In the 3MUX drive each segment line drives three segments.
The example shows one digit of the liquid crystal display displaying '5', including an example of the connections, together with the output wave forms.

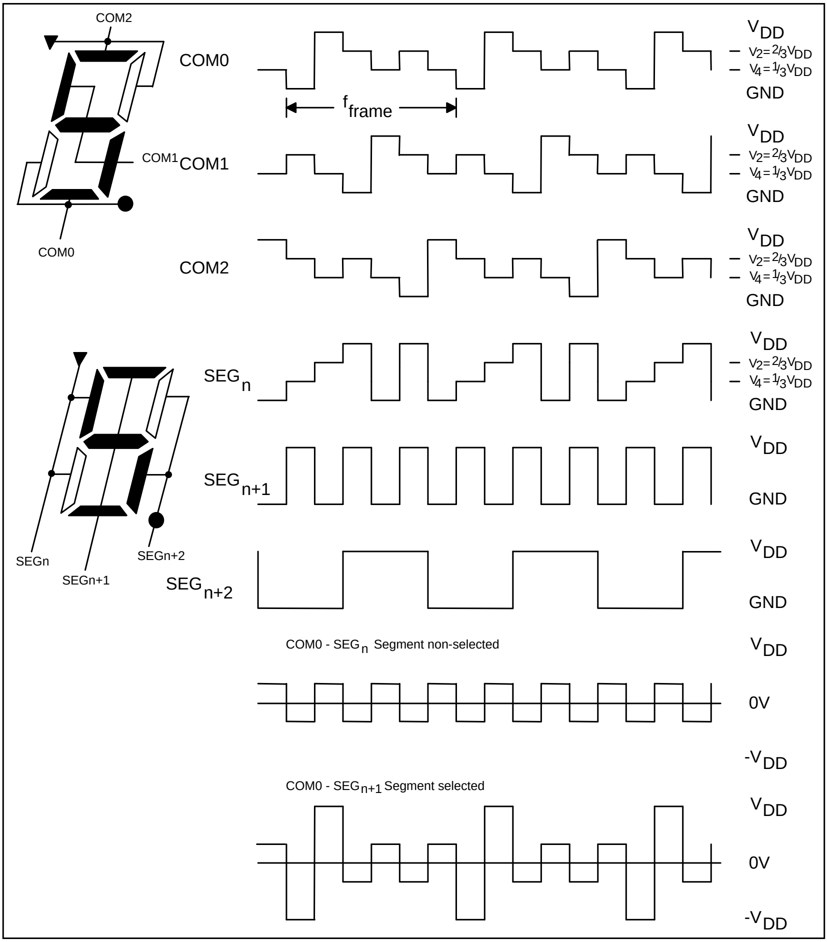


Figure 14.3: Example of 3MUX wave form drive

Four MUX, $^1/_3$ Bias

In the 4MUX drive each segment line drives four segments.
The example shows one digit of the liquid crystal display displaying '5', including an example of the connections, together with the output wave forms.

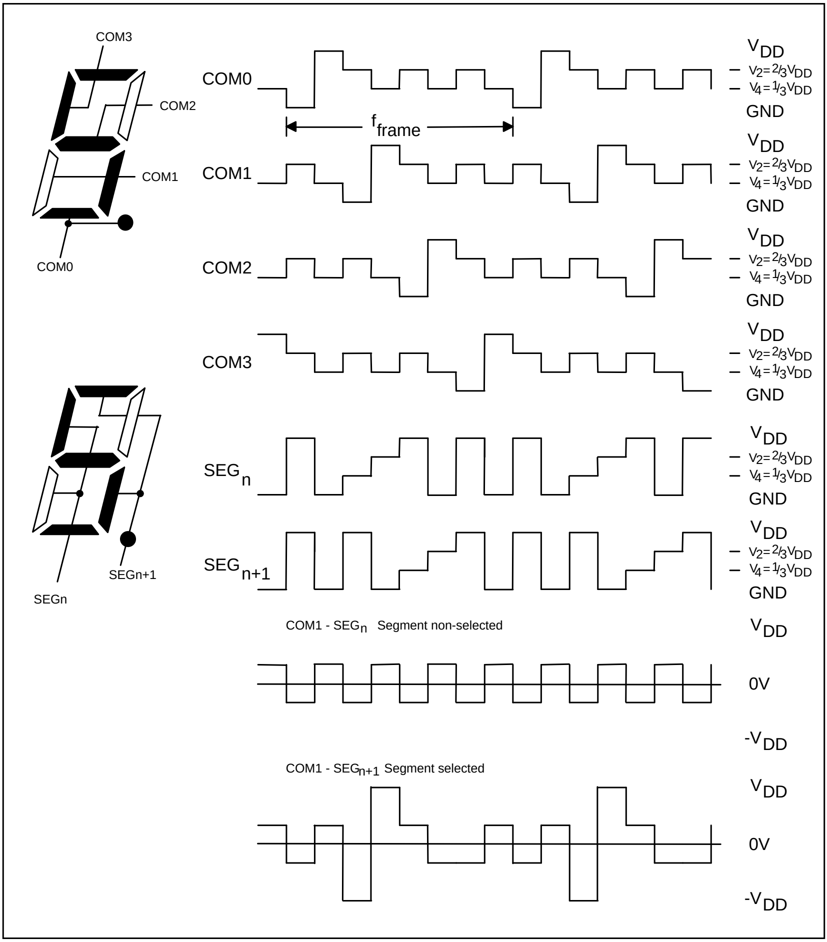


Figure 14.4: Example of 4MUX wave form drive

14.2 LCD Controller/Driver

The LCD controller/driver peripheral generates the segment and common signals according to the data in the display data memory. It contains all functional blocks to drive an external directly connected LCD. The main blocks in the LCD peripherals are:

- Data memory containing the segment information
- Timing generator
- Module bus interface
- LCD Module Analog voltage applied externally
- LCD+ Module only: Analog voltage generator internally

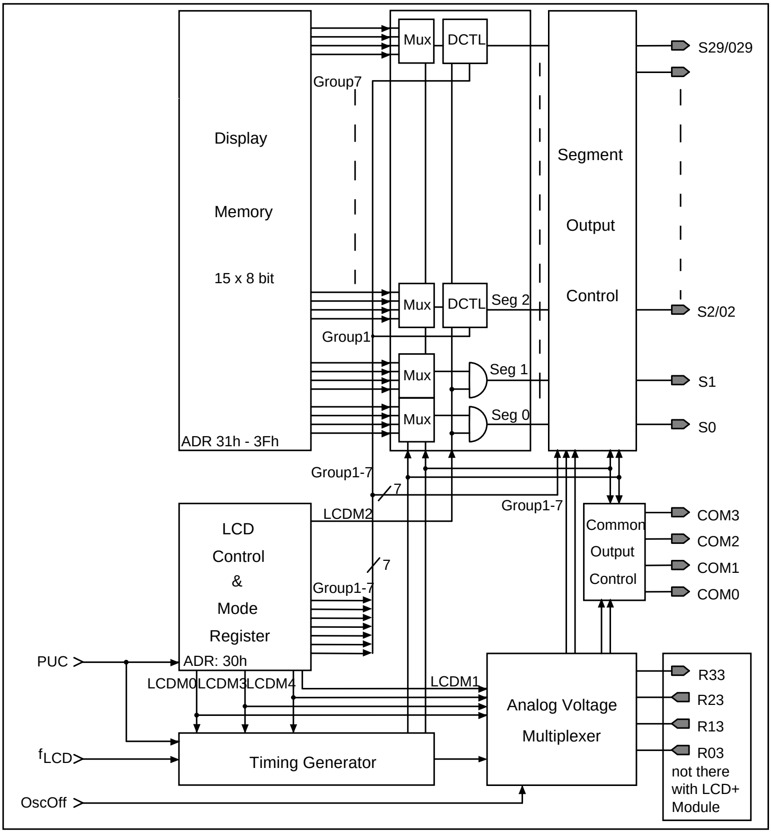


Figure 14.5: LCD Controller/Driver Block Diagram

Differences between LCD Module and LCD+ Module:

	LCD Module	**LCD+ Module**
● Analog Voltage Generation	external options: ● 2 inputs R23, R13 V1 = VCC V5 = VSS ● 3 inputs R23, R13, R03 V1 = VCC ● 4 inputs R33, R23, R13, R03	internal
● Control bit LCDM1	unused	selects impedance of R-Ladder
● Control bit LCDM0	● stops timing generator	● stops timing generator ● stops current through R-Ladder

14.2.1 LCD Controller/Driver Functions

The functions of the LCD Controller/Driver are:

- Reads automatically data from the display memory, and generates the segment and common signals
- Four different display modes are selectable:
  Static mode
  2MUX , 1/2 bias
  3MUX , 1/3 bias
  4MUX , 1/3 bias.
  Within the basic timer BT, there are two bits to select one of four different frame frequencies.
- Segment signal outputs can be switched to an output port
- Display memory not used for segment information can be used as a normal memory.
- Operation via the basic timer with the auxiliary clock (ACLK).
- LCD+ Module only:
  Resistive network to supply the analog voltage levels for LCD drive
  One bit in the control register LCDCTL controls the switch through which the resistive network is connected with V1.

The frame frequency of the LCD lines is:

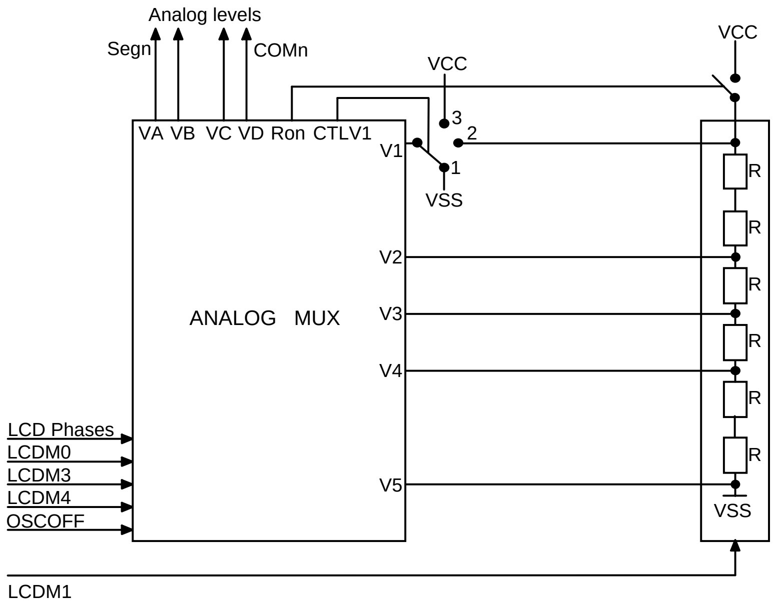
- Static method: $f_{frame} = \frac{1}{2} \text{ x } f_{LCD}$
- 2MUX: $f_{frame} = \frac{1}{4} \text{ x } f_{LCD}$
- 3MUX: $f_{frame} = \frac{1}{6} \text{ x } f_{LCD}$
- 4MUX: $f_{frame} = \frac{1}{8} \text{ x } f_{LCD}$

LCD+ Module:

The analog voltage is generated internally.

When the OSCOff bit in the status register is set, the power supply to the resistor network is switched off independently of the LCDM0 bit.

During static mode, the analog generator is switched to be inactive, since the static mode uses only V1 and V5 levels. Supply current consumption is reduced.



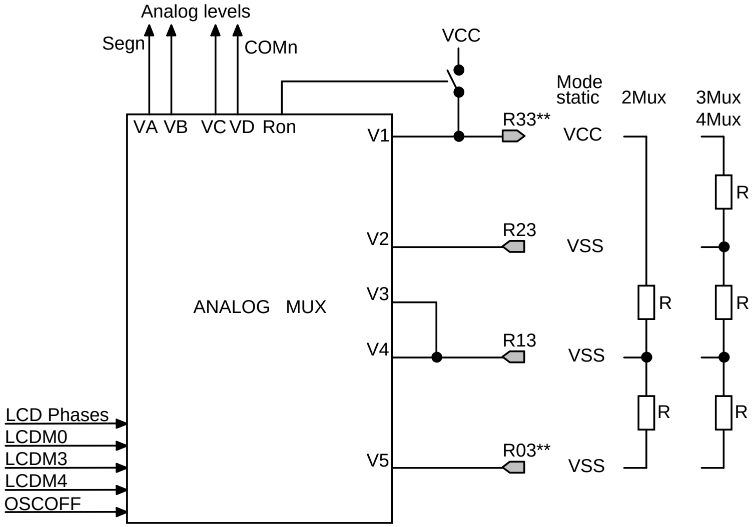
OSCOFF	LCDM4	LCDM3	LCDM0	VA	VB	VC	VD	RON	CTLV1
X	X	X	0	0	0	0	0	OFF	1*
1	X	X	X	0	0	0	0	OFF	1*
0	0	0	1	V5/V1	V1/V5	V5/V1	V1/V5	OFF	3*
0	0	1	1	V5/V1	V1/V5	V3/V3	V1/V5	ON	2*
0	1	X	1	V5/V1	V2/V4	V4/V2	V1/V5	ON	2*

* ≡ to the position of the switch

Figure 14.6: Internal analog voltage generated by LCD+ Module

LCD Module:

The analog voltage is supplied externally, applied on pins R33**, R23, R13, R03**.



OSCOFF	LCDM4	LCDM3	LCDM0	VA	VB	VC	VD	RON *
X	X	X	0	0	0	0	0	OFF
1	X	X	X	0	0	0	0	OFF
0	0	0	1	V5/V1	V1/V5	V5/V1	V1/V5	OFF
0	0	1	1	V5/V1	V1/V5	V3/V3	V1/V5	ON
0	1	X	1	V5/V1	V2/V4	V4/V2	V1/V5	ON

* ≡ to the position of the switch

** Supply pins for V1 and V5 are optional. Devices without R33 and R03 pins will have V1 tied to VCC and V5 tied to VSS. External resistor ladder should be connected to VCC and VSS.

Figure 14.7: External analog voltage applied to LCD Module

Note: ** Supply pins R33 and R03 are optional. Please see device data-sheet.

14.2.2 LCD Control & Mode Register

The content of the LCD control & mode register defines the different operating conditions. The LCD module is byte structured and should be accessed by byte instructions (suffix .B).

	7							0
LCDCTL 030h	LCDM7	LCDM6	LCDM5	LCDM4	LCDM3	LCDM2	LCDM1	LCDM0
	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

LCDM0: LCDM0 = 0: The timing generator is switched off.
Common and segment lines are "L".
Outputs selected for port output lines are not affected.
LCD+ Module: Power supply to resistor network is off.
LCDM0 = 1: Common and segment lines output the signal corresponding to the display memory.
Outputs selected for port output lines are not affected.
LCD+ Module: Power supply to resistor network is switched on at 2MUX, 3MUX and 4MUX not at static mode.

LCDM1: This bit selects the LCD drive magnitude, by selecting the internal resistance of the Analog Generator. It is only valid along with the LCD+ Module.
LCDM1 = 0 : High impedance of Analog Generator.
LCDM1 = 1 : Low impedance of Analog Generator.

LCDM2,3,4: These three bits select the display mode and can switch the segment output to non-selected level.

LCDM4	LCDM3	LCDM2	Display mode	Bias, LCD +	Bias, LCD
X	X	0	Not affected, Display is off - all Segment signals are non-selected level. The port outputs remain stable		
0	0	1	Static mode	1/1	R33*, R03*
0	1	1	2MUX mode	1/2	R33*, R13, R03*
1	0	1	3MUX mode	1/3	R33*, R23, R13, R03*
1	1	1	4Mux mode	1/3	R33*, R23, R13, R03*

* optional pins

The signal LCDM2 disables (0) or enables (1) the segment lines. This is done with an AND combination with each individual segment information. It is located in the parallel serial conversion block between the output of the display memory and the segment output control. The segment information in the display memory remains.
The major purpose of this function is to support applications with flashing displays.

LCDM5,6,7: The information of the three bits selects groups of outputs to carry segment information or bit port information. The outputs selected for port function are driven with the state of the display memory bit, and are no longer part of the LCD segment lines.

LCDM7	LCDM6	LCDM5	Group0	Group1	Group2	Group3	Group4	Group5	Group6	Group7
0	0	0	S0-S1	O2-O5	O6-O9	O10-O13	O14-O17	O18-O21	O22-O25	O26-O29
0	0	1	S0-S1	S2-S5	O6-O9	O10-O13	O14-O17	O18-O21	O22-O25	O26-O29
0	1	0	S0-S1	S2-S5	S6-S9	O10-O13	O14-O17	O18-O21	O22-O25	O26-O29
0	1	1	S0-S1	S2-S5	S6-S9	S10-S13	O14-O17	O18-O21	O22-O25	O26-O29
1	0	0	S0-S1	S2-S5	S6-S9	S10-S13	S14-S17	O18-O21	O22-O25	O26-O29
1	0	1	S0-S1	S2-S5	S6-S9	S10-S13	S14-S17	S18-S21	O22-O25	O26-O29
1	1	0	S0-S1	S2-S5	S6-S9	S10-S13	S14-S17	S18-S21	S22-S25	O26-O29
1	1	1	S0-S1	S2-S5	S6-S9	S10-S13	S14-S17	S18-S21	S22-S25	S26-S29

← reset condition

Note: LCD control bits

The control bits LCDM5 ... LCDM7 are reset with PUC.

Function Seg: The Sxx signals are part of the display driving signals, and carry modulated voltage levels according to the time frame of the common lines.

Function Port: The signals selected for 'port' function are static signals. They take two digital levels according to bits in the display memory. The logical state of the bits is taken from bit 0 to bit 3 for even S-lines (n=3,5,.....) and from bit 4 to bit 7 for odd S-lines (n=2,4,.....).

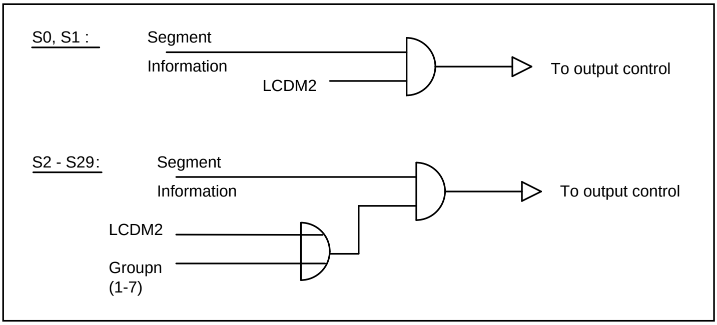


Figure 14.8: Information control

14.2.3 LC Display Memory

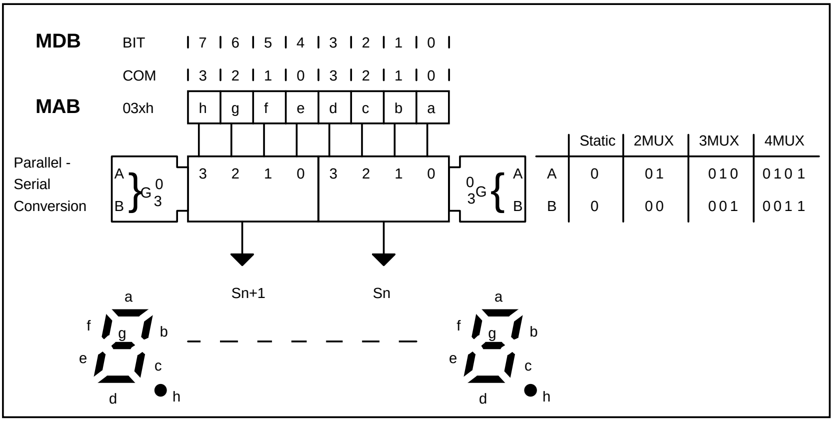
The LC Display Memory holds the information to be displayed during all operating and power down modes. The bits in the memory are directly attached to the segments of the liquid crystal display. The figures displayed at the LCD are decoded by the software from the BCD or binary representation to the segment/common combination of the individual display. The bit information in the memory matches with one common line and one segment line. The bit information in the memory corresponds to the selection of segments - a bit set in the memory is identical with segment selection 'on' and reverse.

One segment line carries the on/off state of one to four segments depending on the multiplex rate:

Static drive -> state of one segment/segment line
2MUX drive -> state of two segment/segment line
3MUX drive -> state of three segment/segment line
4MUX drive -> state of four segment/segment line

The timing generator of the LCD controller/driver drives the conversion of the parallel information stored in the LC Display Memory into the serial information required for the segment line signal. The bits of the LC Display Memory are hard wired to the common lines:

Static drive -> COM0: Bit 0 to Sn, Bit 4 to Sn+1
2MUX drive -> COM0: Bit 0 to Sn, Bit 4 to Sn+1, COM1: Bit 1 to Sn, Bit 5 to Sn+1
3MUX drive -> COM0: Bit 0 to Sn, Bit 4 to Sn+1, COM1: Bit 1 to Sn, Bit 5 to Sn+1
COM2: Bit 2 to Sn, Bit 6 to Sn+1
4MUX drive -> COM0: Bit 0 to Sn, Bit 4 to Sn+1, COM1: Bit 1 to Sn, Bit 5 to Sn+1
COM2: Bit 2 to Sn, Bit 6 to Sn+1, COM3: Bit 3 to Sn, Bit 7 to Sn+1



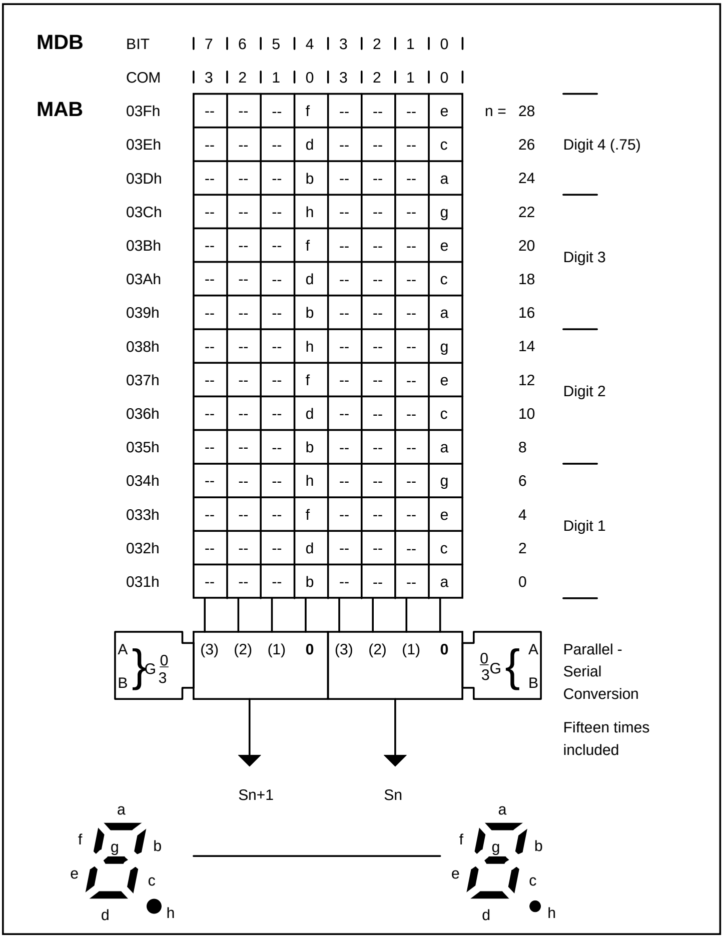
	Static	2MUX	3MUX	4MUX
A	0	0 1	0 1 0	0 1 0 1
B	0	0 0	0 0 1	0 0 1 1

Figure 14.9: Bits of Display Memory attached to Segment lines

Display memory using the static driving method

The static driving method uses one common line. The active common line is COM0. In this mode BIT0 and BIT4 are used for segment information. The other bits can be used like any other memory.

The maximum number of segments is 30.



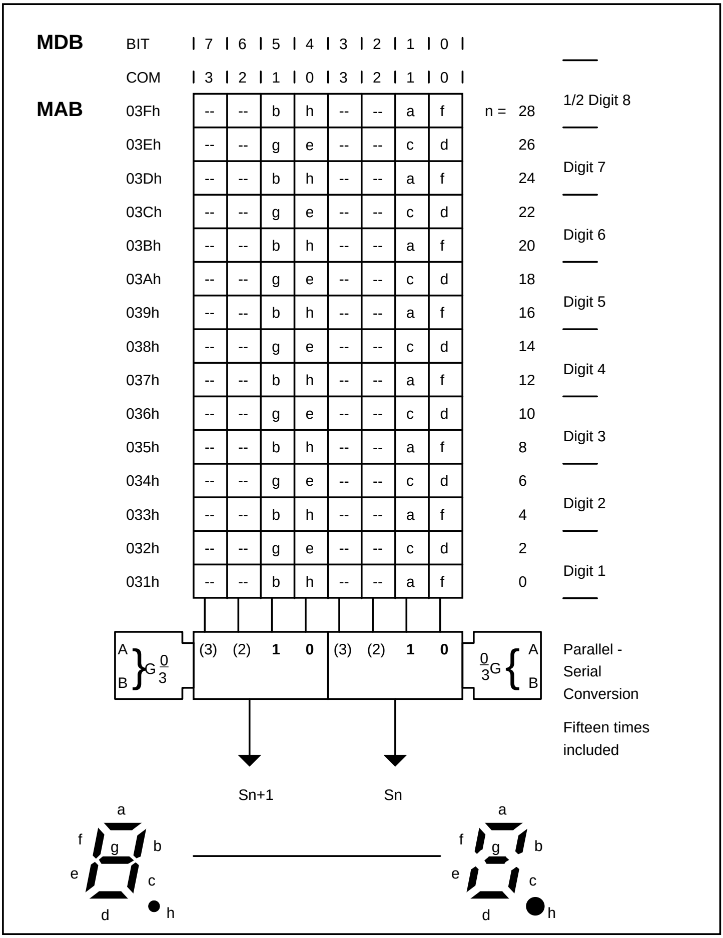
MDB	BIT	7	6	5	4	3	2	1	0		
	COM	3	2	1	0	3	2	1	0		
MAB	03Fh	--	--	--	f	--	--	--	e	n = 28	Digit 4 (.75)
	03Eh	--	--	--	d	--	--	--	c	26	
	03Dh	--	--	--	b	--	--	--	a	24	
	03Ch	--	--	--	h	--	--	--	g	22	Digit 3
	03Bh	--	--	--	f	--	--	--	e	20	
	03Ah	--	--	--	d	--	--	--	c	18	
	039h	--	--	--	b	--	--	--	a	16	
	038h	--	--	--	h	--	--	--	g	14	Digit 2
	037h	--	--	--	f	--	--	--	e	12	
	036h	--	--	--	d	--	--	--	c	10	
	035h	--	--	--	b	--	--	--	a	8	
	034h	--	--	--	h	--	--	--	g	6	Digit 1
	033h	--	--	--	f	--	--	--	e	4	
	032h	--	--	--	d	--	--	--	c	2	
	031h	--	--	--	b	--	--	--	a	0	

Figure 14.10: Use of Display Memory with the static driving method

Display memory using 2MUX, 1/2 bias driving method

The 2MUX driving method uses two common lines. The active common lines are COM0 and COM1. In this mode the BIT0, BIT1, BIT4 and BIT5 are used for segment information. The other bits can be used like any other memory.

The maximum number of segments is 60.



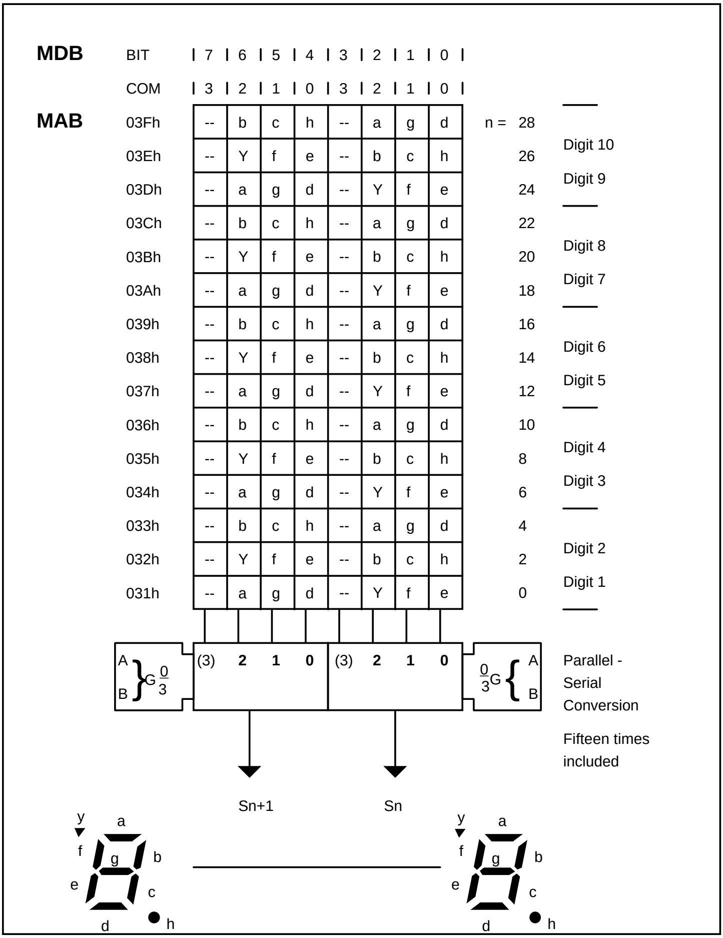
MDB	BIT	7	6	5	4	3	2	1	0		
	COM	3	2	1	0	3	2	1	0		
MAB	03Fh	--	--	b	h	--	--	a	f	n = 28	1/2 Digit 8
	03Eh	--	--	g	e	--	--	c	d	26	Digit 7
	03Dh	--	--	b	h	--	--	a	f	24	
	03Ch	--	--	g	e	--	--	c	d	22	Digit 6
	03Bh	--	--	b	h	--	--	a	f	20	
	03Ah	--	--	g	e	--	--	c	d	18	Digit 5
	039h	--	--	b	h	--	--	a	f	16	
	038h	--	--	g	e	--	--	c	d	14	Digit 4
	037h	--	--	b	h	--	--	a	f	12	
	036h	--	--	g	e	--	--	c	d	10	Digit 3
	035h	--	--	b	h	--	--	a	f	8	
	034h	--	--	g	e	--	--	c	d	6	Digit 2
	033h	--	--	b	h	--	--	a	f	4	
	032h	--	--	g	e	--	--	c	d	2	Digit 1
	031h	--	--	b	h	--	--	a	f	0	

Figure 14.11: Use of Display Memory with the 2MUX method

Display memory using 3MUX, 1/3 bias driving method

The 3MUX driving method uses three common lines. The active common lines are COM0, COM1 and COM2. In this mode BIT0, BIT1, BIT2, BIT4, BIT5 and BIT6 are used for segment information. The other bits can be used like any other memory.

The maximum number of segments is 90.



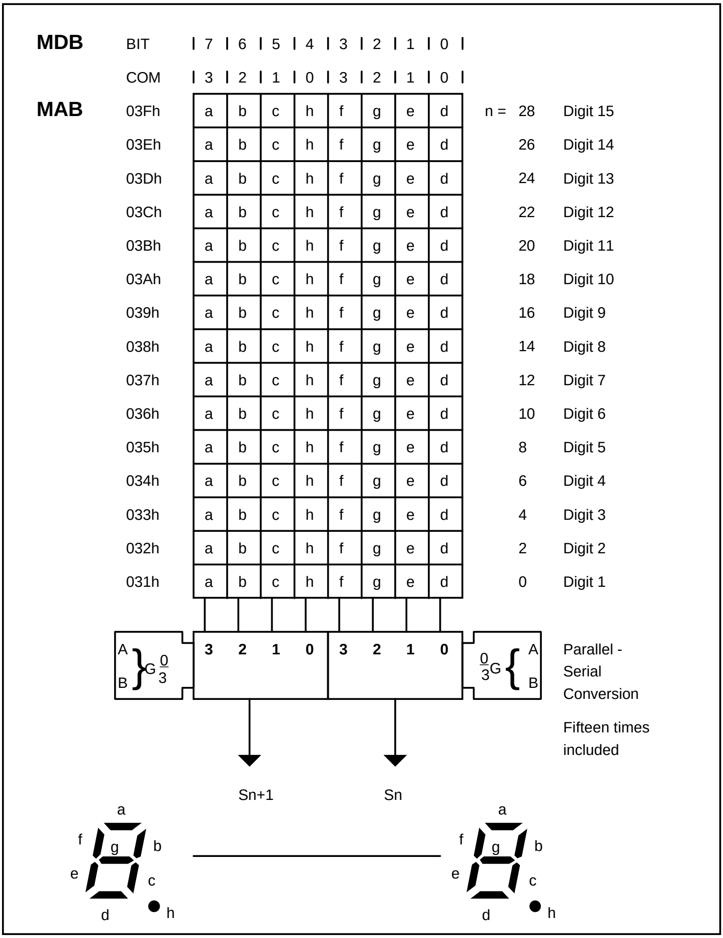
MDB BIT	7	6	5	4	3	2	1	0		
COM	3	2	1	0	3	2	1	0		
MAB 03Fh	--	b	c	h	--	a	g	d	n = 28	
03Eh	--	Y	f	e	--	b	c	h	26	Digit 10
03Dh	--	a	g	d	--	Y	f	e	24	Digit 9
03Ch	--	b	c	h	--	a	g	d	22	
03Bh	--	Y	f	e	--	b	c	h	20	Digit 8
03Ah	--	a	g	d	--	Y	f	e	18	Digit 7
039h	--	b	c	h	--	a	g	d	16	
038h	--	Y	f	e	--	b	c	h	14	Digit 6
037h	--	a	g	d	--	Y	f	e	12	Digit 5
036h	--	b	c	h	--	a	g	d	10	
035h	--	Y	f	e	--	b	c	h	8	Digit 4
034h	--	a	g	d	--	Y	f	e	6	Digit 3
033h	--	b	c	h	--	a	g	d	4	
032h	--	Y	f	e	--	b	c	h	2	Digit 2
031h	--	a	g	d	--	Y	f	e	0	Digit 1

Figure 14.12: Use of Display Memory with the 3MUX method

Display memory using 4MUX, 1/3 bias driving method

The 4MUX driving method uses four common lines. The active common lines are COM0, COM1, COM2 and COM3. In this mode BIT0, BIT1, BIT2, BIT3, BIT4, BIT5, BIT6 and BIT7 are used for segment information.

The maximum number of segments is 120.



MDB	BIT	7	6	5	4	3	2	1	0		
	COM	3	2	1	0	3	2	1	0		
MAB	03Fh	a	b	c	h	f	g	e	d	n = 28	Digit 15
	03Eh	a	b	c	h	f	g	e	d	26	Digit 14
	03Dh	a	b	c	h	f	g	e	d	24	Digit 13
	03Ch	a	b	c	h	f	g	e	d	22	Digit 12
	03Bh	a	b	c	h	f	g	e	d	20	Digit 11
	03Ah	a	b	c	h	f	g	e	d	18	Digit 10
	039h	a	b	c	h	f	g	e	d	16	Digit 9
	038h	a	b	c	h	f	g	e	d	14	Digit 8
	037h	a	b	c	h	f	g	e	d	12	Digit 7
	036h	a	b	c	h	f	g	e	d	10	Digit 6
	035h	a	b	c	h	f	g	e	d	8	Digit 5
	034h	a	b	c	h	f	g	e	d	6	Digit 4
	033h	a	b	c	h	f	g	e	d	4	Digit 3
	032h	a	b	c	h	f	g	e	d	2	Digit 2
	031h	a	b	c	h	f	g	e	d	0	Digit 1

Figure 14.13: Use of Display Memory with the 4MUX method

14.2.4 Software Examples for LCD Operation

The examples in this paragraph demonstrate the software to display digits on the LCD. They used the standard nomenclature of 7-segment digits.

Software for 4MUX, 1/3 bias LCD

```
                .sect "lcd4mux",0f000h
;   The 4MUX rate is the most easy-to-handle display rate. All eight segments of a digit
;   are located in one display memory byte
;
a               .EQU    080h
b               .EQU    040h
c               .EQU    020h
d               .EQU    001h
e               .EQU    002h
f               .EQU    008h
g               .EQU    004h
h               .EQU    010h
;
;   The  LSDigit of register Rx (000m) should be displayed.
;   The Table represents the 'on'-segments according to the content of Rx.
;
                ..........
LCD1            .EQU    00031h          ; Address of LC Display Memory
                ..........
                ..........
LCD15           .EQU    0003Fh
                ..........

;               ..........
                ..........
                MOV.B   Table(Rx),&LCDn ; n = 1 ..... 15
                                        ; all eight segments are written to the
                                        ; display memory
                ..........
                ..........
;
Table           .BYTE   a+b+c+d+e+f     ; displays "0"
                .BYTE   b+c             ; displays "1"
                ..........
                ..........
                .BYTE   b+c+d+e+g       ; displays "d"
                .BYTE   a+d+e+f+g       ; displays "E"
                .BYTE   a+e+f+g         ; displays "F"
```

Software for 3MUX, 1/3 bias LCD

```
                .sect "lcd3mux",0f000h
;       The 3MUX rate supports nine segments instead of eight segments for each digit.
 ;      The nine segments of a digit are located in 1 ½  display memory bytes.
;
a               .EQU    0040h
b               .EQU    0400h
c               .EQU    0200h
d               .EQU    0010h
e               .EQU    0001h
f               .EQU    0002h
g               .EQU    0020h
h               .EQU    0100h
Y               .EQU    0004h
;       The  LSDigit of register Rx (000m) should be displayed.
;       The Table represents the 'on'-segments according to the LSDigit of register of Rx.
;       The register Ry is used for temporary memory
;
LCD1            .EQU    00031h
                ...........
LCD15           .EQU    0003Fh
                ...........
                ...........
ODDDIG  RLA     Rx
                MOV     Table(Rx),Ry        ; Load segment information to
                                            ; temporary mem.
                                            ; (Ry) = 0000  0bch  0agd  0Yfe
                MOV.B   Ry,&LCDn            ; write 'a, b, c, d, e, f' of Digit n
                                            ; (LowByte)
                SWPB    Ry                  ; (Ry) =  0agd  0Yfe  0000  0bch
                BIC.B   #07h,&LCDn+1        ; write 'b, c, h' of Digit n (HighByte)
                BIS.B   Ry,&LCDn+1
                .....
EVNDIG  RLA     Rx
                MOV     Table(Rx),Ry        ; Load segment information to
                                            ; temporary mem.
                                            ; (Ry) = 0000  0bch  0agd  0Yfe
                RLA     Ry                  ; (Ry) = 0000  bch0  agd0  Yfe0
                RLA     Ry                  ; (Ry) = 000b  ch0a  gd0Y  fe00
                RLA     Ry                  ; (Ry) = 00bc  h0ag  d0Yf  e000
                RLA     Ry                  ; (Ry) = 0bch  0agd  0Yfe  0000
                BIC.B   #070h,&LCDn+1
                BIS.B   Ry,&LCDn+1          ; write 'Y, f, e' of Digit n+1 (LowByte)
                SWPB    Ry                  ; (Ry) = 0Yfe  0000  0bch  0agd
                MOV.B   Ry,&LCDn+2          ; write 'b, c, h, a, g, d' of Digit n+1
                                            ; (HighByte)
                ...........
```

```
Table           .WORD   a+b+c+d+e+f     ; displays "0"
                .WORD   b+c             ; displays "1"
                ..........
                ..........
                .WORD   a+e+f+g         ; displays "F"
```

Software for 2MUX, 1/2 bias LCD

```
            .sect "lcd2mux",0f000h
;           All eight segments of a digit are located in two display memory bytes with the
;           2MUX display rate
;
a                   .EQU    002h
b                   .EQU    020h
c                   .EQU    008h
d                   .EQU    004h
e                   .EQU    040h
f                   .EQU    001h
g                   .EQU    080h
h                   .EQU    010h
;           The register content of Rx (000m) should be displayed.
;           The Table represents the 'on'-segments according to the content of Rx.
;
                    ..........
;
LCD1                .EQU    00031h
                    ..........
                    ..........
LCD15               .EQU    0003Fh
;
                    ..........
                    ..........
                    MOV.B   Table(Rx),Ry        ; Load segment information to
                                                ; temporary mem.
                    MOV.B   Ry,&LCDn            ; (Ry) = 0000  0000  gebh  cdaf
                                                ; Note:
                                                ; All bits of an LCD memory byte are
                                                ; written
                    RRA     Ry                  ; (Ry) = 0000  0000  0geb  hcda
                    RRA     Ry                  ; (Ry) = 0000  0000  00ge  bhcd
                    MOV.B   Ry,&LCDn+1          ; Note:
                                                ; All bits of an LCD memory byte are
                                                ; written
                    ..........
                    ..........
;
Table               .BYTE   a+b+c+d+e+f         ; displays "0"
                    ..........
                    .BYTE   a+b+c+d+e+f+g+h     ; displays "8"
                    ..........
                    ..........
                    .BYTE
                    ..........
;
```

Software for static LCD

```
           .sect "lcd1mux",0f000h
;          All eight segments of a digit are located in four display memory bytes with the
;          static display method.
;
a                  .EQU        001h
b                  .EQU        010h
c                  .EQU        002h
d                  .EQU        020h
e                  .EQU        004h
f                  .EQU        040h
g                  .EQU        008h
h                  .EQU        080h
;          The register content of Rx should be displayed.
:          The Table represents the 'on'-segments according to the content of Rx.
;
                   ...........
;
LCD1               .EQU        00031h
                   ...........
                   ...........
LCD15              .EQU        0003Fh
;
                   ...........
                   ...........
                   MOV.B       Table(Rx),Ry     ; Load segment information to temporary
                                                ; mem.
                                                ; (Ry) = 0000  0000  hfdb  geca
                   MOV.B       Ry,&LCDn         ; Note:
                                                ; All bits of an LCD memory byte are written
                   RRA         Ry               ; (Ry) = 0000  0000  0hfd  bgec
                   MOV.B       Ry,&LCDn+1       ; Note:
                                                ; All bits of an LCD memory byte are written
                   RRA         Ry               ; (Ry) = 0000  0000  00hf  dbge
                   MOV.B       Ry,&LCDn+2       ; Note:
                                                ; All bits of an LCD memory byte are written
                   RRA         Ry               ; (Ry) = 0000  0000  000h  fdbg
                   MOV.B       Ry,&LCDn+3       ; Note:
                                                ; All bits of an LCD memory byte are written
                   ...........
                   ...........
;
```

```
Table           .BYTE       a+b+c+d+e+f     ; displays "0"
                .BYTE       b+c             ; displays "1"
                ..........
                ..........
                .BYTE
                ..........
```

14.3 LCD Port Function

The large number of LCD common and segment lines, together with the fixed number of pins of the package version, could limit the degree of integration. To support applications which require a reduced number of segments, the signals LCDM5 to LCDM7 can switch the function from segment lines to output lines in groups of four bits. These outputs can be used with the application for various functions. Bits in the display memory define the logical state of the signals. The output signals are digitally switched, either near to ground GND, or near to supply voltage VCC.

The nomenclature convention for signals used as segment lines is Sxx and as port functions is Oxx. A pin is identified equally with the same xx representation. The letter S or O states the function of that pin.

LCDM7	LCDM6	LCDM5	Group0	Group1	Group2	Group3	Group4	Group5	Group6	Group7
0	0	0	S0-S1	O2-O5	O6-O9	O10-O13	O14-O17	O18-O21	O22-O25	O26-O29
0	0	1	S0-S1	S2-S5	O6-O9	O10-O13	O14-O17	O18-O21	O22-O25	O26-O29
0	1	0	S0-S1	S2-S5	S6-S9	O10-O13	O14-O17	O18-O21	O22-O25	O26-O29
0	1	1	S0-S1	S2-S5	S6-S9	S10-S13	O14-O17	O18-O21	O22-O25	O26-O29
1	0	0	S0-S1	S2-S5	S6-S9	S10-S13	S14-S17	O18-O21	O22-O25	O26-O29
1	0	1	S0-S1	S2-S5	S6-S9	S10-S13	S14-S17	S18-S21	O22-O25	O26-O29
1	1	0	S0-S1	S2-S5	S6-S9	S10-S13	S14-S17	S18-S21	S22-S25	O26-O29
1	1	1	S0-S1	S2-S5	S6-S9	S10-S13	S14-S17	S18-S21	S22-S25	S26-S29

Reset Condition

Figure 14.14: Groups of Segment and Output Lines

Note: Control bits

The control bits LCDM5 ... LCDM7 are reset with PUC.

The segment signals Sxx are part of the display driving signals and carry modulated voltage levels according to the time frame of the common lines.

The output signals Oxx selected for 'port' function are static signals. They take two digital level according to bits in the display memory. The logical state of the bits is taken from bit 0 to bit 3* for odd S-lines (n=3,5,.....) and from bit 4 to bit 7* for even S-lines (n=2,4,.....).

* Bits taken are dependent on the MUX rate.

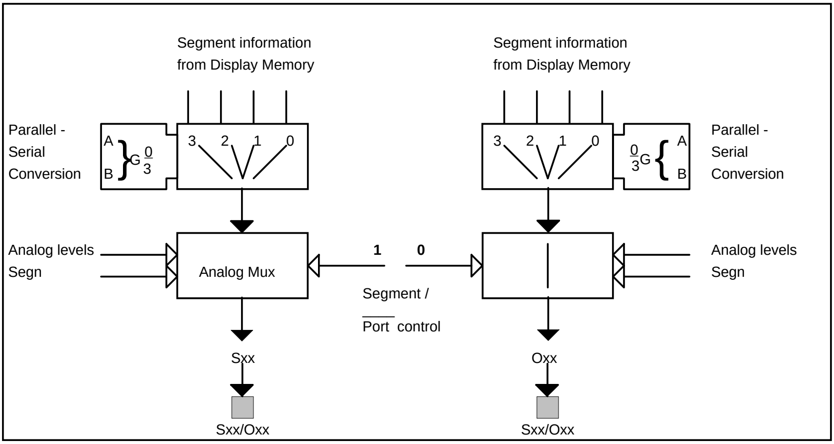


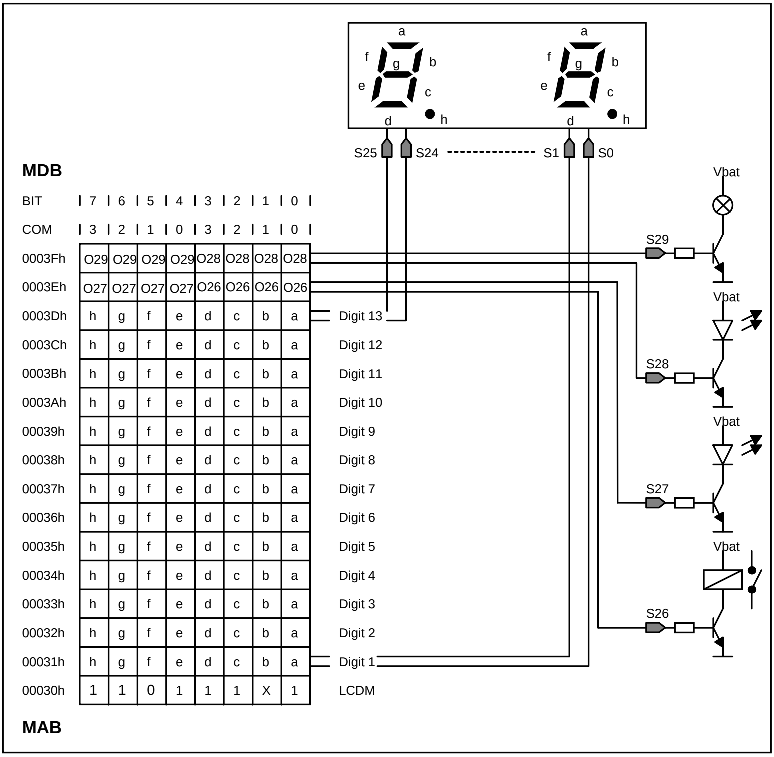
Figure 14.15: Segment Line or Output Line

The logical information of an output Oxx is defined in the display memory. Its location is either bit0 to bit3 or bit4 to bit7, depending on whether an odd or even assignment:

- xx = 2,4, ... 28: Oxx is defined with bit0 to bit3
- xx = 3,5, ... 29: Oxx is defined with bit4 to bit7

14.4 Application Example showing mixed LCD and Port Mode

The example uses the mixed mode: 4MUX LCD drive for 13 digits and one port group with four digital outputs.



MDB									
BIT	7	6	5	4	3	2	1	0	
COM	3	2	1	0	3	2	1	0	
0003Fh	O29	O29	O29	O29	O28	O28	O28	O28	
0003Eh	O27	O27	O27	O27	O26	O26	O26	O26	
0003Dh	h	g	f	e	d	c	b	a	Digit 13
0003Ch	h	g	f	e	d	c	b	a	Digit 12
0003Bh	h	g	f	e	d	c	b	a	Digit 11
0003Ah	h	g	f	e	d	c	b	a	Digit 10
00039h	h	g	f	e	d	c	b	a	Digit 9
00038h	h	g	f	e	d	c	b	a	Digit 8
00037h	h	g	f	e	d	c	b	a	Digit 7
00036h	h	g	f	e	d	c	b	a	Digit 6
00035h	h	g	f	e	d	c	b	a	Digit 5
00034h	h	g	f	e	d	c	b	a	Digit 4
00033h	h	g	f	e	d	c	b	a	Digit 3
00032h	h	g	f	e	d	c	b	a	Digit 2
00031h	h	g	f	e	d	c	b	a	Digit 1
00030h	1	1	0	1	1	1	X	1	LCDM

Figure 14.16: Application Example

> **Note: LCD port output**
>
> Any LCD port output is defined with four bits. All four bits of the group should have the same logical level, otherwise the output is not static. Assume O28 should be 'H', all the bit0 to bit3 are 'H'.

Software example to set O28, O29 should be unchanged

```
LCD15           .EQU     0003Fh
                BIS.B    00Fh,&LCD15
```

15 Analog-To-Digital Converter

Topic **Page**

Features of the A/D module:

- Eight Analog or Digital input channels
- Programmable (via external resistor REXT) current source on four analog pins
- Ratiometric or Absolute measurement
- Built in Sample-and-Hold
- End-Of-Conversion (EOC) interrupt flag
- ADAT register holds conversion results until next Start-Of-Conversion (SOC)
- Low power consumption
- Converts standalone without further CPU processing overhead
- Programmable 12-bit or 14-bit resolution
- Four programmable ranges give 14-bit dynamic range
- Fast conversion time
- Large supply voltage range
- Monotonic over the entire A/D conversion range

15.1 Overview

The (12+2)-bit Analog-to-Digital converter is a peripheral module, accessed by word instructions. The result of the converter is available on this 16-bit wide bus by reading the ADAT register. It must be noted that when a conversion is started, the bits as they are resolved by the converter are visible in the successive approximation register (SAR). They are available immediately at the ADAT register and are not cleared until the next conversion is initiated by setting the Start-Of-Conversion (SOC) bit in the ACTL register. Since the SAR is transparent to the MDB the conversion progress can be monitored by reading data via the read-only ADAT register. The SOC bit clears the SAR register for the new result as well as starting the clock of the A/D converter for another conversion.

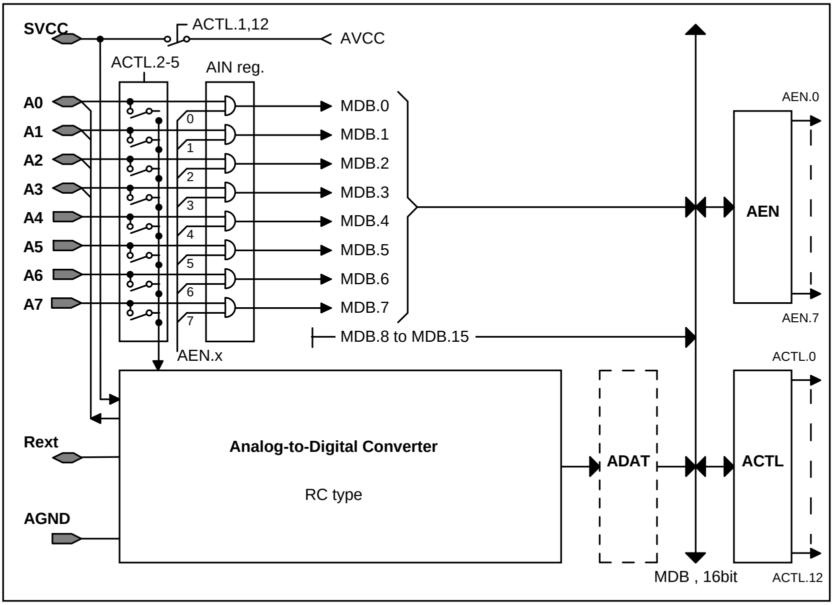


Figure 15.1: ADC Module Configuration

The module has eight individually selectable analog input channels that are multiplexed to the converter's input circuitry, such that a conversion can be made on any one of these channels at any time. Four of these channels, A0, A1, A2 and A3, are also configured as four current source outputs whose values can be programmed by an external resistor R_{ext}. Any of these current source outputs can be turned on (one at a time) to drive external sensors, in order to make ratiometric measurements. An absolute measurement can also be made to the accuracy of the reference voltage, by applying an external stable voltage source to the SVCC pin.

The eight channels can also be configured as analog or digital inputs. The digital data can be presented at all eight channels or individually selected channels, by writing to the respective bits of the AEN register. The digital data presented at the channels can be read from the AIN register. When a sensitive analog conversion is being done, any digital activity on adjacent channels will cause cross-talk and interference, leading to noise and incorrect output codes.

The converter has two modes of operation depending on the status of bit11 of the ACTL register, either a 12-bit or a (12+2)-bit conversion is possible. When the range of the input signal is known, two bits from the ACTL register can be used to define the range required with bit11 reset. The converter will sample the input, and then convert it to 12 bits of resolution, within any one of these four ranges. In this manual mode, this effectively yields a 14 bit dynamic range of operation for the converter. However in the Auto Mode, selected by setting bit11 of the ACTL register, the range is automatically selected by the converter to resolve effectively to 14 bits. The input is sampled twice, once for the 2-bit range selection and lastly for the remaining 12-bits of the conversion, to give a (12+2)-bit result. In both modes, when a conversion is completed (End-Of-Conversion EOC), the interrupt flag is set automatically to signal the microprocessor that a conversion has been completed. The EOC signal also disables the clock for the A/D converter, to conserve power until the next SOC bit is set.

Note: ADC, Start-of-Conversion

After starting a conversion it should always be completed before the next conversion is initiated. Otherwise, unpredictable conversion data will result.

The microprocessor core communicates to the A/D via the internal bus system, by applying the correct address for the module and supplying the required conditions for the ACTL and AEN registers. It reads the conversion results back via the ADAT registers.

Under power-down the whole Analog-to-Digital converter shuts down to stop current consumption. This is valid while SVCC is not externally driven. Upon a conversion start or power-up signal, the converter wakes up, but may take up to 6 µs to reach steady state conditions for an accurate conversion.

15.2 Analog-to-Digital Operation

15.2.1 A/D Conversion

After power-up, the ACTL register should be programmed to decide whether to make a ratiometeric or absolute measurement, and whether the range is to be manually or automatically selected. In manual mode, once the range bits have been selected these bits can not be changed during the conversion, as this will invalidate the results.

Setting the Start-of-Conversion (SOC) bit in the ACTL register activates the clock for the A/D converter for a new conversion to begin. The converter is based on a successive approximation technique utilizing a resistor array to resolve the M most significant bits (MSBs) first, and a switched capacitor array to resolve the remaining L least significant bits (LSBs).

The resistor array consisting of 2^M individually, equally weighted resistors forming the DAC, and the capacitor array consisting of L capacitors forms a charge redistribution A/D. The capacitors are binary-weighted; that is, they increase from the smallest value in powers of two. The number of capacitors corresponds to the range of the converter or L bits of the digital output code.

The sequence starts by selecting the analog channel of interest and sampling the analog input voltage onto the top plates of the capacitor array, the analog mux is then disconnected from the A/D and the analog input need not be present anymore after this sample period.

A successive approximation search is done on the resistor string to find the tap that corresponds to being within 2^L LSBs of VIN; this then gives the VH and VL voltages across one element and has resolved the M MSBs. The capacitor array then resolves this (VH-VL) difference voltage to L bits of resolution using a similar successive approximation search on the capacitor array starting with the MSB capacitor.

This switching procedure continues with the MSB or largest capacitor to the smallest (LSB) capacitor in the capacitor array, thereby the initial charge is redistributed among the capacitors. The particular setting of the switches both in the resistor array and in those connected to the bottom plates of the capacitors, has then induced a change on the top plate that is as close to the input voltage VIN as possible, and the switch settings then correspond to the binary code (12-bit or (12+2)-bit) that represents the fraction VIN/VREF.

The top plate voltage is monitored by a comparator with built-in input offset cancellation circuitry, which senses whether the input voltage is less than or greater than the voltage on the top plate, and generates a digital output which determines which way the successive approximation search is to be performed.

The smallest voltage change (LSB) occurs when the smallest capacitor is switched in, and this is the resolution of the converter, or VREF/2^n where n is the number of bits.

When this sequence is completed, the top plate voltage is as close to zero as the resolution of the converter allows and the LSB has been determined. An End-of-Conversion (EOC) signal is then sent to indicate that a 12-bit or (12+2)-bit conversion result is available for reading from the ADAT register for further processing.

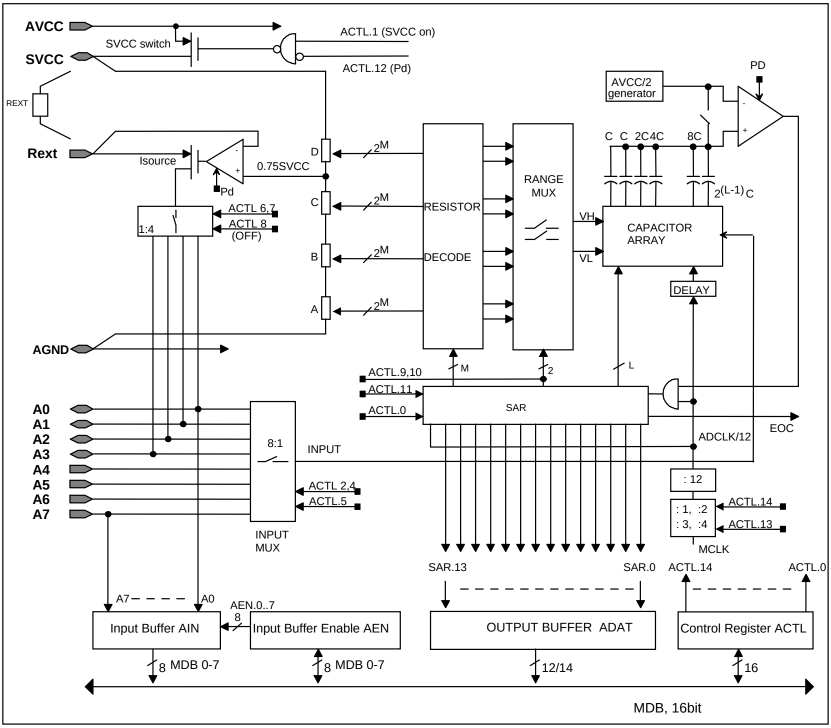


Figure 15.2: ADC Schematic

A/D conversion timing

After the ADC module has been activated with the Power Down bit reset, at least 6 µs must elapse before a new conversion is attempted, in order to allow the correct internal biases to be established.

The A/D converter always runs at a clock rate set to one twelfth of the ADCLK. The frequency of ADCLK should be chosen to meet the conversion time defined in the actual electrical characteristics. If the ADCLK is too fast an accurate conversion to 12 bits cannot be guaranteed, due to internal time constants associated with sampling the analog input and the conversion network. If the ADCLK is too slow a conversion accurate to 12 bits cannot be guaranteed, due to charge loss within the capacitor array of the A/D, even if the input signal is valid and steady for the required acquisition time.
The correct frequency for ADCLK can be selected by two bits (ADCLK) in the control register ACTL. The applied MCLK clock signal is then divided by factor 1, 2, 3 or 4.

Sampling the analog input signal takes twelve ADCLK clock pulses and the 12-bit conversion takes another twelve times seven (84) ADCLK clock cycles. This is true for a 12-bit conversion with pre-selected range; ACTL.11 is reset. Altogether the 12-bit conversion takes 96 ADCLK cycles.

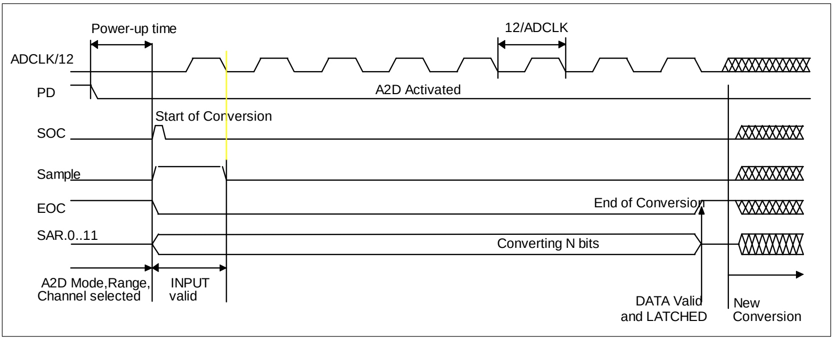


Figure 15.3: ADC Timing, 12-bit conversion

When ACTL.11 is set a (12+2)-bit conversion with auto range selected takes place. The analog input signal is sampled twice, each taking twelve ADCLK clock pulses. After the first sampling of the input signal, the range conversion is done and takes 24 ADCLK clocks. After the second sampling of the input signal, the 12-bit conversion is done and takes another 84 (12*7) ADCLK clock cycles. Altogether the (12+2)-bit conversion takes 132 ADCLK cycles.

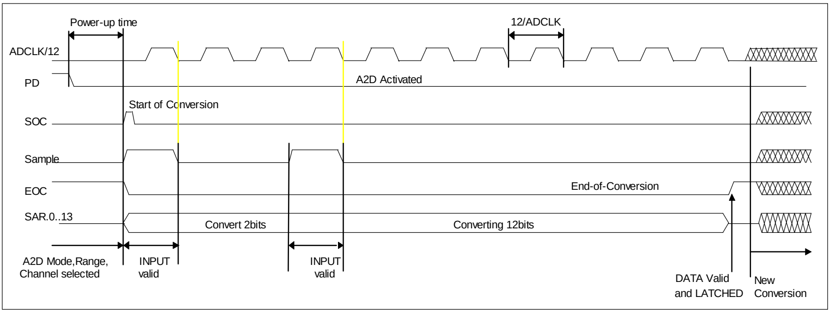


Figure 15.4: ADC Timing, (12+2)-bit conversion

The input signal must be valid and steady during this sampling period in order to obtain an accurate conversion. It is also desirable not to have any digital activity on any adjacent digital channels during the whole of the conversion period to ensure that errors due to supply glitching and ground bounce or cross-talk interference do not corrupt the results.

The A/D converter uses the charge redistribution method and thus when the inputs are internally switched to sample the input, the switching action causes displacement currents to flow into and out of the analog inputs.

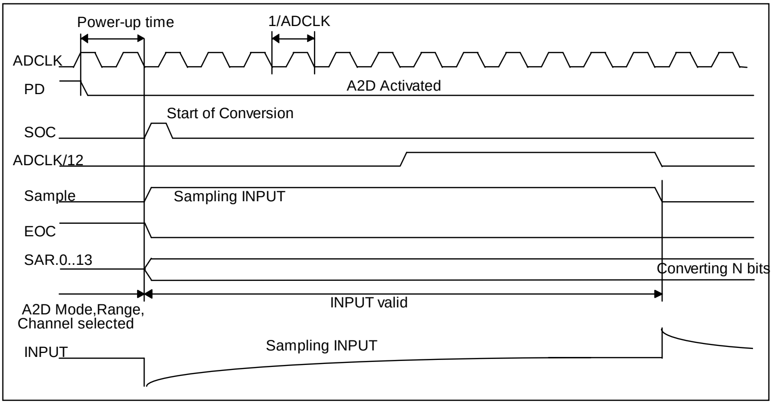


Figure 15.5: ADC, input sampling timing

These current spikes or transients occur at the leading and falling edge of the internal sample pulse, and quickly decay and settle before causing any problems, because the time constant is less than that given by the internal 'effective RC'. Internally the analog inputs see a nominal RC of effectively a 32 pF (C-array) capacitor in series with a 2-kΩ resistor (Ron of switches). However if the external dynamic source impedance is large, then these transients may not settle within the allocated sampling time to ensure 12 or (12+2) bits of accuracy.

15.2.2 A/D Interrupt

When an A/D conversion is complete, the EOC signal goes high and activates the A/D interrupt circuit by setting the interrupt flag ADIFG, which informs the rest of the system when a conversion has been completed. An interrupt is requested when the enable bit ADIE is set.

15.2.3 A/D Ranges

One of four ranges can be selected to yield a result with 12 bits of resolution within any one given range. If the bit ACTL.11 is reset, effectively 14 bits of dynamic range are possible. The range is defined prior to conversion start with the bits ACTL.9 and ACTL.10. However if bit11 is set, then the converter will find the appropriate range during the conversion by sampling the input twice, once for the range selection and secondly for the 12-bit conversion, thereby giving overall a (12+2)-bit conversion result.

The ranges are:

$0.00\text{xVREF} \leq \text{VIN} < 0.25\text{xVREF}$ Range A
$0.25\text{xVREF} \leq \text{VIN} < 0.50\text{xVREF}$ Range B
$0.50\text{xVREF} \leq \text{VIN} < 0.75\text{xVREF}$ Range C
$0.75\text{xVREF} \leq \text{VIN} < 1.00\text{xVREF}$ Range D

where VREF is the voltage at the SVCC pin, either applied externally or that voltage (close to AVCC) derived by closing the SVCC switch with bit12 of the ACTL register.

After the proper range has been selected, the input channel, selected by the appropriate bits in the control register, is connected to the input of the converter. The A/D converter processes the signal at the selected input channel and the software can then access the result of the conversion via the ADAT register.

The digital code (Decimal) expected within any one range is:

$$N_{typ} = \text{INT} \left| \frac{\text{VIN} \ \text{x} \ 2^{14}}{\text{VREF}} - 2^{13} \ \text{x ACTL.10} \ - \ 2^{12} \ \text{x ACTL.9} \right|$$

where ACTL.10 and ACTL.9 are bits 10 and 9 respectively in the ACTL register.

Thus for a 12-bit conversion:

$0000h \leq N \leq 0FFFh$ Range A
$0000h \leq N \leq 0FFFh$ Range B
$0000h \leq N \leq 0FFFh$ Range C
$0000h \leq N \leq 0FFFh$ Range D

and a (12+2)-bit conversion:

$0000h \leq N \leq 3FFFh$

Note: ADC Offset voltage

Any offset voltage (Vio) due to voltage drops at the bottom or top of the resistor array, caused by parasitic impedances to the SVCC pin or the ground AGND pin, will distort the digital code output and the formula.

15.2.4 A/D Current Source

One of four analog I/Os can be used for the current source output. The current out of the current source (Isource) can be programmed by an external resistor REXT and is then available on pins A0, A1, A2 and A3, with the value:

I_{source} = (0.25 x SVCC)/R_{ext} where SVCC is the voltage at pin SVCC and R_{ext} is the external resistor between pins SVCC and R_{ext}.

Therefore for ratiometric measurements the voltage (Vin) developed at the input to the channel with the resistive elements (Channels A0, A1, A2 and A3 only) is:

Vin = (0.25 x SVCC) x (R_{sens}/R_{ext}) where R_{sens} is the external resistive element.

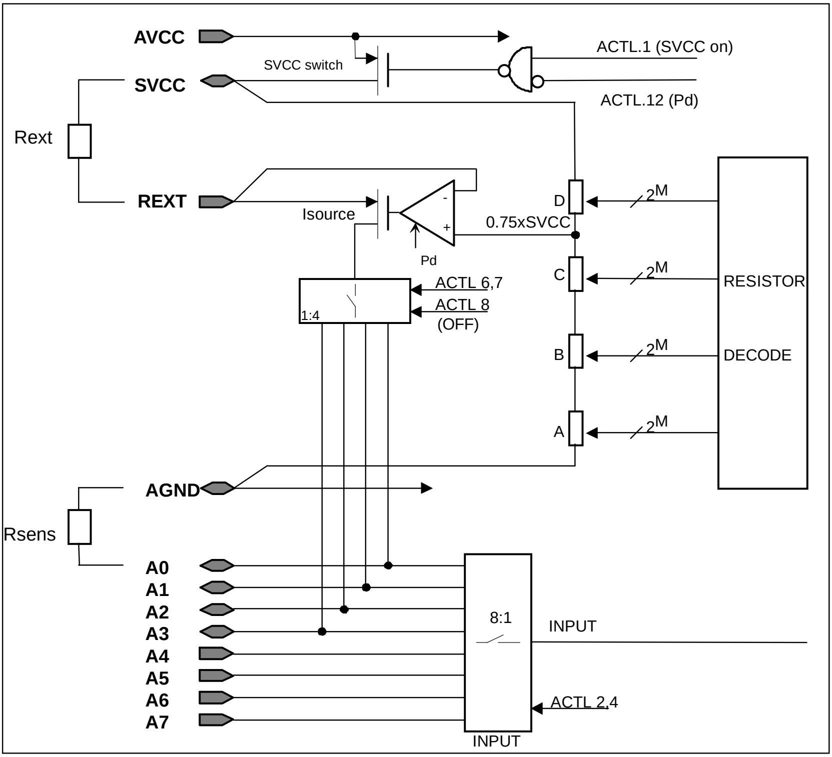


Figure 15.6: A/D Current Source

When the A/D converter is used in conjunction with resistive elements in sensor applications, current sources of precise value are required, so that the input signal can be referred back to the supply voltage or voltage reference in the same manner as the current source, thereby allowing a ratiometric measurement to take place, independently of the accuracy of the stable reference.

15.2.5 Analog Inputs and Multiplexer

Analog Inputs

The analog input signal is sampled onto an internal capacitor and held during the conversion. The charge of the capacitance is supplied by the source and the time to charge it up is defined by the sampling time of twelve ADCLK clocks. Therefore, the external source resistances and dynamic impedances must be limited, so that the RC time constant is short enough to allow the analog inputs to completely settle within the allocated sampling time to a 12-bit accuracy. This is typically a time constant less than $0.8/f_{ADCLK}$. High source impedances have an adverse effect on the accuracy of the converter, not only due to the RC settling behavior, but also due to voltage drops at the inputs due to leakage current or averaged DC input currents (due to input switching currents). Typically for a 12-bit converter, the error in LSBs due to leakage current is:

Error(LSBs) = 4*(µA of leakage current)*(kΩ of source resistance)/(volt of VREF)

Example: 50 nA leakage, 10 kΩ of source resistance, 3-V VREF gives 0.7 LSBs of error.

This also applies equally to the output impedance of the voltage reference source VREF as well. It must be low enough to enable the transients to settle within $(0.2/_{ADCLK})$ seconds, and generate leakage current induced errors of <<1LSB.

Analog Multiplexer

The analog multiplexer selects one of eight single-ended input channels, as determined by the bits in the ACTL register. It is based on a 'T-switch' to minimize the coupling between channels corrupting the analog input. Channels that are not selected are isolated from the A/D and the intermediate node connected to the analog ground AGND so that the stray capacitance is 'grounded' to eliminate cross-talk.

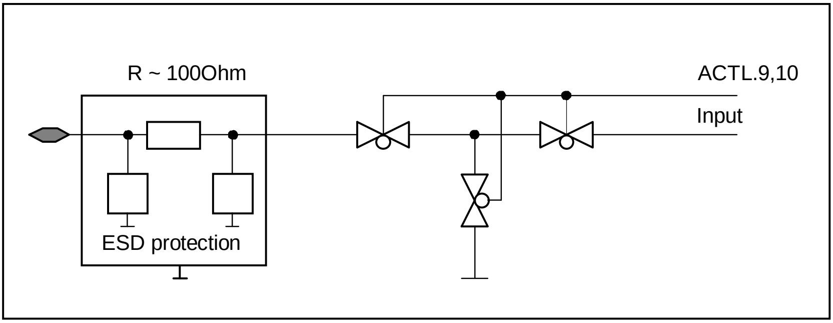


Figure 15.7: Analog Multiplexer

Cross-talk exists because there is always some parasitic coupling capacitance across the switch and between switches. This can take several forms, such as coupling from the input to the output of an 'OFF' switch, or coupling from an 'OFF' analog input channel to the output of another adjacent 'ON' output channel, causing errors to creep into the digital code output. So for high accuracy conversions, cross-talk interference must be minimized altogether, by shielding and other well-known printed circuit board (PCB) layout techniques.

15.2.6 A/D Grounding and Noise Considerations

As with any high resolution converter (≥ 12 bits) care and special attention has to be paid to the printed circuit board layout and the grounding scheme, to eliminate ground loops and any unwanted parasitic components/effects and noise. There are standard techniques which are well documented in application notes that address these issues.

Ground Loops are formed when the return current from the resistor divider of the A/D flows through tracks that are common with other analog or digital circuitry. If care is not taken, this current can generate small unwanted offset voltages that can add to or subtract from the reference or input voltages of the A/D converter. One way to avoid ground loops is to use the scheme where a 'star connection' is used for the AGND; in this way the ground current or reference currents do not flow through any common input leads, eliminating any error voltages.

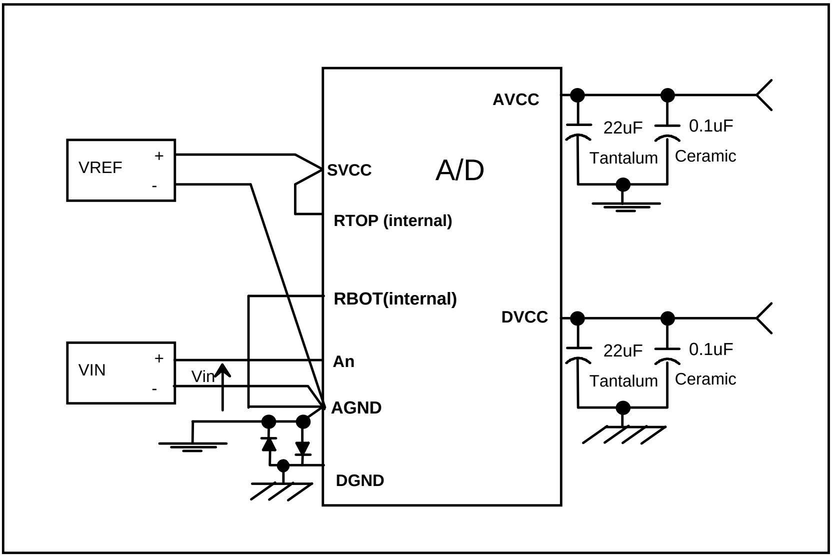


Figure 15.8: A/D Grounding and Noise Considerations

The digital ground DGND and analog ground AGND can also be star connected together, but if separate supplies are used then two reverse biased diodes limit the voltage difference to less than ±700mV.

Furthermore ripple and noise spikes on the power supply lines due to digital switching or switching power supplies are especially troublesome.

Normally the internal noise is very small and the total input referred noise is far less than one LSB, so the output code is fairly stable. However, as noise couples into the device via the supply and ground changes, the noise margin reduces and code uncertainty and jitter can creep in, which may mean taking several readings to average out the noise effects. Another consequence is that, as the reference voltage SVCC or VREF is reduced, the absolute value of the LSB also reduces, and therefore the noise becomes even more dominant as the noise margin reduces. Thus a clean, totally noise-free setup becomes of even more paramount importance to achieve the accuracy desired.

Adding carefully placed bypass capacitors returned to the respective ground planes helps in stabilizing the supply current and minimizing the 'noise' .

15.2.7 A/D Converter Input and Output Pins

Input Pins

There are two different types of input signals; the inputs of analog signals A0, A1, A2, A3, A4, A5, A6, A7 and REXT, SVCC.

The input signals coming from channel A0 to A7 can be treated as analog signals that should be handled with the A/D converter, or as digital inputs to be read into the processing unit.

An external resistor between REXT and SVCC determines the amount of current flowing through an activated current source operation. The pin SVCC is then used as an output or input. The SVCC pin is an input when the internal SVCC switch is off and the Vref is applied externally. It is an output when the internal SVCC switch is on.

Output Pins

There are two different types of output signals, the outputs A0, A1, A2, A3 and the SVCC.
Current will flow out of one of the analog pins A0, A1, A2, A3 if the current source function is selected. The SVCC pin will then have a voltage just below the AVCC when the SVCC switch is on.

Supply Pins

There are four supply pins to split the digital and the analog current paths:

AVCC,DVCC,AGND,DGND.

Some of the MSP430 family members will have all four supply pins bonded out for high analog resolution, while others will have analog and digital VCC and/or GND rails internally connected.

15.3 ADC Control Registers

Four control registers are implemented:

Register	short form	Register type	Address	Initial state
● Input register:	AIN	Type of read only	0110h	- - -
● Input enable register:	AEN	Type of read/write	0112h	reset
● ADC control register:	ACTL	Type of read/write	0114h	→see figure
● Reserved				0116h
● ADC data register:	ADAT	Type of read	0118h	- - -

All registers may be accessed by any instruction subject to register read/write restrictions.

Input register AIN

The signals at the pins A0 to A7 can be signals from an analog source or a digital source. The value of digital sources can be read with access to the input register. The reading of the digital sources is enabled by a selection done in the input enable register.

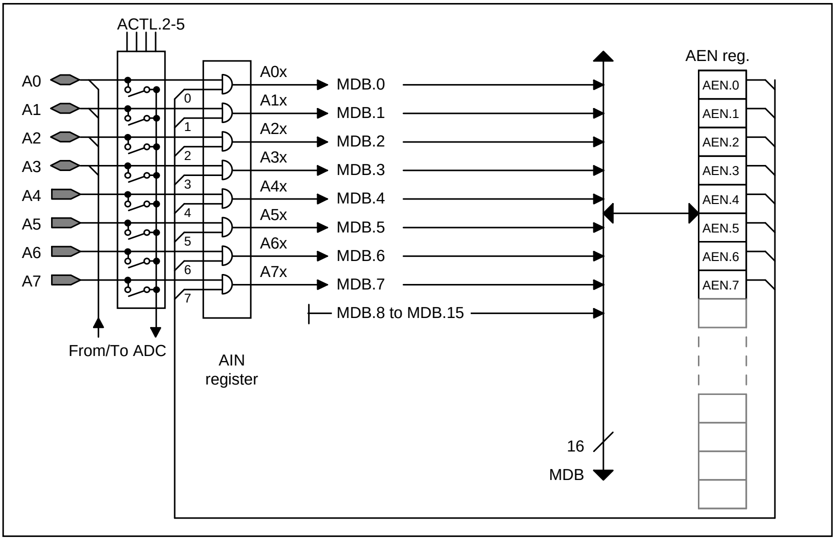


Figure 15.9: ADC Input Register, Input Register Enable

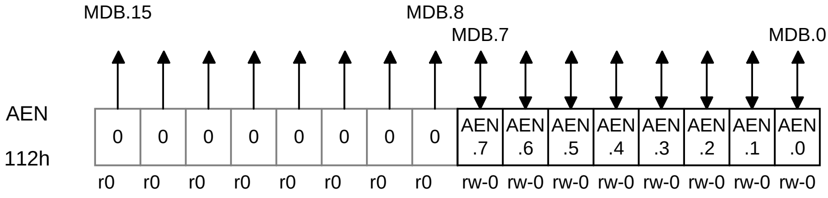
The input register AIN is a read only register connected to the 16-bit MDB. The LowByte of the register is implemented and MDB.0 to MDB.7 corresponds to A0 to A7. The HighByte of the register is read as 00h.

	MDB.15							MDB.8	MDB.7							MDB.0
AIN 110h	0	0	0	0	0	0	0	0	AIN .7	AIN .6	AIN .5	AIN .4	AIN .3	AIN .2	AIN .1	AIN .0
	r0	r0	r0	r0	r0	r0	r0	r0	r	r	r	r	r	r	r	r
									A7x	A6x	A5x	A4x	A3x	A2x	A1x	A0x

The signal at the corresponding input is logically gated with the appropriate enable signal, Ax .AND. AEN.x. Unselected (disabled) bits are read as '0'.

Input Enable Register AEN

The input enable register AEN is a read/write register connected to the 16-bit MDB. The LowByte of the register is implemented and MDB.0 to MDB.7 corresponds to A0 to A7. The HighByte of the register is read as 00h.



The input enable register bits control the definition of the individual bit:

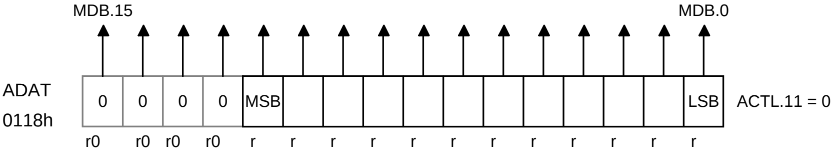
AEN.x = 0 : Analog input. The bit read at an access to the AIN register is 0.

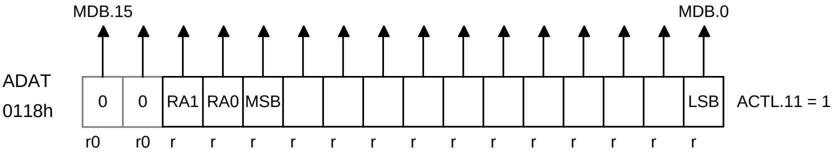
AEN.x = 1 : Digital input. The bit read at an access to the AIN represents the logical level at the appropriate pin.

The initial state of all bits is reset.

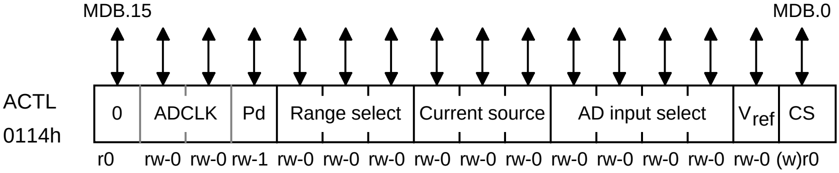
ADC Data Register ADAT

The ADC data register holds the result of the analog-to-digital conversion. The conversion data are correct in the register at the end of a conversion and correct until another conversion is started with setting SOC bit.





ADC Control Register ACTL



Bit 0, ACTL.0 : Convert Start

Bit 1, ACTL.1 : Source of Vref.

ACTL.1 = 0 : Switch SVCC is off. The output pin SVCC is not connected to VCC. The reference voltage of the ADC should be supplied from an external source.

ACTL.1 = 1 : Switch SVCC is on. The output pin SVCC is connected to VCC. The reference voltage of the ADC should not be supplied from an external source.

Bit 2-5, ACTL.2-.5 : AD input select.

These bits select the channel used for conversion. Channels should be changed only after completion of a conversion. Changing the channel while a conversion is active invalidates the conversion in progress.

ACTL.5	ACTL.4	ACTL.3	ACTL.2	Channel
0	0	0	0	A0
0	0	0	1	A1
0	0	1	0	A2
0	0	1	1	A3
0	1	0	0	A4
0	1	0	1	A5
0	1	1	0	A6
0	1	1	1	A7
1	X	X	X	NONE

Bit 6-8, ACTL.6-.8 : AD current source output select.

These bits select the channel used for output of the current source. Channels should be changed only after completion of a conversion. Changing the channel while a conversion is active invalidates the conversion in progress.

ACTL.8	ACTL.7	ACTL.6	Channel
0	0	0	A0
0	0	1	A1
0	1	0	A2
0	1	1	A3
1	X	X	NONE

Bit9-11, ACTL.9,.11: Range Select

These bits must not be changed once conversion has started. Any manipulation of these bits during a conversion will result in incorrect conversion data in ADAT.

ACTL.11	ACTL.10	ACTL.9	Range
0	0	0	A
0	0	1	B
0	1	0	C
0	1	1	D
1	X	X	Auto

Bit11, ACTL.11: Range Select Mode

ACTL.11 = 0 : The range select bits ACTL.9 and ACTL.10 have to be applied for manual range select.

ACTL.11 = 1 : The automatic range selection for a (12+2)-bit conversion is active. Since the manual range select is inactive the states of the range select bits ACTL.9 and ACTL.10 are “don't care”.

Bit12, ACTL.12: Power Down (Pd)

ACTL.12 = 1: SVCC switch is off,
COMPARATOR is powered down
Current source is off

Bits 13, 14 ADCLK

The clock frequency of the ADC should be adjusted to the maximum frequency described in the device’s data-sheet.

ACTL.14	ACTL.13	ADCLK
0	0	MCLK
0	1	MCLK/2
1	0	MCLK/3
1	1	MCLK/4

Bits 13-15 Reserved.

Test underflow and overflow condition by software

; Since the ADAT register is implemented in the 16-bit peripheral address space and
; integrated to handle data in word mode, no implementation of overflow or underflow
; error detection is mandatory. It is reduced to simple commands in program flow:

```
; Bit11 of the ACTL register is reset, a 12-bit conversion was active
;
    CMP     #0,&ADAT            ; test for 12-bit ADC underflow
    JEQ     UndFlow             ; Yes, continue with underflow handling

    CMP     #0FFFh,&ADAT        ; test for 12-bit ADC overflow
;
; The MSBits, not implemented in the converter's hardware are read as 0s
    JEQ     OverFlow            ; Yes, continue with overflow handling



; Bit11 of the ACTL register is set, a (12+2)-bit conversion was active
; The conversion range should be limited to Range A to C
;
    CMP     #0,&ADAT            ; test for (12+2)-bit ADC underflow
    JEQ     UndFlow             ; Yes, continue with underflow handling

    CMP     #2FFFh,&ADAT        ; test only for Range A to C overflow
;
; The MSBits, not implemented in the converter's hardware are read as 0s
    JHS     OverFlow            ; Yes, continue with overflow handling
```

16 Miscellaneous Modules

Topic **Page**

16.1 Crystal Oscillator

All elements for crystal operation are integrated into the MSP430 - no additional external components are necessary for operation. Since the oscillator is designed for ultra-low power dissipation the PWB layout should provide short connections between the crystal and the MSP430 device.

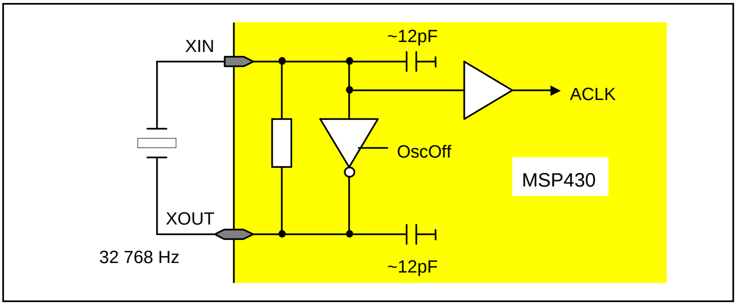
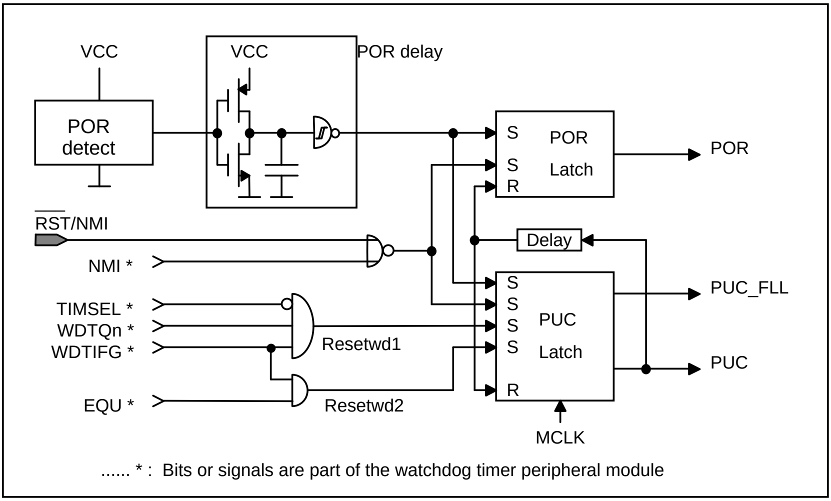


Figure 16.1: Crystal Oscillator schematic

When OscOff mode is selected the ACLK signal is held to high.

16.2 Power-on Circuitry

The power-on circuitry is part of the system reset scheme, and consists of two parts: the power-on reset detection, and the power-on reset delay. The output of the POR delay is fed into the POR latch and the PUC latch to set both latches, in order to supply the system with the reset condition.



...... * : Bits or signals are part of the watchdog timer peripheral module

Figure 16.2: Power-on reset and Power-up clear schematic

When the VCC supply provides a fast VCC rise time, the POR delay gives enough active time on the POR signal to allow it to initialize the circuit correctly after power-up.

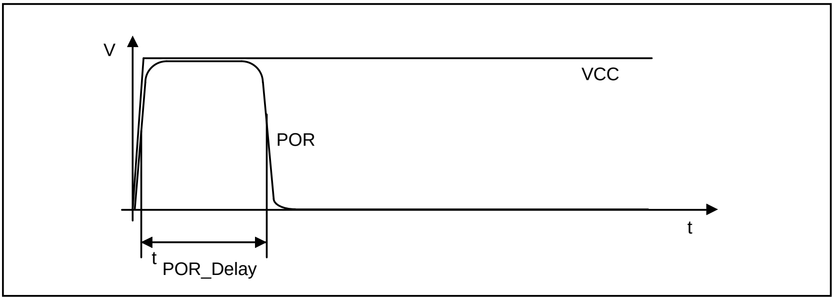


Figure 16.3: Power-on reset timing on fast VCC rise time

When the VCC supply provides a 'slow' VCC rise condition the POR detect defines the POR signal to allow it to initialize the circuit correctly after power-up.

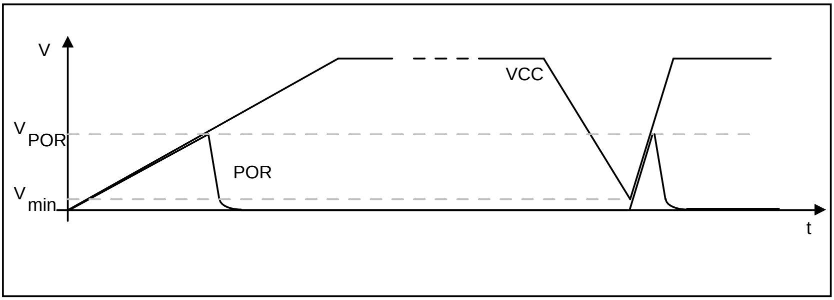


Figure 16.4: Power-on reset timing on slow VCC rise time

The supply voltage VCC should fall below V_{min} to ensure another POR signal occurs with the next increase in supply voltage. If V_{CC} does not fall below V_{min} a POR will not be generated and power-up conditions will not be set properly.

16.3 Crystal Buffer Output

The frequency of the buffer output is selected via the control register CBCTL.

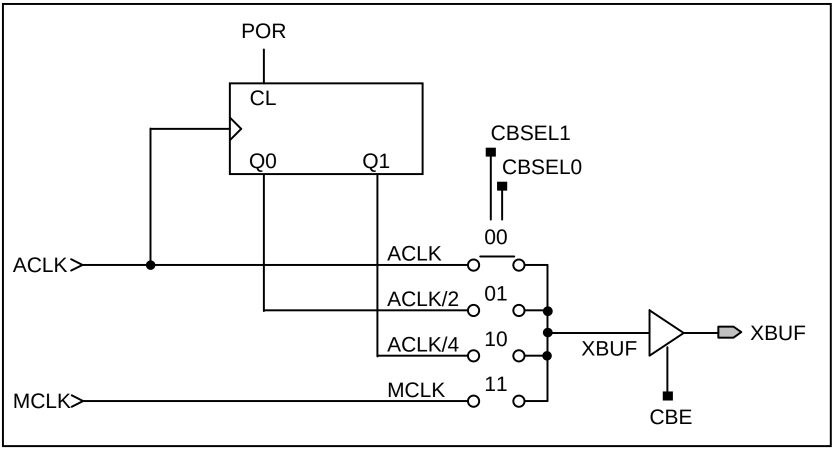


Figure 16.5: Schematic of Crystal Buffer

The control register CBCTL of the clock buffer output peripheral has bits that control the frequency applied to pin XBUF, and one bit that controls the 3-state condition of the output buffer.

The divider runs with the minimum of logic necessary for correct operation. For example, it is halted when ACLK or MCLK is selected or if the CBE bit is set.

The three bits in the control register CBSEL1, CBSEL0 and CBE are reset with POR signal. The POR signal is active either during switching on V_{CC} or when , RST/NMI -pin is tied to V_{SS} when reset function is selected.

CBCTL 053h	---	---	---	---	---	CBSEL 1	CBSEL 0	CBE
						w-(0)	w-(0)	w-(0)

Bit 0: The bit CBE controls the 3-state condition of the output buffer.
CBE = 1: Output buffer enabled
CBE = 0: Output buffer disabled
During power-on reset (POR) the output buffer is always disabled. External components are not supplied with the selected frequency.

Bit 1,2: The bits CBSEL1 and CBSEL0 select the frequency that can be put onto output pin XBUF.

CBSEL1	CBSEL0	XBUF1	
0	0	ACLK	← State after POR
0	1	ACLK/2	
1	0	ACLK/4	
1	1	MCLK	

A. Peripheral File Map

This appendix summarizes the Peripheral File (PF) and control bit information into a single location for reference.

Each PF register is presented as a row of boxes containing the control or status bits belonging to the register. The register symbol (e.g. P0IN) and the PF hex address are to the left of each register.

The accessibility and/or hardware conditions of each bit are indicated below each bit symbol, with the following definitions:

● rw:	read/write
● r:	read only
● r0:	read as '0'
● r1:	read as '1'
● w:	write only
● w0:	write a '0'
● w1:	write a '1'
● (w):	no register bit implemented; writing a '1' will result in a pulse the register bit is always read as '0'
● h0:	cleared by hardware
● h1:	set by hardware
● -0,-1:	condition after PUC signal active (Reset + WDT conditions).
● -(0),-(1):	condition after POR signal active (Reset condition).

Special function register, byte access

Bit # -	7	6	5	4	3	2	1	0
000Fh								
Module enable 2, ME2 0005h							UTXE rw-0	URXE rw-0
Module enable 1, ME1 0004h								
Interrupt flag 2, IFG2 0003h	BTIFG rw					ADIFG rw-0	UTXIFG rw-0	URXIFG rw-0
Interrupt flag 1, IFG1 0002h				NMIIFG rw-0	P0IFG.1 rw-0	P0IFG.0 rw-0	OFIFG rw-1	WDTIFG rw 1)
Interrupt enable 2,IE2 0001h	BTIE rw-0				3) rw-0	2) rw-0	UTXIE rw-0	URXIE rw-0
Interrupt enable 1, IE1 0000h					P0IE.1 rw-0	P0IE.0 rw-0	OFIE rw-0	WDTIE rw-0

1) The WDTIFG is reset on POR signal and set with WDT overflow or WDT password violation.
2) Configuration ‘320: ADIE for 12+2b ADC (type: rw-0)
Configuration ‘310: TPIE for Timer/Port Module (type: rw-0)
3) Configuration ‘320, ‘330: TPIE for Timer/Port Module (type: rw-0)

Digital I/O frame, byte access

Bit # -	7	6	5	4	3	2	1	0
Direction reg., P4SEL 001Fh	P4SEL.7 rw-0	P4SEL.6 rw-0	P4SEL.5 rw-0	P4SEL.4 rw-0	P4SEL.3 rw-0	P4SEL.2 rw-0	P4SEL.1 rw-0	P4SEL.0 rw-0
Direction reg., P4DIR 001Eh	P4DIR.7 rw-0	P4DIR.6 rw-0	P4DIR.5 rw-0	P4DIR.4 rw-0	P4DIR.3 rw-0	P4DIR.2 rw-0	P4DIR.1 rw-0	P4DIR.0 rw-0
Output reg., P4OUT 001Dh	P4OUT.7 rw	P4OUT.6 rw	P4OUT.5 rw	P4OUT.4 rw	P4OUT.3 rw	P4OUT.2 rw	P4OUT.1 rw	P4OUT.0 rw
Input register, P4IN 001Ch	P4IN.7 r	P4IN.6 r	P4IN.5 r	P4IN.4 r	P4IN.3 r	P4IN.2 r	P4IN.1 r	P4IN.0 r
Direction reg., P3SEL 001Bh	P3SEL.7 rw-0	P3SEL.6 rw-0	P3SEL.5 rw-0	P3SEL.4 rw-0	P3SEL.3 rw-0	P3SEL.2 rw-0	P3SEL.1 rw-0	P3SEL.0 rw-0
Direction reg., P3DIR 001Ah	P3DIR.7 rw-0	P3DIR.6 rw-0	P3DIR.5 rw-0	P3DIR.4 rw-0	P3DIR.3 rw-0	P3DIR.2 rw-0	P3DIR.1 rw-0	P3DIR.0 rw-0
Output reg., P3OUT 0019h	P3OUT.7 rw	P3OUT.6 rw	P3OUT.5 rw	P3OUT.4 rw	P3OUT.3 rw	P3OUT.2 rw	P3OUT.1 rw	P3OUT.0 rw
Input register, P3IN 0018h	P3IN.7 r	P3IN.6 r	P3IN.5 r	P3IN.4 r	P3IN.3 r	P3IN.2 r	P3IN.1 r	P3IN.0 r
0017h								
0016h								
Interrupt Enable, P0IE 0015h	P0IE.7 rw-0	P0IE.6 rw-0	P0IE.5 rw-0	P0IE.4 rw-0	P0IE.3 rw-0	P0IE.2 rw-0	*) r0	*) r0
Int. Edge Sel., P0IES 0014h	P0IES.7 rw	P0IES.6 rw	P0IES.5 rw	P0IES.4 rw	P0IES.3 rw	P0IES.2 rw	P0IES.1 rw	P0IES.0 rw
Interrupt Flags, P0IFG 0013h	P0IFG.7 rw-0	P0IFG.6 rw-0	P0IFG.5 rw-0	P0IFG.4 rw-0	P0IFG.3 rw-0	P0IFG.2 rw-0	*) r0	*) r0
Direction reg., P0DIR 0012h	P0DIR.7 rw-0	P0DIR.6 rw-0	P0DIR.5 rw-0	P0DIR.4 rw-0	P0DIR.3 rw-0	P0DIR.2 rw-0	P0DIR.1 rw-0	P0DIR.0 rw-0
Output reg., P0OUT 0011h	P0OUT.7 rw	P0OUT.6 rw	P0OUT.5 rw	P0OUT.4 rw	P0OUT.3 rw	P0OUT.2 rw	P0OUT.1 rw	P0OUT.0 rw
Input register, P0IN 0010h	P0IN.7 r	P0IN.6 r	P0IN.5 r	P0IN.4 r	P0IN.3 r	P0IN.2 r	P0IN.1 r	P0IN.0 r

*) These interrupt enable bits and flags are included in the SFR frame.

Digital I/O frame, byte access

Bit # -	7	6	5	4	3	2	1	0
002Fh								
Direction reg., P2SEL 002Eh	P2SEL.7 rw-0	P2SEL.6 rw-0	P2SEL.5 rw-0	P2SEL.4 rw-0	P2SEL.3 rw-0	P2SEL.2 rw-0	P2SEL.1 rw-0	P2SEL.0 rw-0
Interrupt Enable, P2IE 002Dh	P0IE.7 rw-0	P2IE.6 rw-0	P2IE.5 rw-0	P2IE.4 rw-0	P2IE.3 rw-0	P2IE.2 rw-0	P2IE.1 rw-0	P2IE.0 rw-0
Int. Edge Sel., P2IES 002Ch	P2IES.7 rw	P2IES.6 rw	P2IES.5 rw	P2IES.4 rw	P2IES.3 rw	P2IES.2 rw	P2IES.1 rw	P2IES.0 rw
Interrupt Flags, P2IFG 002Bh	P2IFG.7 rw-0	P2IFG.6 rw-0	P2IFG.5 rw-0	P2IFG.4 rw-0	P2IFG.3 rw-0	P2IFG.2 rw-0	P2IFG.1 rw-0	P2IFG.0 rw-0
Direction reg., P2DIR 002Ah	P2DIR.7 rw-0	P2DIR.6 rw-0	P2DIR.5 rw-0	P2DIR.4 rw-0	P2DIR.3 rw-0	P2DIR.2 rw-0	P2DIR.1 rw-0	P2DIR.0 rw-0
Output reg., P2OUT 0029h	P2OUT.7 rw	P2OUT.6 rw	P2OUT.5 rw	P2OUT.4 rw	P2OUT.3 rw	P2OUT.2 rw	P2OUT.1 rw	P2OUT.0 rw
Input register, P2IN 0028h	P2IN.7 r	P2IN.6 r	P2IN.5 r	P2IN.4 r	P2IN.3 r	P2IN.2 r	P2IN.1 r	P2IN.0 r
0027h								
Direction reg., P1SEL 0026h	P1SEL.7 rw-0	P1SEL.6 rw-0	P1SEL.5 rw-0	P1SEL.4 rw-0	P1SEL.3 rw-0	P1SEL.2 rw-0	P1SEL.1 rw-0	P1SEL.0 rw-0
Interrupt Enable, P1IE 0025h	P0IE.7 rw-0	P1IE.6 rw-0	P1IE.5 rw-0	P1IE.4 rw-0	P1IE.3 rw-0	P1IE.2 rw-0	P1IE.1 rw-0	P1IE.0 rw-0
Int. Edge Sel., P1IES 0024h	P1IES.7 rw	P1IES.6 rw	P1IES.5 rw	P1IES.4 rw	P1IES.3 rw	P1IES.2 rw	P1IES.1 rw	P1IES.0 rw
Interrupt Flags, P1IFG 0023h	P1IFG.7 rw-0	P1IFG.6 rw-0	P1IFG.5 rw-0	P1IFG.4 rw-0	P1IFG.3 rw-0	P1IFG.2 rw-0	P1IFG.1 rw-0	P1IFG.0 rw-0
Direction reg., P1DIR 0022h	P1DIR.7 rw-0	P1DIR.6 rw-0	P1DIR.5 rw-0	P1DIR.4 rw-0	P1DIR.3 rw-0	P1DIR.2 rw-0	P1DIR.1 rw-0	P1DIR.0 rw-0
Output reg., P1OUT 0021h	P1OUT.7 rw	P1OUT.6 rw	P1OUT.5 rw	P1OUT.4 rw	P1OUT.3 rw	P1OUT.2 rw	P1OUT.1 rw	P1OUT.0 rw
Input register, P1IN 0020h	P1IN.7 r	P1IN.6 r	P1IN.5 r	P1IN.4 r	P1IN.3 r	P1IN.2 r	P1IN.1 r	P1IN.0 r

LCD register frame, byte access

Bit # -	7	6	5	4	3	2	1	0
LCD Memory 15 003Fh	S29C3 rw	S29C2 rw	S29C1 rw	S29C0 rw	S28C3 rw	S28C2 rw	S28C1 rw	S28C0 rw
LCD Memory 14 003Eh	S27C3 rw	S27C2 rw	S27C1 rw	S27C0 rw	S26C3 rw	S26C2 rw	S26C1 rw	S26C0 rw
LCD Memory 13 003Dh	S25C3 rw	S25C2 rw	S25C1 rw	S25C0 rw	S24C3 rw	S24C2 rw	S24C1 rw	S24C0 rw
LCD Memory 12 003Ch	S23C3 rw	S23C2 rw	S23C1 rw	S23C0 rw	S22C3 rw	S22C2 rw	S22C1 rw	S22C0 rw
LCD Memory 11 003Bh	S21C3 rw	S21C2 rw	S21C1 rw	S21C0 rw	S20C3 rw	S20C2 rw	S20C1 rw	S20C0 rw
LCD Memory 10 003Ah	S19C3 rw	S19C2 rw	S19C1 rw	S19C0 rw	S18C3 rw	S18C2 rw	S18C1 rw	S18C0 rw
LCD Memory 9 0039h	S17C3 rw	S17C2 rw	S17C1 rw	S17C0 rw	S16C3 rw	S16C2 rw	S16C1 rw	S16C0 rw
LCD Memory 8 0038h	S15C3 rw	S15C2 rw	S15C1 rw	S15C0 rw	S14C3 rw	S14C2 rw	S14C1 rw	S14C0 rw
LCD Memory 7 0037h	S13C3 rw	S13C2 rw	S13C1 rw	S13C0 rw	S12C3 rw	S12C2 rw	S12C1 rw	S12C0 rw
LCD Memory 6 0036h	S11C3 rw	S11C2 rw	S11C1 rw	S11C0 rw	S10C3 rw	S10C2 rw	S10C1 rw	S10C0 rw
LCD Memory 5 0035h	S9C3 rw	S9C2 rw	S9C1 rw	S9C0 rw	S8C3 rw	S8C2 rw	S8C1 rw	S8C0 rw
LCD Memory 4 0034h	S7C3 rw	S7C2 rw	S7C1 rw	S7C0 rw	S6C3 rw	S6C2 rw	S6C1 rw	S6C0 rw
LCD Memory 3 0033h	S5C3 rw	S5C2 rw	S5C1 rw	S5C0 rw	S4C3 rw	S4C2 rw	S4C1 rw	S4C0 rw
LCD Memory 2 0032h	S3C3 rw	S3C2 rw	S3C1 rw	S3C0 rw	S2C3 rw	S2C2 rw	S2C1 rw	S2C0 rw
LCD Memory 1 0031h	S1C3 rw	S1C2 rw	S1C1 rw	S1C0 rw	S0C3 rw	S0C2 rw	S0C1 rw	S0C0 rw
LCD Cntl&Mode, LCDC 0030h	LCDM7 rw-0	LCDM6 rw-0	LCDM5 rw-0	LCDM4 rw-0	LCDM3 rw-0	LCDM2 rw-0	LCDM1 rw-0	LCDM0 rw-0

Note: The LCD Memory bits are named with the MSP430 convention. The first part of the bit name indicates the corresponding segment line and the second indicates the corresponding common line.
Example for a segment using S4 and Com3: S4C3.

8bit Timer/Counter frame, Basic Timer frame, Timer/Port frame, byte access

Bit # -	7	6	5	4	3	2	1	0
Timer/Port Enable reg., TPE 04Fh	TPSSEL3 rw-0	TPSSEL2 rw-0	TPE.5 rw-0	TPE.4 rw-0	TPE.3 rw-0	TPE.2 rw-0	TPE.1 rw-0	TPE.0 rw-0
Timer/Port Data reg., TPD 04Eh	B16 rw-0	CPON rw-0	TPD.5 rw-0	TPD.4 rw-0	TPD.3 rw-0	TPD.2 rw-0	TPD.1 rw-0	TPD.0 rw-0
Timer/Port Counter1, TPCNT2 04Dh	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
Timer/Port Counter1, TPCNT1 04Ch	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
Timer/Port control reg., TPCTL 04Bh	TPSSEL1 rw-0	TPSSEL0 rw-0	ENB rw-0	ENA rw-0	EN1 r-0	RC2FG rw-0	RC1FG rw-0	EN1FG rw-0
Counter Data, 8bit Basic Timer, BTCNT2 0047h	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
Counter Data, 8bit Basic Timer, BTCNT1 0046h	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
045h								
Counter Data, 8bit Timer/Counter, TCDAT 0044h	TCDAT.7 rw	TCDAT.6 rw	TCDAT.5 rw	TCDAT.4 rw	TCDAT.3 rw	TCDAT.2 rw	TCDAT.1 rw	TCDAT.0 rw
Pre-load Register, 8bit Timer/Counter, TCPLD 0043h	TCPLD.7 rw	TCPLD.6 rw	TCPLD.5 rw	TCPLD.4 rw	TCPLD.3 rw	TCPLD.2 rw	TCPLD.1 rw	TCPLD.0 rw
Control Register, 8bit Timer/Counter, TCCTL 0042h	SSEL1 rw-0	SSEL0 rw-0	ISCTL rw-0	TXEN rw-0	ENCNT rw-0	RXACT rw-0	TXD rw-0	RXD r(-1)
0041h								
Basic Timer, BTCTL 0040h	SSEL rw	Reset * Hold ** rw	DIV rw	FRFQ1 rw	FRFQ0 rw	IP2 rw	IP1 rw	IP0 rw

PWM Timer, EPROM control register and System Clock Generator frame, byte access

Bit # -	7	6	5	4	3	2	1	0
PWM timer counter PWMCNT.2 005Fh	2^7 rw-0	2^6 rw-0	2^5 rw-0	2^4 rw-0	2^3 rw-0	2^2 rw-0	2^1 rw-0	2^0 rw-0
PWM duty register PWMDTR.2 005Eh	2^7 rw-1	2^6 rw-1	2^5 rw-1	2^4 rw-1	2^3 rw-1	2^2 rw-1	2^1 rw-1	2^0 rw-1
PWM duty buffer PWMDTB.2 005Dh	2^7 rw-0	2^6 rw-0	2^5 rw-0	2^4 rw-0	2^3 rw-0	2^2 rw-0	21 rw-0	2^0 rw-0
PWM timer control register PWMCTL.2 005Ch	---- rw-0	SSEL2 rw-0	SSEL1 rw-0	SSEL0 rw-0	CMPM r	---- rw-0	OS rw-0	OE rw-0
PWM timer counter PWMCNT.1 005Bh	2^7 rw-0	2^6 rw-0	2^5 rw-0	2^4 rw-0	2^3 rw-0	2^2 rw-0	2^1 rw-0	2^0 rw-0
PWM duty register PWMDTR.1 005Ah	2^7 rw-1	2^6 rw-1	2^5 rw-1	2^4 rw-1	2^3 rw-1	2^2 rw-1	2^1 rw-1	2^0 rw-1
PWM duty buffer PWMDTB.1 0059h	2^7 rw-0	2^6 rw-0	2^5 rw-0	2^4 rw-0	2^3 rw-0	2^2 rw-0	2^1 rw-0	2^0 rw-0
PWM timer control register PWMCTL.1 0058h	---- rw-0	SSEL2 rw-0	SSEL1 rw-0	SSEL0 rw-0	CMPM r	---- rw-0	OS rw-0	OE rw-0
EPROM control register EPCTL 0054h	r-0	r-0	r-0	r-0	r-0	r-0	VPPS rw-0	EXE rw-0
Crystal Buffer ctl. reg. *) CBCTL 053h						CBSEL1 w-(0)	CBSEL0 w-(0)	CBE w-(0)
System Clock Gen., Freq. Cntl. SCFQCTL 0052h	M rw-0	2^6 rw-0	2^5 rw-0	2^4 rw-1	2^3 rw-1	2^2 rw-1	2^1 rw-1	2^0 rw-1
System Clock Gen., Freq. Integrator SCFI1 0051h	2^9 rw-0	2^8 rw-0	2^7 rw-0	2^6 rw-0	2^5 rw-0	2^4 rw-0	2^3 rw-0	2^2 rw-0
System Clock Gen., Freq. Integrator SCFI0 0050h	0 r	0 r	0 r	FN_4 rw-0	FN_3 rw-0	FN_2 rw-0	2^1 rw-0	2^0 rw-0

*) CBSel1, CBSEL0 and CBE bit are reset with POR signal.

USART frame, UART Mode selected: SYNC bit = 0, byte access

Bit # -	7	6	5	4	3	2	1	0
07Fh								
Transmit Buffer TXBUF 077h	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
Receive Buffer RXBUF 076h	2^7 r	2^6 r	2^5 r	2^4 r	2^3 r	2^2 r	2^1 r	2^0 r
Baud Rate UBR1 075h	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^9 rw	2^8 rw
Baud Rate UBR0 074h	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
Modulation Control UMCTL 073h	m7 rw	m6 rw	m5 rw	m4 rw	m3 rw	m2 rw	m1 rw	m0 rw
Receive Control URCTL 072h	FE rw-0	PE rw-0	OE rw-0	BRK rw-0	URXEIE rw-0	URXWIE rw-0	RXWake rw-0	RXERR rw-0
Transmit Control UTCTL 071h	unused rw-0	CKPL rw-0	SSEL1 rw-0	SSEL0 rw-0	URXSE rw-0	TXWAKE rw-0	unused rw-0	TXEPT rw-1
USART Control UCTL 070h	PENA rw-0	PEV rw-0	SP rw-0	CHAR rw-0	Listen rw-0	SYNC rw-0	MM rw-0	SWRST rw-1

USART frame, SPI Mode selected: SYNC bit = 1, byte access

Bit # -	7	6	5	4	3	2	1	0
07Fh								
Transmit Buffer TXBUF 077h	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
Receive Buffer RXBUF 076h	2^7 r	2^6 r	2^5 r	2^4 r	2^3 r	2^2 r	2^1 r	2^0 r
Baud Rate UBR1 075h	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^9 rw	2^8 rw
Baud Rate UBR0 074h	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
Modulation Control UMCTL 073h	m7 rw	m6 rw	m5 rw	m4 rw	m3 rw	m2 rw	m1 rw	m0 rw
Receive Control URCTL 072h	FE rw-0	undef. rw-0	OE rw-0	undef. rw-0	unused rw-0	unused rw-0	undef. rw-0	undef. rw-0
Transmit Control UTCTL 071h	CKPH rw-0	CKPL rw-0	SSEL1 rw-0	SSEL0 rw-0	unused rw-0	unused rw-0	STC rw-0	TXEPT rw-1
USART Control UCTL 070h	unused rw-0	unused rw-0	unused rw-0	CHAR rw-0	Listen rw-0	SYNC rw-0	MM rw-0	SWRST rw-1

A/D converter register frame, word access

Bit # -	15	14	13	12	11	10	9	8
11Fh								
AD Converter, Data Register ADAT 118h	r0	r0	R1 *) r0	R0 *) r0	2^{11} r	2^{10} r	2^{9} r	2^{8} r
reserved 116h								
AD Converter, Control Register ACTL 114h	ACTL.15 r0	ACTL.14 r0	ACTL.13 r0	ACTL.12 rw-1	ACTL.11 rw-0	ACTL.10 rw-0	ACTL.9 rw-0	ACTL.8 rw-0
AD Converter, Input Enable Reg. AEN 112h	r0	r0	r0	r0	r0	r0	r0	r0
AD Converter, Input Data Reg. AIN 110h	r0	r0	r0	r0	r0	r0	r0	r0

*) The bits ADAT.12 and ADAT.13 are read as 0 when ACTL.11=0 otherwise signals R0 and R1 are read.

Bit # -	7	6	5	4	3	2	1	0
11Eh								
AD Converter, Data Register ADAT 118h	2^{7} r	2^{6} r	2^{5} r	2^{4} r	2^{3} r	2^{2} r	2^{1} r	2^{0} r
reserved 116h								
AD Converter, Control Register ACTL 114h	ACTL.7 rw-0	ACTL.6 rw-0	ACTL.5 rw-0	ACTL.4 rw-0	ACTL.3 rw-0	ACTL.2 rw-0	ACTL.1 rw-0	ACTL.0 (w)r0
AD Converter, Input Enable Reg. AEN 112h	AEN.7 rw-0	AEN.6 rw-0	AEN.5 rw-0	AEN.4 rw-0	AEN.3 rw-0	AEN.2 rw-0	AEN.1 rw-0	AEN.0 rw-0
AD Converter, Input Data Reg. AIN 110h	AIN.7 r	AIN.6 r	AIN.5 r	AIN.4 r	AIN.3 r	AIN.2 r	AIN.1 r	AIN.0 r

Watchdog/Timer register and Timer_A interrupt Vector register frame, word access

Bit # -	15	14	13	12	11	10	9	8
Timer_A Interrupt Vector TAIV 12Eh	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0
Watchdog Timer, Control Reg. WDTCTL 120h	w0 r0	w1 r1	w0 r1	w1 r0	w1 r1	w0 r0	w1 r0	w0 r1

Bit # -	7	6	5	4	3	2	1	0
Timer_A Interrupt Vector TAIV 12Eh	0 r0	0 r0	0 r0	TAIV r-(0)	0 r-(0)	0 r-(0)	0 r-(0)	0 r0
Watchdog Timer, Control Reg. WDTCTL 120h	HOLD rw-0	NMIES rw-0	NMI rw-0	TMSEL rw-0	CNTCL (w),r0	SSEL rw-0	IS1 rw-0	IS0 rw-0

Multiplier register frame, word access

Bit # -	15	14	13	12	11	10	9	8
Sum Extend, SumExt 013Eh	*) r	*) r	*) r	*) r	*) r	*) r	*) r	*) r
Result High Word ResHi 013Ch	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw
Result Low Word ResLo 013Ah	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw
Second Operand OP2 0138h	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw
0136h								
MPY+ACC MAC 0134h	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw
Multiply signed MPYS 0132h	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw
Multiply unsigned MPY 0130h	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw

Bit # -	7	6	5	4	3	2	1	0
Sum Extend, SumExt 013Eh	*) r	*) r	*) r	*) r	*) r	*) r	*) r	*) r
Result High Word ResHi 013Ch	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
Result Low Word ResLo 013Ah	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
Second Operand OP2 0138h	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
0136h								
MPY+ACC MAC 0134h	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
Multiply signed MPYS 0132h	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
Multiply unsigned MPY 0130h	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw

*) The SUM Extend register SumExt holds the sign of the result of a 16x16-bit multiplication (MPYS) or it holds the overflow of the multiply and accumulate (MAC) operation.

The SumExt register is:

- 0FFFFh when a MPYS operation ends in a negative result
- 0h when a MPYS operation ends in a negative result
- 0h when a MAC operation has no overflow
- 1h when a MAC operation has an overflow

Timer_A register frame (I), word access

Bit # -	15	14	13	12	11	10	9	8
017Eh								
017Ch								
Cap/Com register CCR4 017Ah	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Cap/Com register CCR3 0178h	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Cap/Com register CCR2 0176h	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Cap/Com register CCR1 0174h	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Cap/Com register CCR0 0172h	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Timer_A register TAR 0170h	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
016Eh								
016Ch								
Cap/Com Control CCTL4, 0164h	CM41 rw-(0)	CM40 rw-(0)	CCIS41 rw-(0)	CCIS40 rw-(0)	SCS4 rw-(0)	SCCI4 rw-(0)	unused r0	CAP4 rw-(0)
Cap/Com Control CCTL3, 0164h	CM31 rw-(0)	CM30 rw-(0)	CCIS31 rw-(0)	CCIS30 rw-(0)	SCS3 rw-(0)	SCCI3 rw-(0)	unused r0	CAP3 rw-(0)
Cap/Com Control CCTL2, 0164h	CM21 rw-(0)	CM20 rw-(0)	CCIS21 rw-(0)	CCIS20 rw-(0)	SCS2 rw-(0)	SCCI2 rw-(0)	unused r0	CAP2 rw-(0)
Cap/Com Control CCTL1, 0164h	CM11 rw-(0)	CM10 rw-(0)	CCIS11 rw-(0)	CCIS10 rw-(0)	SCS1 rw-(0)	SCCI1 rw-(0)	unused r0	CAP1 rw-(0)
Cap/Com Control CCTL0, 0162h	CM01 rw-(0)	CM00 rw-(0)	CCIS01 rw-(0)	CCIS00 rw-(0)	SCS0 rw-(0)	SCCI0 rw-(0)	unused r0	CAP0 rw-(0)
Timer_A Control TACTL 0160h	unused rw-(0)	unused rw-(0)	unused rw-(0)	unused rw-(0)	unused rw-(0)	SSEL2 rw-(0)	SSEL1 rw-(0)	SSEL0 rw-(0)

Timer_A register frame (II), word access

Bit # -	7	6	5	4	3	2	1	0
017Eh								
017Ch								
Cap/Com register CCR4 017Ah	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Cap/Com register CCR3 0178h	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Cap/Com register CCR2 0176h	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Cap/Com register CCR1 0174h	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Cap/Com register CCR0 0172h	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Timer_A register TAR 0170h	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
016Eh								
016Ch								
Cap/Com Control CCTL4, 016Ah	OutMod42 rw-(0)	OutMod41 rw-(0)	OutMod40 rw-(0)	CCIE4 rw-(0)	CCI4 r	OUT4 rw-(0)	COV4 rw-(0)	CCIFG4 rw-(0)
Cap/Com Control CCTL3, 0168h	OutMod32 rw-(0)	OutMod31 rw-(0)	OutMod30 rw-(0)	CCIE3 rw-(0)	CCI3 r	OUT3 rw-(0)	COV3 rw-(0)	CCIFG3 rw-(0)
Cap/Com Control CCTL2, 0166h	OutMod22 rw-(0)	OutMod21 rw-(0)	OutMod20 rw-(0)	CCIE2 rw-(0)	CCI2 r	OUT2 rw-(0)	COV2 rw-(0)	CCIFG2 rw-(0)
Cap/Com Control CCTL1, 0164h	OutMod12 rw-(0)	OutMod11 rw-(0)	OutMod10 rw-(0)	CCIE1 rw-(0)	CCI1 r	OUT1 rw-(0)	COV1 rw-(0)	CCIFG1 rw-(0)
Cap/Com Control CCTL0, 0162h	OutMod02 rw-(0)	OutMod01 rw-(0)	OutMod00 rw-(0)	CCIE0 rw-(0)	CCI0 r	OUT0 rw-(0)	COV0 rw-(0)	CCIFG0 rw-(0)
Timer_A Control TACTL 0160h	ID1 rw-(0)	ID0 rw-(0)	MC1 rw-(0)	MC0 rw-(0)	unused rw-(0)	CLR rw-(0)	TAIE rw-(0)	TAIFG rw-(0)

B. Instruction Set Desciption

The MSP430 Core CPU architecture evolved from the idea of using a reduced instruction set with highly transparent instruction formats. There are core instructions that are implemented into hardware, and emulated instructions that use the hardware construction and emulate instructions with high efficiency. The emulated instructions use core instructions with the additional built-in constant generators CG1 and CG2. Both the core instructions and the emulated instructions are described in this section. The mnemonics of the emulated instructions are used with the examples.

The words in program memory used by an instruction vary from 1 to 3 words, depending on the combination of addressing modes.
Each instruction uses a minimum of one word (two bytes) in the program memory. The indexed, symbolic, absolute and immediate modes need one additional word in the program memory. These four modes are available for the source operand. The indexed, symbolic and absolute mode can be used for the destination operand.
The instruction combination for source and destination consumes one to three words of code memory.

Instruction Set Overview

				Status Bits			
				V	N	Z	C
*	ADC[.W];ADC.B	dst	dst + C -> dst	*	*	*	*
	ADD[.W];ADD.B	src,dst	src + dst -> dst	*	*	*	*
	ADDC[.W];ADDC.B	src,dst	src + dst + C -> dst	*	*	*	*
	AND[.W];AND.B	src,dst	src .and. dst -> dst	0	*	*	*
	BIC[.W];BIC.B	src,dst	.not.src .and. dst -> dst	-	-	-	-
	BIS[.W];BIS.B	src,dst	src .or. dst -> dst	-	-	-	-
	BIT[.W];BIT.B	src,dst	src .and. dst	0	*	*	*
*	BR	dst	Branch to	-	-	-	-
	CALL	dst	PC+2 -> stack, dst -> PC	-	-	-	-
*	CLR[.W];CLR.B	dst	Clear destination	-	-	-	-
*	CLRC	Clear carry bit		-	-	-	0
*	CLRN	Clear negative bit		-	0	-	-
*	CLRZ	Clear zero bit		-	-	0	-
	CMP[.W];CMP.B	src,dst	dst - src	*	*	*	*
*	DADC[.W];DADC.B	dst	dst + C -> dst (decimal)	*	*	*	*
	DADD[.W];DADD.B	src,dst	src + dst + C -> dst (decimal)	*	*	*	*
*	DEC[.W];DEC.B	dst	dst - 1 -> dst	*	*	*	*
*	DECD[.W];DECD.B	dst	dst - 2 -> dst	*	*	*	*
*	DINT	Disable interrupt		-	-	-	-
*	EINT	Enable interrupt		-	-	-	-
*	INC[.W];INC.B	dst	Increment destination, dst +1 -> dst	*	*	*	*
*	INCD[.W];INCD.B	dst	Double-Increment destination, dst+2->dst	*	*	*	*
*	INV[.W];INV.B	dst	Invert destination	*	*	*	*
	JC/JHS	Label	Jump to Label if Carry-bit is set	-	-	-	-
	JEQ/JZ	Label	Jump to Label if Zero-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	-	-	-	-
	JN	Label	Jump to Label if Negative-bit is set	-	-	-	-
	JNC/JLO	Label	Jump to Label if Carry-bit is reset	-	-	-	-
	JNE/JNZ	Label	Jump to Label if Zero-bit is reset	-	-	-	-

Note: Marked instructions are emulated instructions

All marked instructions (*) are emulated instructions. The emulated instructions use core instructions combined with the architecture and implementation of the CPU, for higher code efficiency and faster execution.

				Status Bits			
				V	N	Z	C
	MOV[.W];MOV.B	src,dst	src -> dst	-	-	-	-
*	NOP		No operation	-	-	-	-
*	POP[.W];POP.B	dst	Item from stack, SP+2 → SP	-	-	-	-
	PUSH[.W];PUSH.B	src	SP - 2 → SP, src → @SP	-	-	-	-
	RETI		Return from interrupt TOS → SR, SP + 2 → SP TOS → PC, SP + 2 → SZP	*	*	*	*
*	RET		Return from subroutine TOS → PC, SP + 2 → SP	-	-	-	-
*	RLA[.W];RLA.B	dst	Rotate left arithmetically	*	*	*	*
*	RLC[.W];RLC.B	dst	Rotate left through carry	*	*	*	*
	RRA[.W];RRA.B	dst	MSB → MSB → ....LSB → C	0	*	*	*
	RRC[.W];RRC.B	dst	C → MSB → ....LSB → C	*	*	*	*
*	SBC[.W];SBC.B	dst	Subtract carry from destination	*	*	*	*
*	SETC		Set carry bit	-	-	-	1
*	SETN		Set negative bit	-	1	-	-
*	SETZ		Set zero bit	-	-	1	-
	SUB[.W];SUB.B	src,dst	dst + .not.src + 1 → dst	*	*	*	*
	SUBC[.W];SUBC.B	src,dst	dst + .not.src + C → dst	*	*	*	*
	SWPB	dst	swap bytes	-	-	-	-
	SXT	dst	Bit7 → Bit8 .... Bit15	0	*	*	*
*	TST[.W];TST.B	dst	Test destination	0	*	*	1
	XOR[.W];XOR.B	src,dst	src .xor. dst → dst	*	*	*	*

Note: Marked instructions

All marked instructions (*) are emulated instructions. The emulated instructions use core instructions combined with the architecture and implementation of the CPU, for higher code efficiency and faster execution.

Instruction Formats

Double operand instructions (core instructions)

The instruction format using double operands consists of four main fields, in total a 16bit code:

● operational code field, 4bit	[OP-Code]
● source field, 6bit	[source register + As]
● byte operation identifier, 1bit	[BW]
● destination field, 5bit	[dest. register + Ad]

The source field is composed of two addressing bits and the 4bit register number (0....15); the destination field is composed of one addressing bit and the 4bit register number (0....15). The byte identifier B/W indicates whether the instruction is executed as a byte (B/W=1) or as a word instruction (B/W=0)

15 12	11 8	7	6	5 4	3 0
OP - Code	source register	Ad	B/W	As	dest. register
operational code field					

				Status Bits			
				V	N	Z	C
ADD[.W];	ADD.B	src,dst	src + dst -> dst	*	*	*	*
ADDC[.W];	ADDC.B	src,dst	src + dst + C -> dst	*	*	*	*
AND[.W];	AND.B	src,dst	src .and. dst -> dst	0	*	*	*
BIC[.W];	BIC.B	src,dst	.not.src .and. dst -> dst	-	-	-	-
BIS[.W];	BIS.B	src,dst	src .or. dst -> dst	-	-	-	-
BIT[.W];	BIT.B	src,dst	src .and. dst	0	*	*	*
CMP[.W];	CMP.B	src,dst	dst - src	*	*	*	*
DADD[.W];	DADD.B	src,dst	src + dst + C -> dst (dec)	*	*	*	*
MOV[.W];	MOV.B	src,dst	src -> dst	-	-	-	-
SUB[.W];	SUB.B	src,dst	dst + .not.src + 1 -> dst	*	*	*	*
SUBC[.W];	SUBC.B	src,dst	dst + .not.src + C -> dst	*	*	*	*
XOR[.W];	XOR.B	src,dst	src .xor. dst -> dst	*	*	*	*

Note: Operations using Status Register SR for destination

All operations using Status Register SR for destination overwrite the contents of SR with the result of that operation: the status bits are not affected as described in that operation.

Example: ADD #3,SR ; Operation: (SR) + 3 --> SR

Single operand instructions (core instructions)

The instruction format using a single operand consists of two main fields, in total 16bit:

- operational code field, 9bit with 4MSB equal '1h'
- byte operation identifier, 1bit [BW]
- destination field, 6bit [destination register + Ad]

The destination field is composed of two addressing bits and the 4bit register number (0....15). The bit position of the destination field is located in the same position as the two operand instructions. The byte identifier B/W indicates whether the instruction is executed as a byte (B/W=1) or as a word instruction (B/W=0)

15			12	11	10	9		7	6	5 4	3 0
0	0	0	1	X	X	X	X	X	B/W	Ad	destination register
operational code field									destination field		

				Status Bits			
				V	N	Z	C
RRA[.W];	RRA.B	dst	MSB → MSB → ...LSB → C	0	*	*	*
RRC[.W];	RRC.B	dst	C → MSB → ....LSB → C	*	*	*	*
PUSH[.W];	PUSH.B	dst	SP - 2 → SP, src → @SP	-	-	-	-
SWPB		dst	swap bytes	-	-	-	-
CALL		dst	PC+2 → @SP, dst → PC	-	-	-	-
RETI			TOS → SR, SP + 2 → SP TOS → PC, SP + 2 → SP	*	*	*	*
SXT		dst	Bit7 -> Bit8 .... Bit15	0	*	*	*

B

Conditional and unconditional Jumps (core instructions)

The instruction format for (un-)conditional jumps consists of two main fields, in total 16bit :

- operational code (OP-Code) field, 6bit
- jump offset field, 10bit

The operational code field is composed of OP-Code (3bits), and 3 bits according to the following conditions.

15		13	12		10	9									0
0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X
OP-Code			Jump-on .Code			Sign	Offset								
operational code field						Jump offset field									

The conditional jumps allow jumps to addresses in the range -511 to +512 words relative to the current address. The assembler computes the signed offsets and inserts them into the opcode.

JC/JHS	Label	Jump to Label if Carry-bit is set
JEQ/JZ	Label	Jump to Label if Zero-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
JN	Label	Jump to Label if Negative-bit is set
JNC/JLO	Label	Jump to Label if Carry-bit is reset
JNE/JNZ	Label	Jump to Label if Zero-bit is reset

Note: Conditional and unconditional Jumps

The conditional and unconditional Jumps do not effect the status bits.

A Jump which has been taken alters the PC with the offset:

PCnew=PCold + 2 + 2*offset.

A Jump which has not been taken continues the program with the ascending instruction.

Emulation of instructions without ROM penalty

The following instructions can be emulated with the reduced instruction set, without additional ROM words. The assembler accepts the mnemonic of the emulated instruction, and inserts the opcode of the suitable core instruction.

Note: Emulation of the following instructions

The emulation of the following instructions is possible using the contents of R2 and R3:
The register R2(CG1) contains the immediate values 2 and 4; the register R3(CG2) contains -1 or 0FFFFh, 0, +1 and +2 depending on the addressing bits As. The assembler sets the addressing bits according to the immediate value used.

Short form of emulated instructions

Mnemonic		Description	Statusbits				Emulation	
			V	N	Z	C		
Arithmetical instructions								
ADC[.W]	dst	Add carry to destination	*	*	*	*	ADDC	#0,dst
ADC.B	dst	Add carry to destination	*	*	*	*	ADDC.B	#0,dst
DADC[.W]	dst	Add carry decimal to destination	*	*	*	*	DADD	#0,dst
DADC.B	dst	Add carry decimal to destination	*	*	*	*	DADD.B	#0,dst
DEC[.W]	dst	Decrement destination	*	*	*	*	SUB	#1,dst
DEC.B	dst	Decrement destination	*	*	*	*	SUB.B	#1,dst
DECD[.W]	dst	Double-Decrement destination	*	*	*	*	SUB	#2,dst
DECD.B	dst	Double-Decrement destination	*	*	*	*	SUB.B	#2,dst
INC[.W]	dst	Increment destination	*	*	*	*	ADD	#1,dst
INC.B	dst	Increment destination	*	*	*	*	ADD.B	#1,dst
INCD[.W]	dst	Increment destination	*	*	*	*	ADD	#2,dst
INCD.B	dst	Increment destination	*	*	*	*	ADD.B	#2,dst
SBC[.W]	dst	Subtract carry from destination	*	*	*	*	SUBC	#0,dst
SBC.B	dst	Subtract carry from destination	*	*	*	*	SUBC.B	#0,dst
Logical instructions								
INV[.W]	dst	Invert destination	*	*	*	*	XOR	#0FFFFh,dst
INV.B	dst	Invert destination	*	*	*	*	XOR.B	#0FFFFh,dst
RLA[.W]	dst	Rotate left arithmetically	*	*	*	*	ADD	dst,dst
RLA.B	dst	Rotate left arithmetically	*	*	*	*	ADD.B	dst,dst
RLC[.W]	dst	Rotate left through carry	*	*	*	*	ADDC	dst,dst
RLC.B	dst	Rotate left through carry	*	*	*	*	ADDC.B	dst,dst
Data instructions (common use)								
CLR[.W]		Clear destination	-	-	-	-	MOV	#0,dst
CLR.B		Clear destination	-	-	-	-	MOV.B	#0,dst
CLRC		Clear carry bit	-	-	-	0	BIC	#1,SR
CLRN		Clear negative bit	-	0	-	-	BIC	#4,SR
CLRZ		Clear zero bit	-	-	0	-	BIC	#2,SR
POP	dst	Item from stack	-	-	-	-	MOV	@SP+,dst
SETC		Set carry bit	-	-	-	1	BIS	#1,SR
SETN		Set negative bit	-	1	-	-	BIS	#4,SR
SETZ		Set zero bit	-	-	1	-	BIS	#2,SR
TST[.W]	dst	Test destination	0	*	*	1	CMP	#0,dst
TST.B	dst	Test destination	0	*	*	1	CMP.B	#0,dst
Program flow instructions								
BR	dst	Branch to	-	-	-	-	MOV	dst,PC
DINT		Disable interrupt	-	-	-	-	BIC	#8,SR
EINT		Enable interrupt	-	-	-	-	BIS	#8,SR
NOP		No operation	-	-	-	-	MOV	#0h,#0h
RET		Return from subroutine	-	-	-	-	MOV	@SP+,PC

Instruction set description - alphabetical order

This section catalogues and describes all core and emulated instructions. Some examples are given for explanation and as application hints.
The suffix .W or no suffix in the instruction mnemonic will result in a word operation.
The suffix .B at the instruction mnemonic will result in a byte operation.

* ADC[.W] Add carry to destination
* ADC.B Add carry to destination

Syntax ADC dst or ADC.W dst
ADC.B dst

Operation dst + C -> dst

Emulation ADDC #0,dst
ADDC.B #0,dst

Description The carry C is added to the destination operand. The previous contents of the destination are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if dst was incremented from 0FFFFh to 0000, reset otherwise
Set if dst was incremented from 0FFh to 00, reset otherwise
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The 16-bit counter pointed to by R13 is added to a 32-bit counter pointed to by R12.

```
ADD     @R13,0(R12)     ; Add LSDs
ADC     2(R12)                  ; Add carry to MSD
```

Example The 8-bit counter pointed to by R13 is added to a 16-bit counter pointed to by R12.

```
ADD.B     @R13,0(R12)     ; Add LSDs
ADC.B     1(R12)          ; Add carry to MSD
```

ADD[.W] Add source to destination
ADD.B Add source to destination

Syntax

```
ADD      src,dst    or    ADD.W     src,dst
ADD.B    src,dst
```

Operation src + dst -> dst

Description The source operand is added to the destination operand. The source operand is not affected, the previous contents of the destination are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if there is a carry from the result, cleared if not.
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example R5 is increased by 10. The 'Jump' to TONI is performed on a carry

```
ADD      #10,R5
JC       TONI          ; Carry occurred
......                 ; No carry
```

Example R5 is increased by 10. The 'Jump' to TONI is performed on a carry

```
ADD.B    #10,R5        ; Add 10 to Lowbyte of R5
JC       TONI          ; Carry occurred, if (R5) ≥ 246 [0Ah+0F6h]
......                 ; No carry
```

ADDC[.W] Add source and carry to destination.
ADDC.B Add source and carry to destination.

Syntax

```
ADDC    src,dst   or   ADDC.W   src,dst
ADDC.B  src,dst
```

Operation src + dst + C -> dst

Description The source operand and the carry C are added to the destination operand. The source operand is not affected, the previous contents of the destination are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if there is a carry from the MSB of the result, reset if not
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The 32-bit counter pointed to by R13 is added to a 32-bit counter eleven **words** (20/2 + 2/2) above pointer in R13.

```
ADD     @R13+,20(R13)      ; ADD LSDs with no carryin
ADDC    @R13+,20(R13)      ; ADD MSDs with carry
...                        ; resulting from the LSDs
```

Example The 24-bit counter pointed to by R13 is added to a 24-bit counter eleven words above pointer in R13.

```
ADD.B     @R13+,10(R13)    ; ADD LSDs with no carryin
ADDC.B    @R13+,10(R13)    ; ADD medium Bits with carry
ADDC.B    @R13+,10(R13)    ; ADD MSDs with carry
...                        ; resulting from the LSDs
```

AND[.W] source AND destination
AND.B source AND destination

Syntax

```
AND     src,dst    or      AND.W   src,dst
AND.B   src,dst
```

Operation src .AND. dst -> dst

Description The source operand and the destination operand are logically AND'ed. The result is placed into the destination.

Status Bits

N: Set if MSB of result is set, reset if not set
Z: Set if result is zero, reset otherwise
C: Set if result is not zero, reset otherwise (= .NOT. Zero)
V: Reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The bits set in R5 are used as a mask (#0AA55h) for the word addressed by TOM. If the result is zero, a branch is taken to label TONI

```
MOV     #0AA55h,R5      ; Load mask into register R5
AND     R5,TOM          ; mask word addressed by TOM with R5
JZ      TONI            ;
......                  ; Result is not zero
;
;
;       or
;
;
AND     #0AA55h,TOM
JZ      TONI
```

Example The bits of mask #0A5h are logically AND'ed with the Lowbyte TOM. If the result is zero, a branch is taken to label TONI

```
AND.B   #0A5h,TOM       ; mask Lowbyte TOM with R5
JZ      TONI            ;
......                  ; Result is not zero
```

BIC[.W] Clear bits in destination
BIC.B Clear bits in destination

Syntax

```
BIC      src,dst    or      BIC.W    src,dst
BIC.B    src,dst
```

Operation .NOT.src .AND. dst -> dst

Description The inverted source operand and the destination operand are logically AND'ed. The result is placed into the destination. The source operand is not affected.

Status Bits

N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The 6 MSBs of the RAM word LEO are cleared.

```
BIC#0FC00h,LEO                ; Clear 6 MSBs in MEM(LEO)
```

Example The 5 MSBs of the RAM byte LEO are cleared.

```
BIC.B    #0F8h,LEO            ; Clear 5 MSBs in Ram location LEO
```

Example The Portpins P0 and P1 are cleared.

```
P0OUT    .equu    011h        ;Definition of the Portaddress
P0_0     .equ     01h
P0_1     .equ     02h

BIC.B    #P0_0+P0_1,&P0OUT    ;Set P0.0 and P0.1 to low
```

BIS[.W] Set bits in destination
BIS.B Set bits in destination

Syntax

```
BIS       src,dst   or      BIS.W            src,dst
BIS.B     src,dst
```

Operation src .OR. dst -> dst

Description The source operand and the destination operand are logically OR'ed. The result is placed into the destination. The source operand is not affected.

Status Bits
N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The 6 LSB's of the RAM word TOM are set.

```
BIS       #003Fh,TOM  ;  set the 6 LSB's in RAM location TOM
```

Example Start an A/D-conversion

```
ASOC      .equ      1         ; Start of Conversion bit
ACTL      .equ      114h      ; ADC-Control Register

BIS       #ASOC,&ACTL         ; Start A/D-conversion
```

Example The 3 MSBs of the RAM byte TOM are set.

```
BIS.B     #0E0h,TOM           ; set the 3 MSBs in RAM location TOM
```

Example The Portpins P0 and P1 are set to high

```
P0OUT     .equ        011h
P0        .equ        01h
P1        .equ        02h

BIS.B     #P0+P1,&P0OUT
```

BIT[.W] Test bits in destination
BIT.B Test bits in destination

Syntax BIT src,dst or BIT.W src,dst

Operation src .AND. dst

Description The source operand and the destination operand are logically AND'ed. The result affects only the Status Bits. The source and destination operands are not affected.

Status Bits
N: Set if MSB of result is set, reset if not set
Z: Set if result is zero, reset otherwise
C: Set if result is not zero, reset otherwise (.NOT. Zero)
V: Reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example If bit 9 of R8 is set, a branch is taken to label TOM.

```
BIT      #0200h,R8          ; bit 9 of R8 set ?
JNZ      TOM                ; Yes, branch to TOM
...                         ; No, proceed
```

Example Determine which A/D-Channel is configured by the MUX

```
ACTL     .equu    114h      ; ADC Control Register

BIT      #4,&ACTL           ; Is Channel 0 selected ?
jnz      END                ; Yes, branch to END
```

Example If bit 3 of R8 is set, a branch is taken to label TOM.

```
BIT.B      #8,R8
JC         TOM
```

B

BIT (continued)

Example The receive bit RCV of a serial communication is tested. Since while using the BIT instruction to test a single bit the carry is equal to the state of the tested bit, the carry is ; used by the subsequent instruction: the read info is shifted into the register RECBUF.

```
;
; Serial communication with LSB is shifted first:
                                          ; xxxx      xxxx     xxxx     xxxx
          BIT.B      #RCV,RCCTL           ; Bit info into carry
          RRC        RECBUF               ; Carry -> MSB of RECBUF
                                          ; cxxx  xxxx
          ......                          ; repeat previous two instructions
          ......                          ; 8 times
                                          ; cccc  cccc
                                          ; ^          ^
                                          ; MSB       LSB

; Serial communication with MSB is shifted first:
          BIT.B      #RCV,RCCTL           ; Bit info into carry
          RLC.B      RECBUF               ; Carry -> LSB of RECBUF
                                          ; xxxx      xxxc
          ......                          ; repeat previous two instructions
          ......                          ; 8 times
                                          ; cccc      cccc
                                          ; |          LSB
                                          ; MSB
```

* BR, BRANCH Branch to .... destination

Syntax BR dst

Operation dst -> PC

Emulation MOV dst,PC

Description An unconditional branch is taken to an address anywhere in the 64 K address space. All source addressing modes may be used. The branch instruction is a word instruction.

Status Bits Status bits are not affected

Examples Examples for all addressing modes are given

```
BR      #EXEC     ;Branch to label EXEC or direct branch (e.g. #0A4h)
                  ; Core instruction  MOV  @PC+,PC

BR      EXEC      ; Branch to the address contained in EXEC
                  ; Core instruction  MOV  X(PC),PC
                  ; Indirect address

BR      &EXEC     ; Branch to the address contained in absolute
                  ; address EXEC
                  ; Core instruction  MOV  X(0),PC
                  ; Indirect address

BR      R5        ; Branch to the address contained in R5
                  ; Core instruction  MOV  R5,PC
                  ; Indirect R5

BR      @R5       ; Branch to the address contained in the word R5
                  ; points to.
                  ; Core instruction  MOV  @R5,PC
                  ; Indirect, indirect R5

BR      @R5+      ; Branch to the address contained in the word R5
                  ; points to and increments pointer in R5 afterwards.
                  ; The next time - S/W flow uses R5 pointer - it can
                  ; alter the program execution due to access to
                  ; next address in a table, pointed by R5
                  ; Core instruction  MOV  @R5,PC
                  ; Indirect, indirect R5 with autoincrement

BR      X(R5)     ; Branch to the address contained in the address
                  ; pointed to  by R5 + X  (e.g. table with address
                  ; starting at X). X can be an address or a label
                  ; Core instruction MOV  X(R5),PC
                  ; Indirect indirect R5 + X
```

CALL Subroutine

Syntax CALL dst

Operation

```
dst     -> tmp     dst is evaluated and stored
SP - 2  -> SP
PC      -> @SP     updated PC to TOS
tmp     -> PC      saved dst to PC
```

Description A subroutine call is made to an address anywhere in the 64-K-address space. All addressing modes may be used. The return address (the address of the following instruction) is stored on the stack. The call instruction is a word instruction.

Status Bits Status bits are not affected

Example Examples for all addressing modes are given

```
CALL   #EXEC    ; Call on label EXEC or immediate address (e.g.
                ; #0A4h)
                ; SP-2 → SP,  PC+2 → @SP,  @PC+ → PC
CALL   EXEC     ; Call on the address contained in EXEC
                ; SP-2 → SP,  PC+2 → @SP,  X(PC) → PC
                ; Indirect address
CALL   &EXEC    ; Call on the address contained in absolute address
                ; EXEC
                ; SP-2 → SP,  PC+2 → @SP,  X(PC) → PC
                ; Indirect address
CALL   R5       ; Call on the address contained in R5
                ; SP-2 → SP,  PC+2 → @SP,  R5 → PC
                ; Indirect R5
CALL   @R5      ; Call on the address contained in the word R5
                ; points
                ; to
                ; SP-2 → SP,  PC+2 → @SP,  @R5 → PC
                ; Indirect, indirect R5
CALL   @R5+     ; Call on the address contained in the word R5 points
                ; to and increments pointer in R5. The next time -
                ; S/W flow uses R5 pointer - it can alter the
                ; program execution due to access to next address
                ; in a table, pointed ; to by R5
                ; SP-2 → SP,  PC+2 → @SP,  @R5 → PC
                ; Indirect, indirect R5 with autoincrement
CALL   X(R5)    ; Call on the address contained in the address pointed
                ; to by R5 + X  (e.g. table with address starting at X)
                ; X can be an address or a label
                ; SP-2 → SP,  PC+2 → @SP,  X(R5) → PC
                ; Indirect indirect R5 + X
```

* CLR[.W] Clear destination
* CLR.B Clear destination

Syntax

```
CLR      dst     or    CLR.W   dst
CLR.B    dst
```

Operation 0 -> dst

Emulation

```
MOV      #0,dst
MOV.B    #0,dst
```

Description The destination operand is cleared.

Status Bits Status bits are not affected

Example RAM word TONI is cleared

```
CLR      TONI       ; 0 -> TONI
```

Example Register R5 is cleared

```
CLR      R5
```

Example RAM byte TONI is cleared

```
CLR.B      TONI     ; 0 -> TONI
```

* CLRC Clear carry bit

Syntax CLRC

Operation 0 -> C

Emulation BIC #1,SR

Description The Carry Bit C is cleared. The clear carry instruction is a word instruction.

Status Bits
N: Not affected
Z: Not affected
C: Cleared
V: Not affected

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The 16bit decimal counter pointed to by R13 is added to a 32bit counter pointed to by R12.

```
CLRC                          ; C=0: Defines start
DADD    @R13,0(R12)           ; add 16bit counter to Lowword of 32bit
                              ; counter
DADC    2(R12)                ; add carry to Highword of 32bit counter
```

* CLRN Clear Negative bit

Syntax CLRN

Operation 0 → N
or
(.NOT.src .AND. dst -> dst)

Emulation BIC #4,SR

Description The constant 04h is inverted (0FFFBh) and the destination operand are logically AND'ed. The result is placed into the destination. The clear negative bit instruction is a word instruction.

Status Bits
N: Reset to 0
Z: Not affected
C: Not affected
V: Not affected

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The Negative bit in the status register is cleared. This avoids the special treatment of the called subroutine with negative numbers.

```
            CLRN
            CALL     SUBR
            ......
            ......
SUBR        JN       SUBRET        ; If input is negative: do nothing and return
            ......
            ......
            ......
SUBRET      RET
```

* CLRZ Clear Zero bit

Syntax CLRZ

Operation 0 → Z
or
(.NOT.src .AND. dst -> dst)

Emulation BIC #2,SR

Description The constant 02h is inverted (0FFFDh) and the destination operand are logically AND'ed. The result is placed into the destination. The clear zero bit instruction is a word instruction.

Status Bits
N: Not affected
Z: Reset to 0
C: Not affected
V: Not affected

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The Zero bit in the status register is cleared.

```
CLRZ
```

CMP[.W] compare source and destination
CMP.B compare source and destination

Syntax

```
CMP    src,dst    or    CMP.W   src,dst
CMP.B  src,dst
```

Operation

dst + .NOT.src + 1
or
(dst - src)

Description The source operand is subtracted from the destination operand. This is made by adding of the 1's complement of the source operand plus 1. The two operands are not affected and, the result is not stored; only the status bits are affected.

Status Bits
N: Set if result is negative, reset if positive (src >= dst)
Z: Set if result is zero, reset otherwise (src = dst)
C: Set if there is a carry from the MSB of the result, reset if not
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example R5 and R6 are compared. If they are equal, the program continues at the label EQUAL

```
CMP    R5,R6       ; R5 = R6 ?
JEQ    EQUAL       ; YES, JUMP
```

Example Two RAM blocks are compared. If they not equal, the program branches to the label ERROR

```
        MOV    #NUM,R5               ;number of words to be compared
L$1     CMP    &BLOCK1,&BLOCK2       ;Are Words equal ?
        JNZ    ERROR                 ;No, branch to ERROR
        DEC    R5                    ;Are all words compared?
        JNZ    L$1                   ;No, another compare
```

Example The RAM bytes addressed by EDE and TONI are compared. If they are equal, the program continues at the label EQUAL

```
        CMP.B  EDE,TONI              ; MEM(EDE) = MEM(TONI) ?
        JEQ    EQUAL                 ; YES, JUMP
```

CMP.B (continued)

Example Check two Keys, which are connected to the Portpin P0 and P1. If key1 is pressed, the program branches to the label MENU1; if key2 is pressed, the program branches to MENU2.

```
P0IN    .EQU    010h
KEY1    .EQU    01h
KEY2    .EQU    02h

        CMP.B   #KEY1,&P0IN
        JEQ     MENU1
        CMP.B   #KEY2,&P0IN
        JEQ     MENU2
```

* DADC[.W] Add carry decimally
* DADC.B Add carry decimally

Syntax

```
DADC     dst   o   DADC.W   src,dst
DADC.B   dst
```

Operation dst + C -> dst (decimally)

Emulation

```
DADD     #0,dst
DADD.B   #0,dst
```

Description The Carry Bit C is added decimally to the destination

Status Bits

N: Set if MSB is 1
Z: Set if dst is 0, reset otherwise
C: Set if destination increments from 9999 to 0000, reset otherwise
Set if destination increments from 99 to 00, reset otherwise
V: Undefined

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The 4-digit decimal number contained in R5 is added to an 8-digit decimal number pointed to by R8

```
CLRC                        ; Reset carry
                            ; next instruction's start condition is defined
DADD   R5,0(R8)             ; Add LSDs + C
DADC   2(R8)                ; Add carry to MSD
```

Example The 2-digit decimal number contained in R5 is added to an 4-digit decimal number pointed to by R8

```
CLRC                        ; Reset carry
                            ; next instruction's start condition is
                            ; defined
DADD.B     R5,0(R8)         ; Add LSDs + C
DADC       1(R8)            ; Add carry to MSDs
```

DADD[.W] source and carry added decimally to destination
DADD.B source and carry added decimally to destination

Syntax

```
DADD    src,dst        or      DADD.W src,dst
DADD.B  src,dst
```

Operation src + dst + C -> dst (decimally)

Description The source operand and the destination operand are treated as four binary coded decimals (BCD) with positive signs. The source operand and the carry C are added decimally to the destination operand. The source operand is not affected, the previous contents of the destination are lost. The result is not defined for non-BCD numbers.

Status Bits
N: Set if the MSB is 1, reset otherwise
Z: Set if result is zero, reset otherwise
C: Set if the result is greater than 9999.
Set if the result is greater than 99.
V: Undefined

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The 8-digit-BCD-number contained in R5 and R6 is added decimally to a 8-digit-BCD-number contained in R3 and R4 (R6 and R4 contain the MSDs).

```
CLRC                      ; CLEAR CARRY
DADD      R5,R3           ; add LSDs
DADD      R6,R4           ; add MSDs with carry
JC        OVERFLOW        ; If carry occurs go to error handling routine
```

Example The 2-digit decimal counter in RAMbyte CNT is incremented by one.

```
CLRC                      ; clear Carry
DADD.B      #1,CNT        ; increment decimal counter
```

or

```
SETC
DADD.B      #0,CNT        ; ≡ DADC.B      CNT
```

* DEC[.W] Decrement destination
* DEC.B Decrement destination

Syntax DEC dst or DEC.W dst
DEC.B dst

Operation dst - 1 -> dst

Emulation SUB #1,dst
Emulation SUB.B #1,dst

Description The destination operand is decremented by one. The original contents are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if dst contained 1, reset otherwise
C: Reset if dst contained 0, set otherwise
V: Set if an arithmetic overflow occurs, otherwise reset.
Set if initial value of destination was 08000h, otherwise reset.
Set if initial value of destination was 080h, otherwise reset.

Mode Bits **OscOff, CPUOff** and **GIE** are not affected

* DEC (continued)

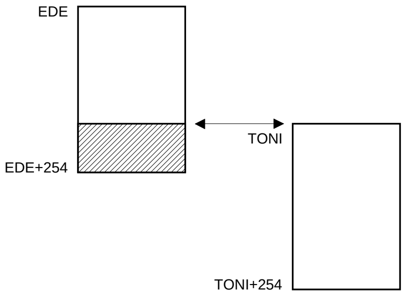
Example R10 is decremented by 1

```
        DEC     R10             ; Decrement R10
```

; Move a block of 255 bytes from memory location starting with EDE to memory location
; starting with TONI
; Tables should not overlap: start of destination address TONI must not be within the range ; EDE to EDE+0FEh
;

```
            MOV         #EDE,R6
            MOV         #255,R10
L$1         MOV.B       @R6+,TONI-EDE-1(R6)
            DEC         R10
            JNZ         L$1
```

; Do not transfer tables with the routine above with this overlap:



Example Memory byte at address LEO is decremented by 1

```
            DEC.B       LEO     ; Decrement MEM(LEO)
```

; Move a block of 255 bytes from memory location starting with EDE to memory location
; starting with TONI
; Tables should not overlap: start of destination address TONI must not be within the
; range EDE to EDE+0FEh
;

```
            MOV         #EDE,R6
            MOV.B       #255,LEO
L$1         MOV.B       @R6+,TONI-EDE-1(R6)
            DEC.B       LEO
            JNZ         L$1
```

* DECD[.W] Double-Decrement destination
* DECD.B Double-Decrement destination

Syntax DECD dst or DECD.W dst
DECD.B dst

Operation dst - 2 -> dst

Emulation SUB #2,dst
Emulation SUB.B #2,dst

Description The destination operand is decremented by two. The original contents are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if dst contained 2, reset otherwise
C: Reset if dst contained 0 or 1, set otherwise
V: Set if an arithmetic overflow occurs, otherwise reset.
Set if initial value of destination was 08001 or 08000h, otherwise reset.
Set if initial value of destination was 081 or 080h, otherwise reset.

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example R10 is decremented by 2

```
          DECD    R10             ; Decrement R10 by two
```

```
; Move a block of 255 words from memory location starting with EDE to memory location
; starting with TONI
; Tables should not overlap: start of destination address TONI must not be within the
; range EDE to EDE+0FEh
;
          MOV     #EDE,R6
          MOV     #510,R10
    L$1   MOV     @R6+,TONI-EDE-2(R6)
          DECD    R10
          JNZ     L$1
```

Example Memory at location LEO is decremented by 2

```
          DECD.B  LEO             ; Decrement MEM(LEO)
```

Decrement status byte STATUS by 2

```
          DECD.B  STATUS
```

* DINT Disable (general) interrupts

Syntax DINT

Operation 0 → GIE
or
(0FFF7h .AND. SR → SR / .NOT.src .AND. dst -> dst)

Emulation BIC #8,SR

Description All interrupts are disabled.
The constant #08h is inverted and logically AND'ed with the status register SR. The result is placed into the SR.

Status Bits
N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits **GIE** is reset.
OscOff and **CPUOff** are not affected

Example The general interrupt enable bit GIE in the status register is cleared to allow a non disrupted move of a 32bit counter. This ensures that the counter is not modified during the move by any interrupt.

```
DINT                    ; All interrupt events using the GIE bit are

                        ; disabled
MOV   COUNTHI,R5        ; Copy counter
MOV   COUNTLO,R6
EINT                    ; All interrupt events using the GIE bit are
                        ; enabled
```

> **Note: Disable Interrupt**
>
> The instruction following the disable interrupt instruction DINT is executed when the interrupt request becomes active during execution of DINT. If any code sequence needs to be protected from being interrupted, the DINT instruction should be executed at least one instruction before this sequence.

B

* EINT Enable (general) interrupts

Syntax EINT

Operation 1 → GIE
or
(0008h .OR. SR -> SR / .NOT.src .OR. dst -> dst)

Emulation BIS #8,SR

Description All interrupts are enabled.
The constant #08h and the status register SR are logically OR'ed. The result is placed into the SR.

Status Bits **N:** Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits **GIE** is set.
OscOff and **CPUOff** are not affected

Example The general interrupt enable bit GIE in the status register is set.

```
; Interrupt routine of port P0.2 to P0.7
; The interrupt level is the lowest in the system
; P0IN is the address of the register where all port bits are read. P0IFG is the address of
; the register where all interrupt events are latched.
;
            PUSH.B    &P0IN
            BIC.B     @SP,&P0IFG ; Reset only accepted flags
            EINT                ; Preset port 0 interrupt flags stored on stack
                                ; other interrupts are allowed
            BIT       #Mask,@SP
            JEQ       MaskOK    ; Flags are present identically to mask: Jump
            ......
MaskOK      BIC       #Mask,@SP
            ......
            INCD      SP        ; Housekeeping: Inverse to PUSH instruction
                                ; at the start of interrupt subroutine. Corrects
                                ; the  stack pointer.
            RETI
```

Note: Enable Interrupt

The instruction following the enable interrupt instruction EINT is executed anyway, even if an interrupt service request is pending.

* INC[.W] Increment destination
* INC.B Increment destination

Syntax

```
INC     dst   or  INC.W    dst
INC.B   dst
```

Operation dst + 1 -> dst

Emulation

```
ADD     #1,dst
```

Description The destination operand is incremented by one. The original contents are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if dst contained 0FFFFh, reset otherwise
Set if dst contained 0FFh, reset otherwise
C: Set if dst contained 0FFFFh, reset otherwise
Set if dst contained 0FFh, reset otherwise
V: Set if dst contained 07FFFh, reset otherwise
Set if dst contained 07Fh, reset otherwise

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The item on the top of a software stack (not the system stack) for byte data is removed.

```
SSP  .EQU   R4
;
     INC    SSP  ; Remove TOSS (top of SW stack) by increment
                 ; Do not use INC.B since SSP is a word register
```

Example The status byte of a process STATUS is incremented. When it is equal to eleven, a branch to OVFL is taken.

```
     INC.B  STATUS
     CMP.B  #11,STATUS
     JEQ    OVFL
```

* INCD[.W] Double-Increment destination
* INCD.B Double-Increment destination

Syntax

```
INCD      dst    or   INCD.W     dst
INCD.B    dst
```

Operation dst + 2 -> dst

Emulation

```
ADD       #2,dst
ADD.B     #2,dst
```

Description The destination operand is incremented by two. The original contents are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if dst contained 0FFFEh, reset otherwise
Set if dst contained 0FEh, reset otherwise
C: Set if dst contained 0FFFEh or 0FFFFh, reset otherwise
Set if dst contained 0FEh or 0FFh, reset otherwise
V: Set if dst contained 07FFEh or 07FFFh, reset otherwise
Set if dst contained 07Eh or 07Fh, reset otherwise

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The item on the top of the stack is removed without the use of a register.

```
.......
PUSH          R5       ; R5 is the result of a calculation, which is stored
                       ; in the system stack
INCD          SP       ; Remove TOS by double-increment from stack
                       ; Do not use INCD.B, SP is a word aligned
                       ; register
RET
```

Example The byte on the top of the stack is incremented by two.

```
INCD.B     0(SP)     ; Byte on TOS is increment by two
```

* INV[.W] Invert destination
* INV.B Invert destination

Syntax

```
INV      dst
INV.B    dst
```

Operation .NOT.dst -> dst

Emulation XOR #0FFFFh,dst
Emulation XOR.B #0FFh,dst

Description The destination operand is inverted. The original contents are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if dst contained 0FFFFh, reset otherwise
Set if dst contained 0FFh, reset otherwise
C: Set if result is not zero, reset otherwise (= .NOT. Zero)
Set if result is not zero, reset otherwise (= .NOT. Zero)
V: Set if initial destination operand was negative, otherwise reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example Content of R5 is negated (two's complement).

```
MOV     #00Aeh,R5     ;                        R5 = 000AEh
INV     R5            ; Invert R5,             R5 = 0FF51h
INC     R5            ; R5 is now negated,     R5 = 0FF52h
```

Example Content of memory byte LEO is negated.

```
MOV.B    #0AEh,LEO    ;                           MEM(LEO) = 0AEh
INV.B    LEO          ; Invert LEO,               MEM(LEO) = 051h
INC.B    LEO          ; MEM(LEO) is negated,      MEM(LEO) = 052h
```

JC
JHS

Jump if carry set
Jump if higher or same

Syntax

```
JC      label
JHS     label
```

Operation

if C = 1: PC + 2*offset -> PC
if C = 0: execute following instruction

Description The Carry Bit C of the Status Register is tested. If it is set, the 10-bit signed offset contained in the LSB's of the instruction is added to the Program Counter. If C is reset, the next instruction following the jump is executed. JC (jump if carry/higher or same) is used for the comparison of unsigned numbers (0 to 65536).

Status Bits Status bits are not affected

Example The signal of input P0IN.1 is used to define or control the program flow.

```
BIT     #10h,&P0IN   ; State of signal -> Carry
JC      PROGA        ; If carry=1 then execute program routine A
......               ; Carry=0, execute program here
```

Example R5 is compared to 15. If the content is higher or same branch to LABEL.

```
CMP     #15,R5
JHS     LABEL        ; Jump is taken if R5 ≥ 15
......               ; Continue here if R5 < 15
```

JEQ, JZ

Jump if equal, Jump if zero

Syntax JEQ label, JZ label

Operation

```
if Z = 1:  PC + 2*offset -> PC
if Z = 0: execute following instruction
```

Description The Zero Bit Z of the Status Register is tested. If it is set, the 10-bit signed offset contained in the LSB's of the instruction is added to the Program Counter. If Z is not set, the next instruction following the jump is executed.

Status Bits Status bits are not affected

Example Jump to address TONI if R7 contains zero.

```
TST     R7              ; Test R7
JZ      TONI            ; if zero: JUMP
```

Example Jump to address LEO if R6 is equal to the table contents.

```
CMP     R6,Table(R5)    ; Compare content of R6 with content of
                        ; MEM(Table address + content of R5)
JEQ     LEO             ; Jump if both data are equal
......                  ; No, data are not equal, continue here
```

Example Branch to LABEL if R5 is 0.

```
TST     R5
JZ      LABEL
......
```

JGE Jump if greater or equal

Syntax JGE label

Operation
if (N .XOR. V) = 0 then jump to label: PC + 2*offset -> PC
if (N .XOR. V) = 1 then execute following instruction

Description The negative bit N and the overflow bit V of the Status Register are tested. If both N and V are set or reset, the 10-bit signed offset contained in the LSB's of the instruction is added to the Program Counter. If only one is set, the next instruction following the jump is executed.
This allows comparison of signed integers.

Status Bits Status bits are not affected

Example When the content of R6 is greater or equal the memory pointed to by R7 the program continues at label EDE.

```
CMP     @R7,R6      ; R6 ≥ (R7)?, compare on signed numbers
JGE     EDE         ; Yes, R6 ≥ (R7)
......              ; No, proceed
......
......
```

JL
Jump if less

Syntax JL label

Operation
if (N .XOR. V) = 1 then jump to label: PC + 2*offset -> PC
if (N .XOR. V) = 0 then execute following instruction

Description The negative bit N and the overflow bit V of the Status Register are tested. If only one is set, the 10-bit signed offset contained in the LSB's of the instruction is added to the Program Counter. If both N and V are set or reset, the next instruction following the jump is executed.
This allows comparison of signed integers.

Status Bits Status bits are not affected

Example When the content of R6 is less than the memory pointed to by R7 the program continues at label EDE.

```
CMP     @R7,R6      ; R6 < (R7)?,  compare on signed numbers
JL      EDE         ; Yes, R6 < (R7)
......              ; No, proceed
......
......
```

JMP Jump unconditionally

Syntax JMP label

Operation PC + 2*offset -> PC

Description The 10-bit signed offset contained in the LSB's of the instruction is added to the Program Counter.

Status Bits Status bits are not affected

Hint This 1word instruction replaces the BRANCH instruction in the range of -511 to +512 words, relative to the current program counter.

JN

Jump if negative

Syntax JN label

Operation

```
if N = 1: PC + 2*offset -> PC
if N = 0: execute following instruction
```

Description The negative bit N of the Status Register is tested. If it is set, the 10-bit signed offset contained in the LSB's of the instruction is added to the Program Counter. If N is reset, the next instruction following the jump is executed.

Status Bits Status bits are not affected

Example The result of a computation in R5 is to be subtracted from COUNT. If the result is negative, COUNT is to be cleared and the program continues execution in another path.

```
        SUB     R5,COUNT        ; COUNT - R5 -> COUNT
        JN      L$1             ; If negative continue with COUNT=0at PC=L$1
        ......                  ; Continue with COUNT≥0
        ......
        ......
        ......
L$1     CLR     COUNT
        ......
        ......
        ......
```

JNC Jump if carry not set
JLO Jump if lower

Syntax

```
JNC     label
JNC     label
```

Operation

if C = 0: PC + 2*offset -> PC
if C = 1: execute following instruction

Description The Carry Bit C of the Status Register is tested. If it is reset, the 10-bit signed offset contained in the LSB's of the instruction is added to the Program Counter. If C is set, the next instruction following the jump is executed. JNC (jump if no carry/lower) is used for the comparison of unsigned numbers (0 to 65536).

Status Bits status bits are not affected

Example The result in R6 is added in BUFFER. If an overflow occurs an error handling routine at address ERROR is going to be used.

```
        ADD     R6,BUFFER       ; BUFFER + R6 -> BUFFER
        JNC     CONT            ; No carry, jump to CONT
ERROR   ......                  ; Error handler start
        ......
        ......
        ......
CONT    ......                  ; Continue with normal program flow
        ......
        ......
```

Example Branch to STL2 if byte STATUS contains 1 or 0.

```
        CMP.B   #2,STATUS
        JLO     STL2            ; STATUS < 2
        ......                  ; STATUS ≥ 2, continue here
```

JNE, JNZ

Jump if not equal, Jump if not zero

Syntax JNE label, JNZ label

Operation

if Z = 0: PC + 2*offset -> PC
if Z = 1: execute following instruction

Description The Zero Bit Z of the Status Register is tested. If it is reset, the 10-bit signed offset contained in the LSB's of the instruction is added to the Program Counter. If Z is set, the next instruction following the jump is executed.

Status Bits Status bits are not affected

Example Jump to address TONI if R7 and R8 have different contents

```
CMP     R7,R8               ; COMPARE R7 WITH R8
JNE     TONI                ; if different: Jump
......                      ; if equal, continue
```

MOV[.W] Move source to destination
MOV.B Move source to destination

Syntax MOV src,dst or MOV.W src,dst
MOV.B src,dst

Operation src -> dst

Description The source operand is moved to the destination.
The source operand is not affected, the previous contents of the destination are lost.

Status Bits Status bits are not affected

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The contents of table EDE (word data) are copied to table TOM. The length of the tables should be 020h locations.

```
        MOV    #EDE,R10                ; Prepare pointer
        MOV    #020h,R9                ; Prepare counter
Loop    MOV    @R10+,TOM-EDE-2(R10)    ; Use pointer in R10 for both tables
        DEC    R9                      ; Decrement counter
        JNZ    Loop                    ; Counter ≠ 0, continue copying
        ......                         ; Copying completed
        ......
        ......
```

Example The contents of table EDE (byte data) are copied to table TOM. The length of the tables should be 020h locations.

```
        MOV    #EDE,R10                ; Prepare pointer
        MOV    #020h,R9                ; Prepare counter
Loop    MOV.B  @R10+,TOM-EDE-1(R10)    ; Use pointer in R10 for
                                       ; both tables
        DEC    R9                      ; Decrement counter
        JNZ    Loop                    ; Counter ≠ 0, continue
                                       ; copying
        ......                         ; Copying completed
        ......
        ......
```

* NOP No operation

Syntax NOP

Operation None

Emulation MOV #0,#0

Description No operation is performed. The instruction may be used for the elimination of instructions during the software check or for defined waiting times.

Status Bits Status bits are not affected

The NOP instruction is mainly used for two purposes:

- hold one, two or three memory words
- adjust software timing

Note: Other instructions can be used to emulate no operation

Other instructions can be used to emulate no-operation instruction, using different numbers of cycles and different numbers of code words.

```
Examples:
MOV    0(R4),0(R4)       ; 6 cycles, 3 words
MOV    @R4,0(R4)         ; 5 cycles, 2 words
BIC    #0,EDE(R4)        ; 4 cycles, 2 words
JMP    $+2               ; 2 cycles, 1 word
BIC    #0,R5             ; 1 cycles, 1 word.
```

* POP[.W] Pop word from stack to destination
* POP.B Pop byte from stack to destination

Syntax

```
POP       dst
POP.B     dst
```

Operation

```
@SP   -> dst
SP + 2 -> SP
```

Emulation

```
MOV     @SP+,dst     or     MOV.W     @SP+,dst
```

Emulation

```
MOV.B    @SP+,dst
```

Description The stack location pointed to by the Stack Pointer (TOS) is moved to the destination. The Stack Pointer is incremented by two afterwards.

Status Bits Status bits are not affected

Example The contents of R7 and the Status Register are restored from the stack.

```
POP     R7              ; Restore R7
POP     SR              ; Restore status register
```

Example The content of RAM byte LEO is restored from the stack.

```
POP.B     LEO          ; The Low byte of the stack is moved to LEO.
```

Example The content of R7 is restored from the stack.

```
POP.B     R7            ; The Low byte of the stack is moved to R7,
                        ; the High byte of R7 is 00h
```

Example The contents of the memory pointed to by R7 and the Status Register are restored from the stack.

```
POP.B     0(R7)         ; The Low byte of the stack is moved to the
                        ; the byte which is pointed to by R7
                        : Example:  R7 = 203h
                        ;           Mem(R7) = Low Byte of system stack
                        : Example:   R7 = 20Ah
                        ;           Mem(R7) = Low Byte of system stack
POP       SR
```

Note: The system Stack Pointer

The system Stack Pointer SP is always incremented by two, independent of the byte suffix. This is mandatory since the system Stack Pointer is used not only by POP instructions; it is also used by the RETI instruction.

PUSH[.W] Push word onto stack
PUSH.B Push byte onto stack

Syntax

```
PUSH    src   or   PUSH.W   src
PUSH.B  src
```

Operation

SP - 2 → SP
src → @SP

Description The Stack Pointer is decremented by two, then the source operand is moved to the RAM word addressed by the Stack Pointer (TOS).

Status Bits
N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The contents of the Status Register and R8 are saved on the stack.

```
PUSH    SR          ; save status register
PUSH    R8          ; save R8
```

Example The content of the peripheral TCDAT is saved on the stack.

```
PUSH.B  &TCDAT      ; save data from 8bit peripheral module,
                    ; address TCDAT, onto stack
```

Note: The system Stack Pointer

The system Stack Pointer SP is always decremented by two, independent of the byte suffix. This is mandatory since the system Stack Pointer is used not only by PUSH instruction; it is also used by the interrupt routine service.

* RET Return from subroutine

Syntax RET

Operation @SP→ PC
SP + 2 → SP

Emulation MOV @SP+,PC

Description The return address pushed onto the stack by a CALL instruction is moved to the Program Counter. The program continues at the code address following the subroutine call.

Status Bits Status bits are not affected

RETI Return from Interrupt

Syntax RETI

Operation

```
TOS      → SR
SP + 2   → SP
TOS      → PC
SP + 2   → SP
```

Description

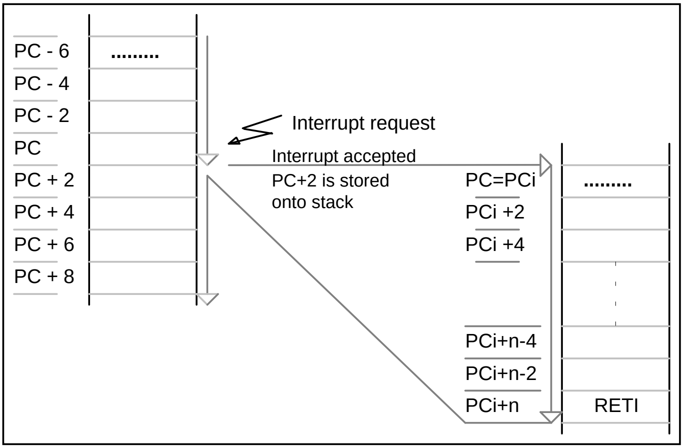
1. The status register is restored to the value at the beginning of the interrupt service routine. This is performed by replacing the present contents of SR with the contents of TOS memory. The stack pointer SP is incremented by two.
2. The program counter is restored to the value at the beginning of interrupt service. This is the consecutive step after the interrupted program flow. Restore is performed by replacing present contents of PC with the contents of TOS memory. The stack pointer SP is incremented.

Status Bits

N: restored from system stack
Z: restored from system stack
C: restored from system stack
V: restored from system stack

Mode Bits **OscOff**, **CPUOff** and **GIE** are restored from system stack

Example Main program is interrupted



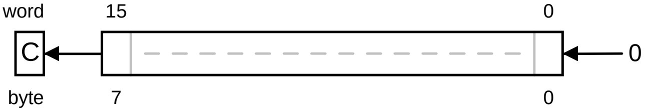
* RLA[.W] Rotate left arithmetically
* RLA.B Rotate left arithmetically

Syntax RLA dst or RLA.W dst
RLA.B dst

Operation C <- MSB <- MSB-1 .... LSB+1 <- LSB <- 0

Emulation ADD dst,dst
ADD.B dst,dst

Description The destination operand is shifted left one position. The MSB is shifted into the carry C, the LSB is filled with 0. The RLA instruction acts as a signed multiplication with 2.
An overflow occurs if dst ≥ 04000h and dst < 0C000h before operation is performed: the result has changed sign.



An overflow occurs if dst ≥ 040h and dst < 0C0h before operation is performed: the result has changed sign.

Status Bits **N:** Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Loaded from the MSB
V: Set if an arithmetic overflow occurs -
the initial value is 04000h ≤ dst < 0C000h; otherwise it is reset
Set if an arithmetic overflow occurs:
the initial value is 040h ≤ dst < 0C0h; otherwise it is reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

* RLA (continued)

Example R7 is multiplied by 4.

```
RLA     R7          ; Shift left R7  (x 2) - emulated by   ADD  R7,R7
RLA     R7          ; Shift left R7  (x 4) - emulated by   ADD  R7,R7
```

Example Lowbyte of R7 is multiplied by 4.

```
RLA.B     R7        ; Shift left Lowbyte of R7  (x 2) - emulated by
                    ; ADD.B  R7,R7
RLA.B     R7        ; Shift left Lowbyte of R7  (x 4) - emulated by
                    ; ADD.B  R7,R7
```

Note: RLA substitution

The Assembler does not recognize the instruction

```
  RLA     @R5+              nor       RLA.B     @R5+.
```

It must be substituted by

```
  ADD     @R5+,-2(R5)       or        ADD.B     @R5+,-1(R5).
```

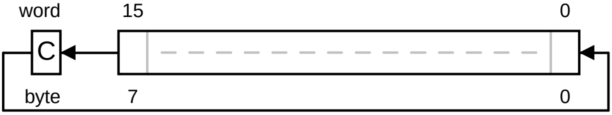
* RLC[.W] Rotate left through carry
* RLC.B Rotate left through carry

Syntax RLC dst or RLC.W dst
RLC.B dst

Operation C <- MSB <- MSB-1 .... LSB+1 <- LSB <- C

Emulation ADDC dst,dst

Description The destination operand is shifted left one position. The carry C is shifted into the LSB, the MSB is shifted into the carry C.



Status Bits **N:** Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Loaded from the MSB
V: Set if arithmetic overflow occurs otherwise reset
Set if 03FFFh < $dst_{initial}$ < 0C000h, otherwise reset
Set if 03Fh < $dst_{initial}$ < 0C0h, otherwise reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

* RLC (continued)

Example R5 is shifted left one position.

```
RLC      R5            ; (R5 x 2) + C -> R5
```

Example The information of input P0IN.1 is to be shifted into LSB of R5.

```
BIT.B    #2,&P0IN      ; Information -> Carry
RLC      R5            ; Carry=P0in.1 -> LSB of R5
```

Example Content of MEM(LEO) is shifted left one position.

```
RLC.B    LEO           ; Mem(LEO) x 2 + C -> Mem(LEO)
```

Example The information of input P0IN.1 is to be shifted into LSB of R5.

```
BIT.B    #2,&P0IN      ; Information -> Carry
RLC.B    R5            ; Carry=P0in.1 -> LSB of R5
                       ; High byte of R5 is reset
```

Note: RLC and RLC.B emulation

The Assembler does not recognize the instruction

```
  RLC    @R5+.
```

It must be substituted by

```
  ADDC   @R5+,-2(R5).
```

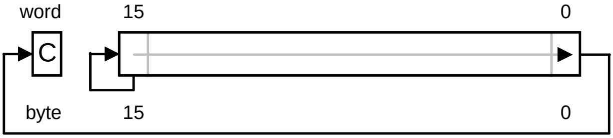
B

RRA[.W] Rotate right arithmetically
RRA.B Rotate right arithmetically

Syntax RRA dst or RRA.W dst
RRA.B dst

Operation MSB -> MSB, MSB -> MSB-1, ... LSB+1 -> LSB, LSB -> C

Description The destination operand is shifted right one position. The MSB is shifted into the MSB, the MSB is shifted into the MSB-1, the LSB+1 is shifted into the LSB.



Status Bits
N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Loaded from the LSB
V: Reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

RRA (continued)

Example R5 is shifted right one position. The MSB remains with the old value. It operates equal to an arithmetic division by 2.

```
              RRA      R5          ; R5/2 -> R5

;             The value in R5 is multiplied by 0.75 (0.5 + 0.25)
;
              PUSH     R5          ; hold R5 temporarily using stack
              RRA      R5          ; R5 x 0.5  ->  R5
              ADD      @SP+,R5     ; R5 x 0.5 + R5 = 1.5 x R5  -> R5
              RRA      R5          ; (1.5 x R5) x 0.5 = 0.75 x R5  -> R5
              ......
              ......
; OR
;
              RRA      R5          ; R5 x 0.5  ->  R5
              PUSH     R5          ; R5 x 0.5  ->  TOS
              RRA      @SP         ; TOS x 0.5 = 0.5 x R5 x 0.5 = 0.25 x R5  -> TOS
              ADD      @SP+,R5     ; R5 x 0.5 + R5 x 0.25 = 0.75 x R5  -> R5
              ......
```

Example The Lowbyte of R5 is shifted right one position. The MSB remains with the old value. It operates equal to an arithmetic division by 2.

```
              RRA.B    R5           ; R5/2 -> R5: Operation is on Low byte only
                                    ; High byte of R5 is reset

;             The value in R5 - Low byte only! - is multiplied by 0.75 (0.5 + 0.25)
;
              PUSH.B   R5           ; hold Low byte of R5 temporarily using stack
              RRA.B    R5           ; R5 x 0.5  ->  R5
              ADD.B    @SP+,R5      ; R5 x 0.5 + R5 = 1.5 x R5  -> R5
              RRA.B    R5           ; (1.5 x R5) x 0.5 = 0.75 x R5  -> R5
              ......
; OR
;
              RRA.B    R5           ; R5 x 0.5  ->  R5
              PUSH.B   R5           ; R5 x 0.5  ->  TOS
              RRA.B    @SP          ;TOS x 0.5 = 0.5 x R5 x 0.5 = 0.25x R5  -> TOS
              ADD.B    @SP+,R5      ; R5 x 0.5 + R5 x 0.25 = 0.75 x R5  -> R5
              ......
```

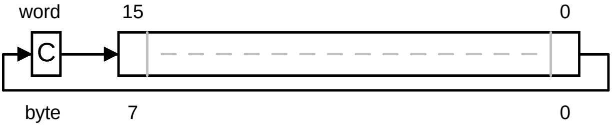
RRC[.W] Rotate right through carry
RRC.B Rotate right through carry

Syntax

```
RRC     dst     or     RRC.W     dst
RRC     dst
```

Operation C -> MSB -> MSB-1 .... LSB+1 -> LSB -> C

Description The destination operand is shifted right one position. The carry C is shifted into the MSB, the LSB is shifted into the carry C.



Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Loaded from the LSB
V: Set if initial destination is positive and initial Carry is set, otherwise reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example R5 is shifted right one position. The MSB is loaded with 1.

```
SETC                ; PREPARE CARRY FOR MSB
RRC     R5          ; R5/2 + 8000h -> R5
```

Example R5 is shifted right one position. The MSB is loaded with 1.

```
SETC                ; PREPARE CARRY FOR MSB
RRC.B   R5          ; R5/2 + 80h -> R5; Low byte of R5 is used
```

* SBC[.W] Subtract borrow*) from destination
* SBC.B Subtract borrow*) from destination

Syntax

```
SBC     dst    or   SBC.W     dst
SBC.B   dst
```

Operation

dst + 0FFFFh + C -> dst
dst + 0FFh + C -> dst

Emulation

```
SUBC     #0,dst
SUBC.B   #0,dst
```

Description The carry C is added to the destination operand minus one. The previous contents of the destination are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Reset if dst was decremented from 0000 to 0FFFFh, set otherwise
Reset if dst was decremented from 00 to 0FFh, set otherwise
V: Set if initially C=0 and dst=08000h
Set if initially C=0 and dst=080h

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The 16-bit counter pointed to by R13 is subtracted from a 32-bit counter pointed to by R12.

```
SUB      @R13,0(R12)      ; Subtract LSDs
SBC      2(R12)           ; Subtract carry from MSD
```

Example The 8bit counter pointed to by R13 is subtracted from a 16bit counter pointed to by R12.

```
SUB.B    @R13,0(R12)      ; Subtract LSDs
SBC.B    1(R12)           ; Subtract carry from MSD
```

Note: Borrow is treated as a .NOT. carry

The borrow is treated as a .NOT. carry:

Borrow	Carry bit
Yes	0
No	1

* SETC Set carry bit

Syntax SETC

Operation 1 -> C

Emulation BIS #1,SR

Description The Carry Bit C is set, an operation which is often necessary.

Status Bits
N: Not affected
Z: Not affected
C: Set
V: Not affected

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example Emulation of the decimal subtraction:
Subtract R5 from R6 decimally
Assume that R5=3987 and R6=4137

```
DSUB        ADD     #6666h,R5       ; Move content R5 from 0-9 to 6-0Fh
                                    ; R5 = 03987 + 6666 = 09FEDh
            INV     R5              ; Invert this(result back to 0-9)
                                    ; R5 = .NOT. R5 = 06012h
            SETC                    ; Prepare carry = 1
            DADD    R5,R6           ; Emulate subtraction by adding of:
                                    ; (10000 - R5 - 1)
                                    ; R6 = R6 + R5 + 1
                                    ; R6 = 4137 + 06012 + 1 = 1 0150 = 0150
```

* SETN Set Negative bit

Syntax SETN

Operation 1 -> N

Emulation BIS #4,SR

Description The Negative bit N is set.

Status Bits
N: Set
Z: Not affected
C: Not affected
V: Not affected

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

* SETZ Set Zero bit

Syntax SETZ

Operation 1 -> Z

Emulation BIS #2,SR

Description The Zero bit Z is set.

Status Bits
N: Not affected
Z: Set
C: Not affected
V: Not affected

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

SUB[.W] subtract source from destination
SUB.B subtract source from destination

Syntax

```
SUB      src,dst      or   SUB.W      src,dst
SUB.B              src,dst
```

Operation

dst + .NOT.src + 1 -> dst
or
[(dst - src -> dst)]

Description The source operand is subtracted from the destination operand. This is made by adding the 1's complement of the source operand and the constant 1. The source operand is not affected, the previous contents of the destination are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if there is a carry from the MSB of the result, reset if not
Set to 1 if no borrow, reset if borrow.
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example See example at the SBC instruction

Example See example at the SBC.B instruction

Note: Borrow is treated as a .NOT. carry

The borrow is treated as a .NOT. carry:

Borrow	Carry bit
Yes	0
No	1

SUBC[.W]SBB[.W] subtract source and borrow/.NOT. carry from destination
SUBC.B,SBB.B subtract source and borrow/.NOT. carry from destination

Syntax

```
SUBC     src,dst   or  SUBC.W   src,dst   or
SBB      src,dst   or  SBB.W    src,dst
SUBC.B   src,dst   or  SBB.B    src,dst
```

Operation

dst + .NOT.src + C -> dst
or
(dst - src - 1 + C -> dst)

Description The source operand is subtracted from the destination operand. This is made by adding of the 1's complement of the source operand and the carry C. The source operand is not affected, the previous contents of the destination are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if there is a carry from the MSB of the result, reset if not
Set to 1 if no borrow, reset if borrow.
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example Two floating point mantissas (24bits) are subtracted .
LSB's are in R13 resp. R10, MSB's are in R12 resp. R9.

```
SUB.W    R13,R10          ; 16bit part, LSB's
SUBC.B   R12,R9           ;   8bit part, MSB's
```

Example The 16-bit counter pointed to by R13 is subtracted from a 16-bit counter in R10 and R11(MSD).

```
SUB.B      @R13+,R10     ; Subtract LSDs without carry
SUBC.B     @R13,R11      ; Subtract MSDs with carry
...                      ; resulting fron the LSDs
```

Note: Borrow is treated as a .NOT. carry

The borrow is treated as a .NOT. carry:

Borrow	Carry bit
Yes	0
No	1

SWPB Swap bytes

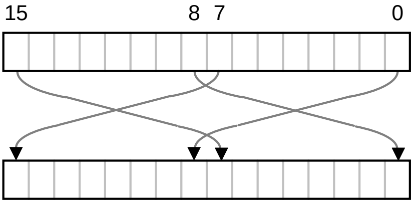
Syntax SWPB dst

Operation bits 15 to 8 <-> bits 7 to 0

Description The high and the low bytes of the destination operand are exchanged.

Status Bits
N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected



Example

```
MOV   #040BFh,R7        ; 0100000010111111 -> R7
SWPB  R7                ; 1011111101000000 in R7
```

Example The value in R5 is multiplied by 256. The result is stored in R5,R4

```
SWPB  R5                ;
MOV   R5,R4             ;Copy the swapped value to R4
BIC   #0FF00h,R5        ;Correct the result
BIC   #00FFh,R4         ;Correct the result
```

SXT Extend Sign

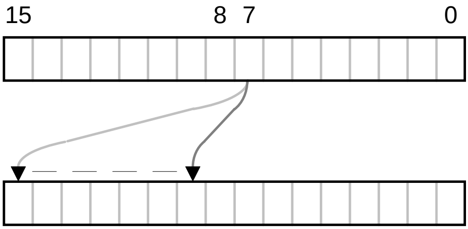
Syntax SXT dst

Operation Bit 7 -> Bit 8 .... Bit 15

Description The sign of the Low byte is extended into the High byte.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if result is not zero, reset otherwise (.NOT. Zero)
V: Reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected



Example R7 is loaded with Timer/Counter value. The operation of the sign extend instruction expands the bit8 to bit15 with the value of bit7.
R7 is added then to R6 where it is accumulated.

```
MOV.B   &TCDAT,R7     ; TCDAT = 080h:  . . . .   . . . . 1000 0000
SXT     R7            ; R7 = 0FF80h:     1111 1111 1000 0000
ADD     R7,R6         ; add value of EDE to 16bit ACCU
```

* TST[.W] Test destination
* TST.B Test destination

Syntax

```
TST      dst     or    TST.W    dst
TST.B    dst
```

Operation

dst + 0FFFFh + 1
dst + 0FFh + 1

Emulation

```
CMP      #0,dst
CMP.B    #0,dst
```

Description The destination operand is compared to zero. The status bits are set according to the result. The destination is not affected.

Status Bits

N: Set if destination is negative, reset if positive
Z: Set if destination contains zero, reset otherwise
C: Set
V: Reset.

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example R7 is tested. If it is negative continue at R7NEG; if it is positive but not zero continue at R7POS.

```
            TST     R7              ; Test R7
            JN      R7NEG           ; R7 is negative
            JZ      R7ZERO          ; R7 is zero
R7POS       ......                  ; R7 is positive but not zero

R7NEG       ......                  ; R7 is negative

R7ZERO      ......                  ; R7 is zero
```

Example Lowbyte of R7 is tested. If it is negative continue at R7NEG; if it is positive but not zero continue at R7POS.

```
            TST.B   R7              ; Test Low byte of R7
            JN      R7NEG           ; Low byte of R7 is negative
            JZ      R7ZERO          ; Low byte of R7 is zero
R7POS       ......                  ; Low byte of R7 is positive but not zero

R7NEG       .....                   ; Lowbyte of R7 is negative

R7ZERO      ......                  ; Lowbyte of R7 is zero
```

XOR[.W] Exclusive OR of source with destination
XOR.B Exclusive OR of source with destination

Syntax

```
XOR     src,dst    or    XOR.W    src,dst
XOR.B                    src,dst
```

Operation src .XOR. dst -> dst

Description The source operand and the destination operand are OR'ed exclusively. The result is placed into the destination. The source operand is not affected.

Status Bits

N: Set if MSB of result is set, reset if not set
Z: Set if result is zero, reset otherwise
C: Set if result is not zero, reset otherwise (= .NOT. Zero)
V: Set if both operands are negative

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The bits set in R6 toggle the bits in the RAM word TONI.

```
XOR      R6,TONI      ; Toggle bits of word TONI on the bits set in R6
```

Example The bits set in R6 toggle the bits in the RAM byte TONI.

```
XOR.B R6,TONI         ; Toggle bits in word TONI on bits
                      ; set in Low byte of R6,
```

Example Reset bits in Lowbyte of R7 to 0 that are different to bits in RAM byte EDE.

```
XOR.B     EDE,R7      ; Set different bit to '1s'
INV.B     R7          ; Invert Lowbyte, Highbyte is 0h
```

Macro instructions emulated with several instructions

The following table shows the instructions which need more words if emulated by the reduced instruction set. This is not of great concern, because they are rarely used. The immediate values -1, 0, +1, 2, 4 and 8 are provided by the Constant Generator Registers R2/CG1 and R3/CG2.

Emulated instruction			**Instruction flow**		**Comment**
ABS	**dst**		TST	dst	; Absolute value of destination
			JN	L$0	; Destination is negative
		L$1	...		; Destination is positive
			...		
			...		
		L$0	INV	dst	; Convert negative destination
			INC	dst	; to positive
			JMP	L$1	
DSUB	**src,dst**		ADD	#6666h,src	; Decimal subtraction
			INV	src	; Source is destroyed!
			SETC		
			DADD	src,dst	; DST - SRC (dec)
NEG	**dst**		INV	dst	; Negation of destination
			INC	dst	
RL	**dst**		ADD	dst,dst	; Rotate left circularly
			ADDC	#0,dst	
RR	**dst**		CLRC		; Rotate right circularly
			RRC	dst	
			JNC	L$1	
			BIS	#8000h,dst	
		L$1	...		

C. EPROM Programming

This appendix describes the MSP430 EPROM module. The EPROM module is erasable with ultraviolet light, and electrically programmable. Devices with an EPROM module are offered in a windowed package for multiple programming and OTP package, for one time programmable.

C.1 EPROM Operation

The CPU can fetch data and instructions from the EPROM. When the programming voltage is applied to the TDI/VPP pin, the CPU can also write to the EPROM module. Reading the EPROM is an identical process to that with other internal peripheral modules. Both programming and reading can occur on byte or word boundaries.

Erasure

Before programming, the entire EPROM should be erased. Erasing of the EPROM module is achieved by exposing the transparent window to ultraviolet light.

Note: EPROM exposed to ambient light

Normal ambient light contains the correct wavelength for erasure. When a device with a transparent window is programmed for use the window should be covered with an opaque label.

Exposing the EPROM module to ultraviolet light will also cause erasure in the EEPROM module, if it is on-chip. Any useful data in the EEPROM module must be reprogrammed after exposure to ultraviolet light.

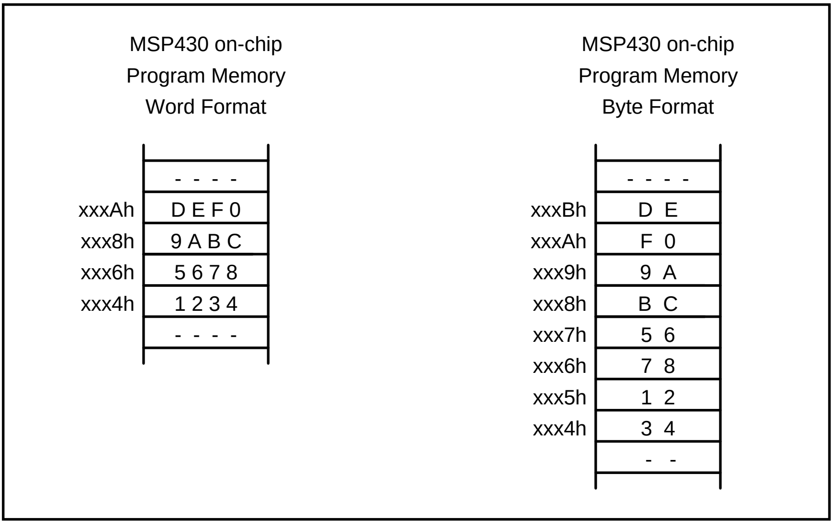
The data in the EPROM module can be programmed serially via the integrated ‘JTAG’ feature, or via software which is part of the application software. The ‘JTAG’ implementation features an internal mechanism for security purposes. Once the ‘security fuse’ is activated, no accesses to the device via the ‘JTAG’ functions are possible. The ‘JTAG’ is permanently switched to the by-pass mode.

Programming

The application must provide an external voltage supply to the TDI/VPP pin, to provide the necessary voltage and current for programming. The minimum programming time is noted in the electrical characteristics of the device data sheets.

The EPROM control register EPCTL controls the EPROM programming, once the external voltage is supplied. The erase state is a ‘1’. When EPROM bits are programmed, they are read as ‘0’.

The programming of the EPROM module can be done for single bytes, words, blocks of individual length, or with the entire module. All bits that have a final level of ‘1’ must be erased before programming. The programming can be done on single devices or even in-system. The supply voltage should be in the range required by the device datasheet. The levels on the ‘JTAG’ pins are defined in the device datasheet, and are usually CMOS levels.



MSP430 on-chip
Program Memory
Word Format

	- - - -
xxxAh	D E F 0
xxx8h	9 A B C
xxx6h	5 6 7 8
xxx4h	1 2 3 4
	- - - -

MSP430 on-chip
Program Memory
Byte Format

	- - - -
xxxBh	D E
xxxAh	F 0
xxx9h	9 A
xxx8h	B C
xxx7h	5 6
xxx6h	7 8
xxx5h	1 2
xxx4h	3 4
	- -

EPROM Control Register EPCTL

	7							0
EPCTL 054h							VPPS	EXE
	r-0	r-0	r-0	r-0	r-0	r-0	rw-0	rw-0

Bit 0: The execute bit EXE initiates and ends the programming to the EPROM module. The external voltage must be supplied to the TDI/VPP before EXE bit is set. The timing conditions are noted in the datasheets.

Bit 1: When the VPPS bit is set, the external programming voltage is connected to the EPROM module. The VPPS bit must be set before EXE bit is set. It can be reset together with the EXE bit. The VPPS bit must not be cleared between programming operations.

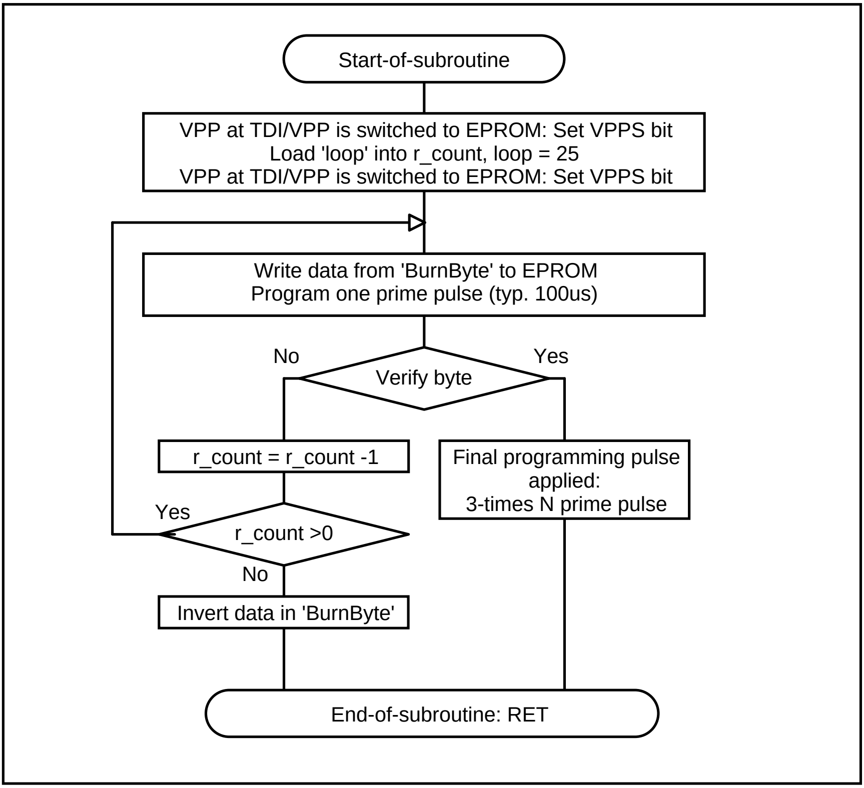
EPROM Protect

The EPROM access via the serial test and programming interface ‘JTAG’ can be inhibited when the ‘security fuse’ is activated. The security fuse is activated via serial instructions shifted into the ‘JTAG’. Activating the fuse is not reversible and any access to the internal system is disrupted. The by-pass function described in the standard IEEE1149.1 is active.

C.2 FAST Programming Algorithm

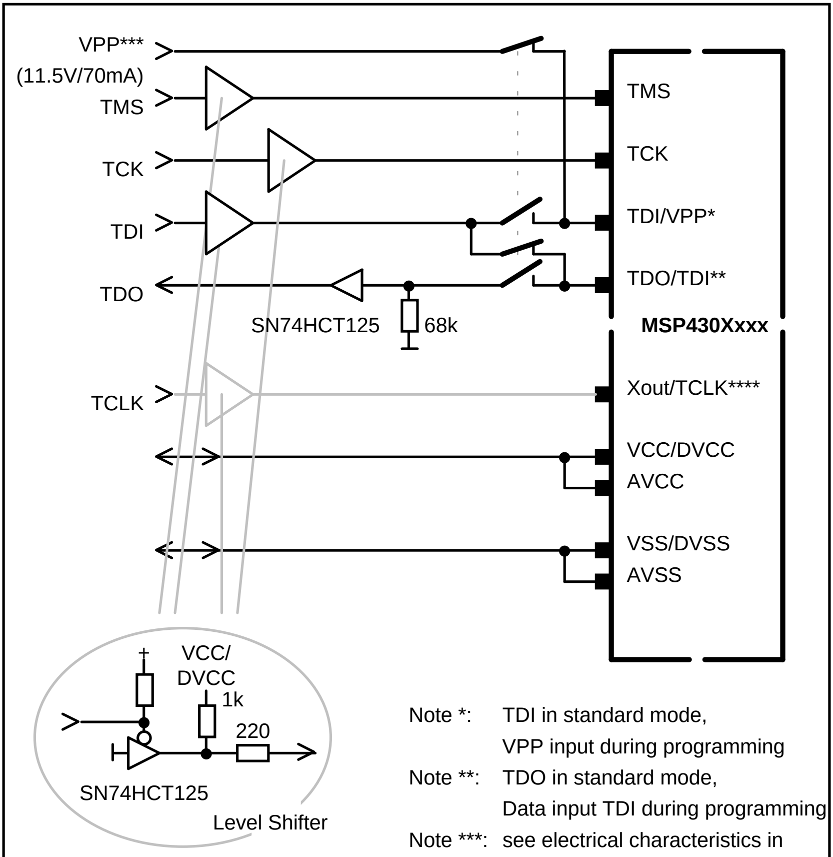
The FAST programming cycle is normally used to program the data into the EPROM. A programmed logical ‘0’ can be erased only by ultraviolet light.

Fast programming uses two types of pulses: prime and final. The length of the prime puls is typically 100µs (see the latest datasheet). After each prime pulse, the programmed data is verified. If it fails 25 times, the programming operation was false. If correct data is read, the final programming pulse is applied; the final programming pulse is 3 times the number of prime pulses applied.


Start-of-subroutine
VPP at TDI/VPP is switched to EPROM: Set VPPS bit
Load 'loop' into r_count, loop = 25
VPP at TDI/VPP is switched to EPROM: Set VPPS bit
Write data from 'BurnByte' to EPROM
Program one prime pulse (typ. 100us)
No
Verify byte
Yes
r_count = r_count -1
Final programming pulse applied:
3-times N prime pulse
Yes
r_count >0
No
Invert data in 'BurnByte'
End-of-subroutine: RET

C3. Program EPROM module via serial data link using ‘JTAG’ feature

The hardware interconnection of the ‘JTAG’ pins is done via four separate pins, plus the ground or VSS reference level. The ‘JTAG’ pins are TMS, TCK, TDI(/VPP) and TDO(/TDI).



Note *: TDI in standard mode,
VPP input during programming

Note **: TDO in standard mode,
Data input TDI during programming

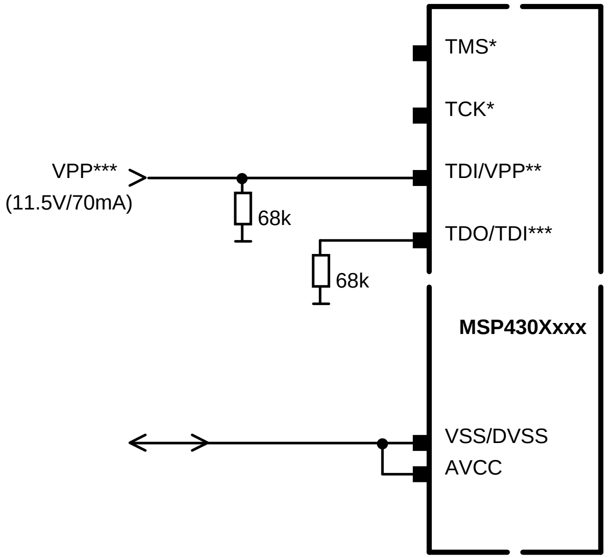
Note ***: see electrical characteristics in
the latest data sheet

Note ****: Optional, fast incrementing of
address via PC possible

Switches shown for programming situation

C4. Programming EPROM module via controller’s software

The hardware needed to program an EPROM module is quite simple: connect the required supply to the TDI/VPP pin, and run the proper software algorithm. The software algorithm that controls the EPROM programming cycle can not run in the same EPROM module to which the data should be written. It is impossible to read instructions from the EPROM and write data to it at the same time. The software needs to run from another memory - from a ROM module, a RAM module or another EPROM module.



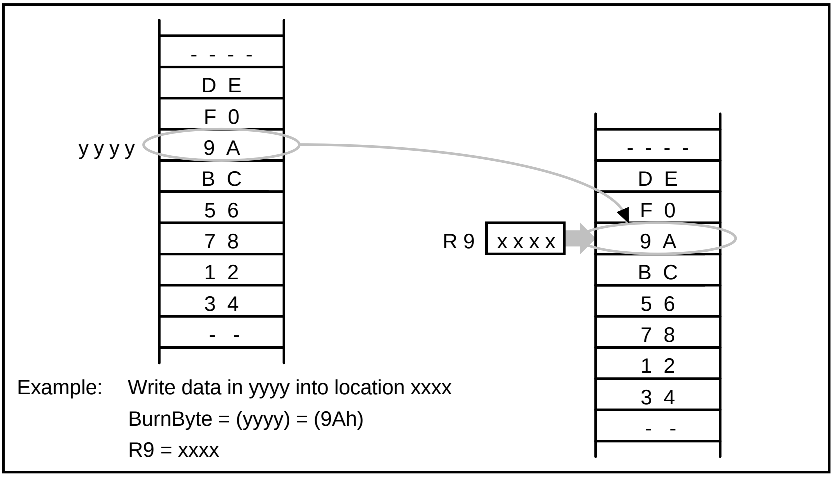
Note *: Internally a pull-up resistor is connected to TMS and TCK

Note **: ROM devices of MSP430 have an internal pull-up resistor at pin TDI/VPP MSP430Pxxx or MSP430Exxx have no internal pull-up resistor. They should have an ext. pull-down resistor preventing floating input node.

Note ***: The TDO/TDI pin should have an ext. pull-down resistor preventing floating input node for secondary TDI function.

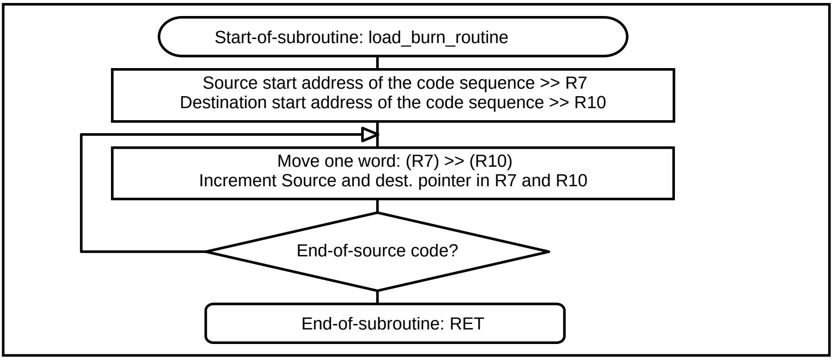
Programming EPROM module via controller’s software, Example;

The software example writes one byte into the EPROM with the fast programming algorithm. The code is written position-independent, and will have been loaded (e.g. to the RAM) before it is used. The programming algorithm runs during the programming sequence in the RAM, thus avoiding conflict when the EPROM is written. The data (byte) which should be written is located in the RAM address ‘BurnByte’, and the target address of the EPROM module is held in the register ‘pointer’ defined with set directive. The timing is adjusted to a cycle time of 1µs. When another cycle time / processor frequency is selected, the software should be adjusted according to the operating conditions.



Example: Write data in yyyy into location xxxx
BurnByte = (yyyy) = (9Ah)
R9 = xxxx

The target EPROM module can not execute the programming code sequence while the data is being written into it. In the example, a subroutine moves the programming code sequence into another memory, e.g. into the RAM.


Start-of-subroutine: load_burn_routine
Source start address of the code sequence >> R7
Destination start address of the code sequence >> R10
Move one word: (R7) >> (R10)
Increment Source and dest. pointer in R7 and R10
End-of-source code?
End-of-subroutine: RET

```
;-------------------------------------------------------------
; Definitions used in Subroutine :
; Move programming code sequence into RAM (load_burn_routine)
; Burn a byte into the EPROM area         (RELOC_Burn_EPROM)
;-------------------------------------------------------------

EPCTL     .set   054h   ; EPROM Control Register
VPPS      .set   2      ; Program Voltage bit
EXE       .set   1      ; Execution bit
BurnByte  .set   0220h  ; address of data to be written
Burn_orig .set   0222h  ; Start address of burn
                        ; program in the RAM
loops     .set   25
r_timer   .set   r8     ; 1us = 1 cycle
pointer   .set   r9     ; pointer to the EPROM address
                        ; r9 is saved in the main routine
                        ; before subroutine call is executed
r_count   .set   r10
lp        .set   3      ; dec r_timer   : 1 cycle  : loop_t100
                        ; jnz           : 2 cycles : loop_t100
ov        .set   2      ; mov #(100-ov)/lp,r_timer : 2 cycles
```

; Load EPROM programming sequence to another location e.g. RAM, Subroutine

```
;--- Burn subroutine: position independent code!

RAM_Burn_EPROM .set    Burn_orig
load_burn_routine
       push   r9
       push   r10
       mov    #Burn_EPROM,R9        ; load pointer source
       mov    #RAM_Burn_EPROM,R10   ; load pointer dest.
load_burn1
       mov    @R9,0(R10)            ; move a word
       incd   R10                   ; dest.  pointer + 2
       incd   R9                    ; source pointer + 2
       cmp     #Burn_end,R9           ; compare to end_of_table
       jne     load_burn1
       pop     r9
       pop     r10
       ret
```

; Program one byte into EPROM, Subroutine

```
Burn_EPROM
       dint                           ;   ensure   correct   burn
timing
       mov.b   #VPPS,&EPCTL           ; VPPS on
       push    r_timer                ; save registers
       push    r_count                ; programming subroutine
       mov     #loops,r_count         ; 2 cycles = 2 us
Repeat_Burn
       mov.b   &BurnByte,0(pointer)   ; write to data to EPROM
                                      ; 6 cycles = 6 us
       bis.b   #EXE,&EPCTL            ; EXE on
                                      ; 4 cycles = 4 us
                                      ; total cycles VPPon to EXE
                                      ; 12 cycles = 12 us (min.)
       mov     #(100-ov)/lp,r_timer   ;:programming pulse of
100us
wait_100                              ;:starts,    actual    time
102us
       dec     r_timer                ;:
       jnz     wait_100               ;:
       bic.b   #EXE,&EPCTL            ;:EXE / prog. puls off

       mov     #4,r_timer             ;:wait min. 10 us
wait_10                               ;:before verifying
       dec     r_timer                ;:programmed EPROM
       jnz     wait_10                ;:location, actual 13+ us

       cmp.b   &BurnByte,0(pointer)   ; verify data = burned data
       jne     Burn_EPROM_bad         ; data ‡ burned data > jump

; Continue here when data correctly burned into EPROM location
       mov.b   &BurnByte,0(pointer)   ; write to EPROM again
       bis.b   #EXE,&EPCTL            ; EXE on
       add     #(0ffffh-loop),r_count; Number of loops for
                                      ; successful programming
final_puls
       mov     #(300-ov)/lp,r_timer   ;:programming pulse of
wait_300                              ;:3*100us*N starts
       dec     r_timer                ;:
       jnz     wait_300               ;:
       inc     r_count                ;:
       jn      final_puls             ;:
       clr.b   &EPCTL                 ;:EXE off / VPPS off
       jmp      Burn_EPROM_end
```

```
Burn_EPROM_bad
       dec    r_count        ; not ok : decrement loop counter
       jnz    Repeat_Burn    ; loop not ended : do another trial
       inv.b  &BurnByte      ; return the inverted data to flag
                             ; failing the programming attempt
                             ; the EPROM address is unchanged
                             ;
Burn_EPROM_end
       pop    r_timer
       pop    r_count
       eint
       ret
Burn_end
```