CPE 323 Introduction to Embedded Computer Systems: The MSP430 System Architecture

Instructor: Dr Aleksandar Milenkovic
Lecture Notes

Outline

MSP430: System Architecture

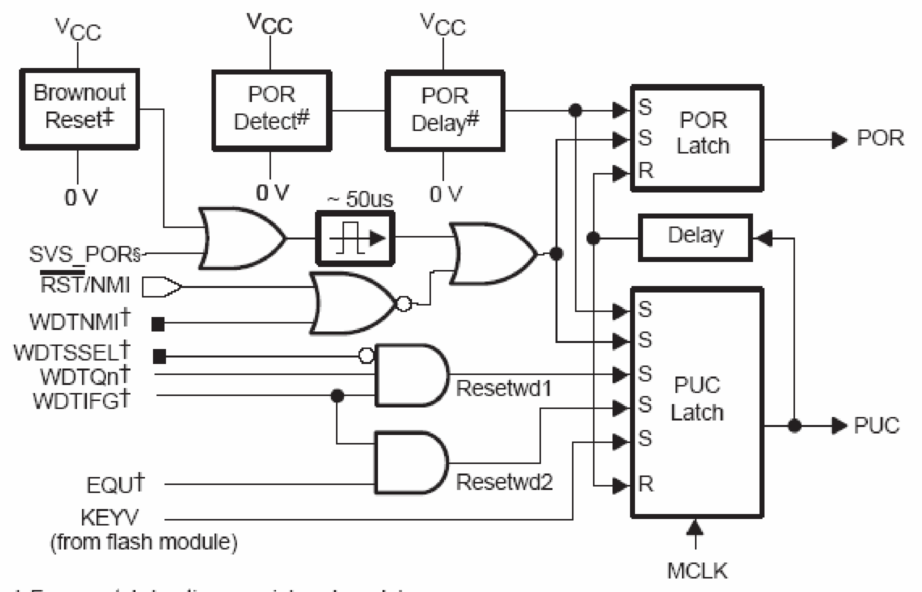
- System Resets, Interrupts, and Operating Modes
- Basic Clock Module
- Watchdog Timer

MSP430: System Resets, Interrupts, and Operating Modes

System Reset

- Power-on Reset (POR)
  - Powering up the device
  - A low signal on the RST/NMI pin when configured in the reset mode
  - An SVS low condition when PORON=1.
- Power-up Clear
  - A POR signal
  - Watchdog timer expiration when in watchdog mode only
  - Watchdog timer security key violation
  - A Flash memory security key violation

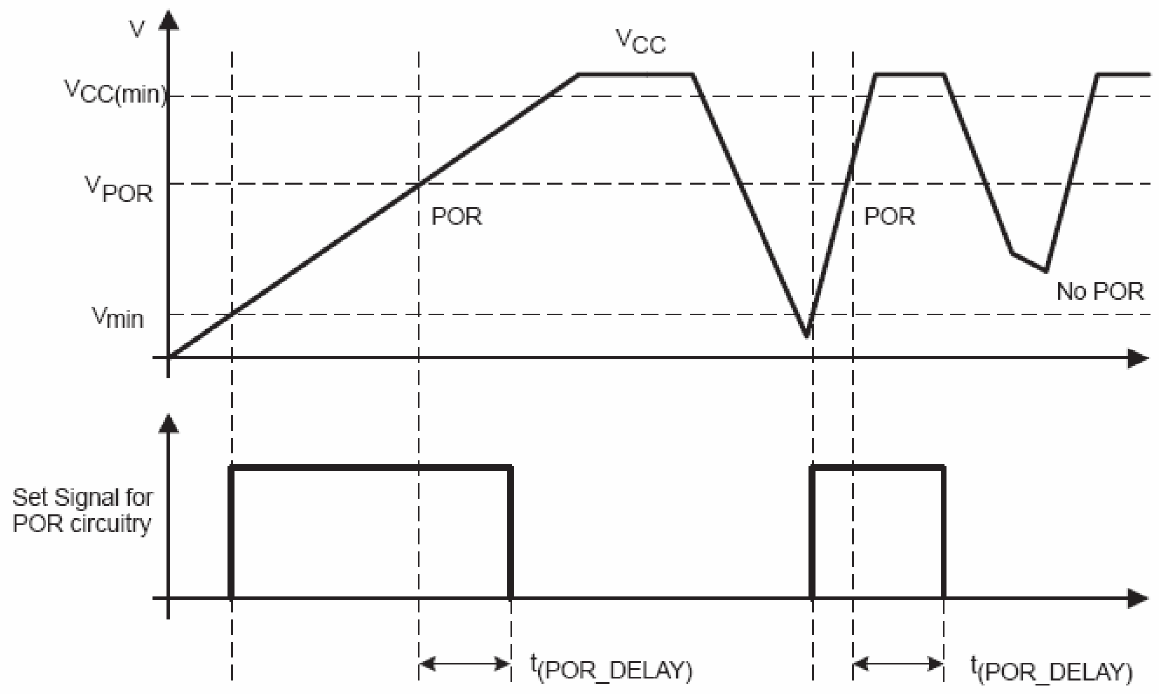
Figure 2−1. Power-On Reset and Power-Up Clear Schematic



† From watchdog timer peripheral module
‡ Devices with BOR only
Devices without BOR only
§ Devices with SVS only

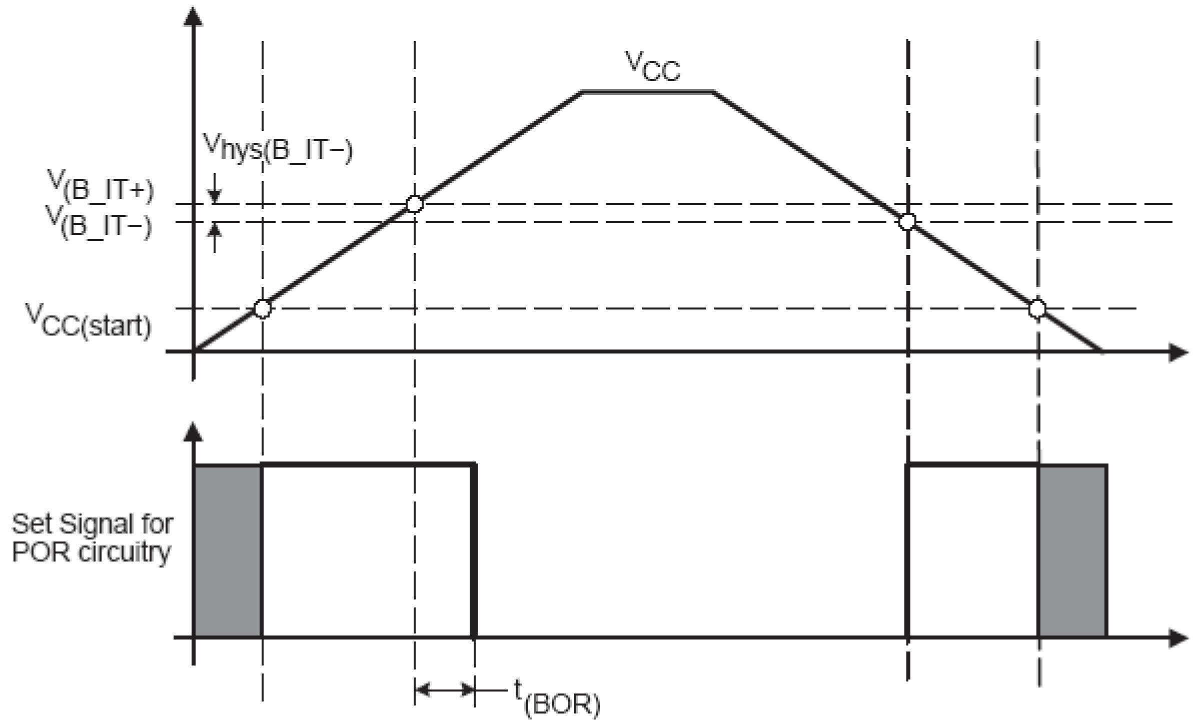
Power-On Reset (POR)

Figure 2−2. POR Timing



Brownout Reset

Figure 2−3. Brownout Timing



Device conditions after system reset

- The RST/NMI pin is configured in the reset mode
- I/O pins are switched to input mode as described in the Digital I/O chapter
- Other peripheral modules and registers are initialized as described in their respective chapters in this manual
- Status register (SR) is reset
- The watchdog timer powers up active in watchdog mode
- Program counter (PC) is loaded with address contained at reset vector location (0FFFEh). CPU execution begins at that address

Software initialization

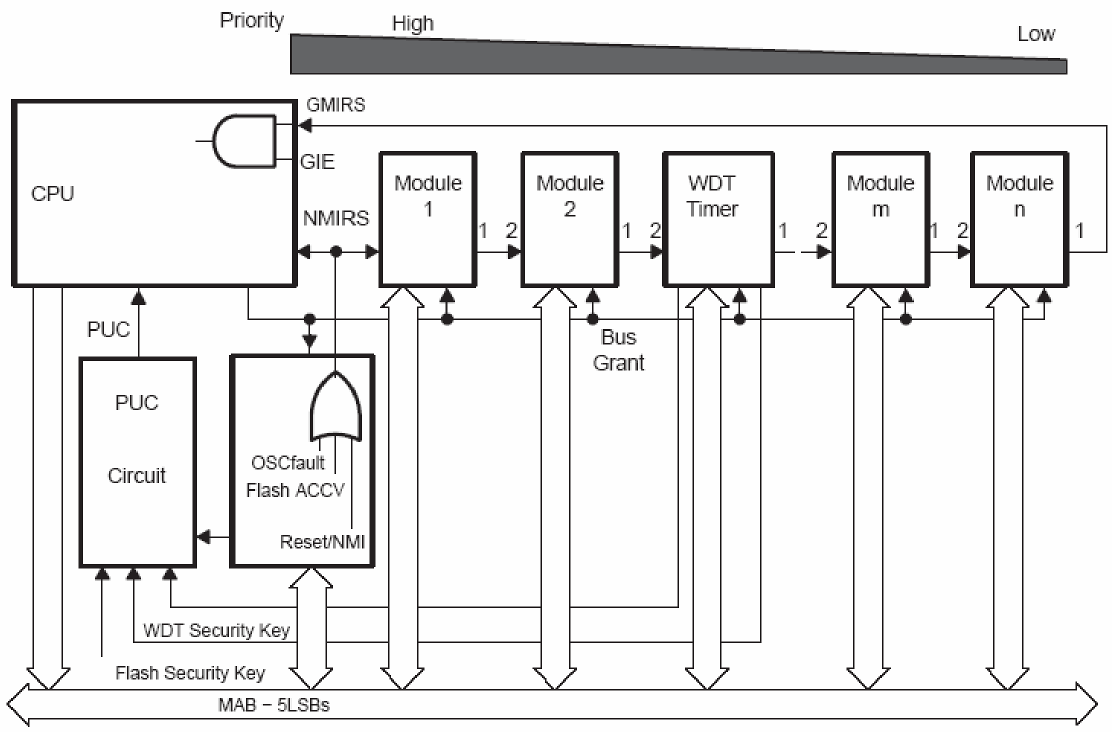
Your SW must initialize the MSP430

- Initialize the SP, typically to the top of RAM
- Initialize the watchdog to the requirements of the application
- Configure peripheral modules to the requirements of the application
- Additionally, the watchdog timer, oscillator fault, and flash memory flags can be evaluated to determine the source of the reset

Interrupts

- 3 types
  - System reset
  - (Non)-maskable NMI
  - Maskable
- Interrupt priorities are fixed and defined by the arrangement of modules

Figure 2−4. Interrupt Priority

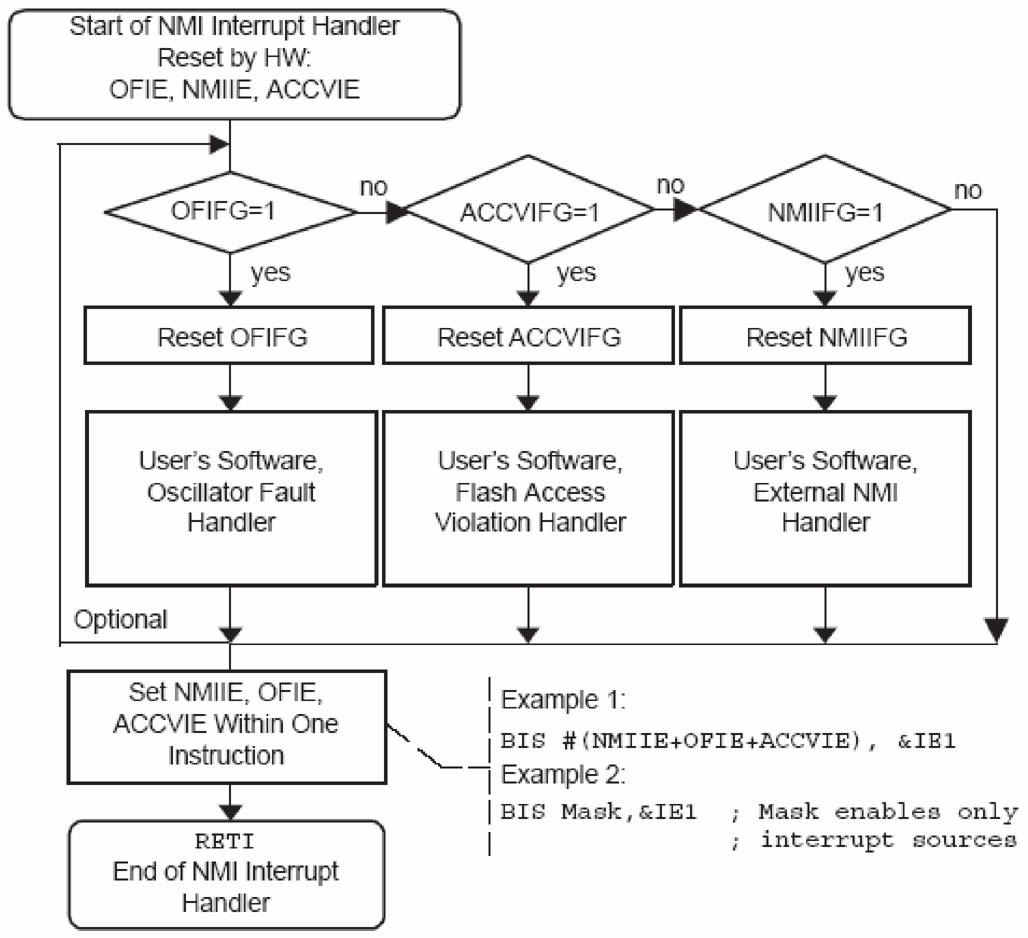


(Non)-Maskable Interrupts (NMI)

- Sources
  - An edge on the RST/NMI pin when configured in NMI mode
  - An oscillator fault occurs
  - An access violation to the flash memory
- Are not masked by GIE (General Interrupt Enable), but are enabled by individual interrupt enable bits (NMIIE, OFIE, ACCVIE)

NMI Interrupt Handler

Figure 2−6. NMI Interrupt Handler


Start of NMI Interrupt Handler
Reset by HW:
OFIE, NMIIE, ACCVIE
OFIFG=1
no
ACCVIFG=1
no
NMIIFG=1
no
yes
yes
yes
Reset OFIFG
Reset ACCVIFG
Reset NMIIFG
User's Software, Oscillator Fault Handler
User's Software, Flash Access Violation Handler
User's Software, External NMI Handler
Optional
Set NMIIE, OFIE, ACCVIE Within One Instruction
RETI
End of NMI Interrupt Handler

Example 1:

```
BIS #(NMIIE+OFIE+ACCVIE), &IE1
```

Example 2:

```
BIS Mask,&IE1   ; Mask enables only
                ; interrupt sources
```

Maskable Interrupts

- Caused by peripherals with interrupt capability
- Each can be disabled individually by an interrupt enable bit
- All can be disabled by GIE bit in the status register

Interrupt acceptance

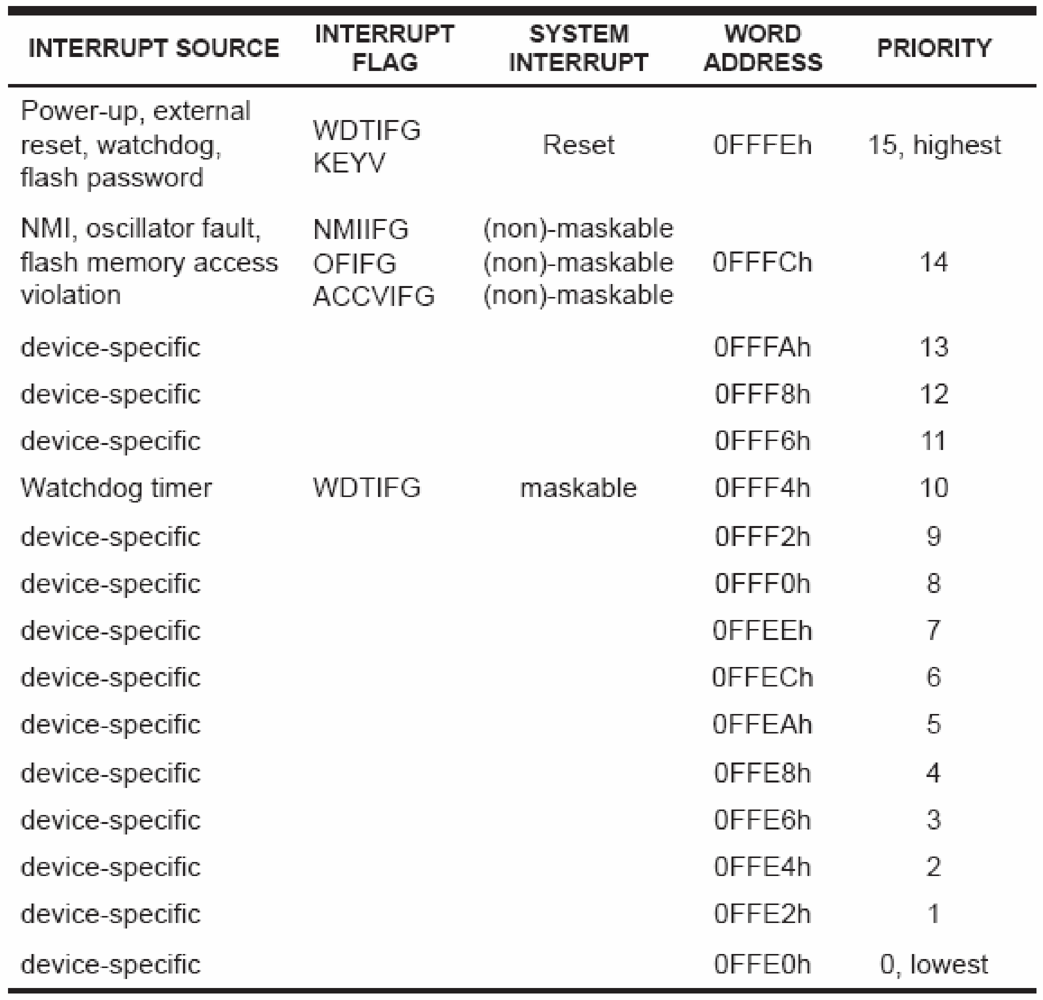
- 1) Any currently executing instruction is completed.
- 2) The PC, which points to the next instruction, is pushed onto the stack.
- 3) The SR is pushed onto the stack.
- 4) The interrupt with the highest priority is selected if multiple interrupts occurred during the last instruction and are pending for service.
- 5) The interrupt request flag resets automatically on single-source flags. Multiple source flags remain set for servicing by software.
- 6) The SR is cleared with the exception of SCG0, which is left unchanged. This terminates any low-power mode. Because the GIE bit is cleared, further interrupts are disabled.
- 7) The content of the interrupt vector is loaded into the PC: the program continues with the interrupt service routine at that address.
- *Takes 6 cc to execute*

Return from Interrupt

- RETI - Return from Interrupt Service Routine
  - 1) The SR with all previous settings pops from the stack. All previous settings of GIE, CPUOFF, etc. are now in effect, regardless of the settings used during the interrupt service routine.
  - 2) The PC pops from the stack and begins execution at the point where it was interrupted.
- *Takes 5 cc to execute*

Interrupt Vectors

Table 2−1. Interrupt Sources,Flags, and Vectors



INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-up, external reset, watchdog, flash password	WDTIFG KEYV	Reset	0FFFEh	15, highest
NMI, oscillator fault, flash memory access violation	NMIIFG OFIFG ACCVIFG	(non)-maskable (non)-maskable (non)-maskable	0FFFCh	14
device-specific			0FFFAh	13
device-specific			0FFF8h	12
device-specific			0FFF6h	11
Watchdog timer	WDTIFG	maskable	0FFF4h	10
device-specific			0FFF2h	9
device-specific			0FFF0h	8
device-specific			0FFEEh	7
device-specific			0FFECh	6
device-specific			0FFEAh	5
device-specific			0FFE8h	4
device-specific			0FFE6h	3
device-specific			0FFE4h	2
device-specific			0FFE2h	1
device-specific			0FFE0h	0, lowest

Interrupt Service Routines

- **Interrupt Service Routine declaration**

```
// Func. declaration
Interrupt[int_vector] void myISR (Void);

Interrupt[int_vector] void myISR (Void)
{
//  ISR code
}
```

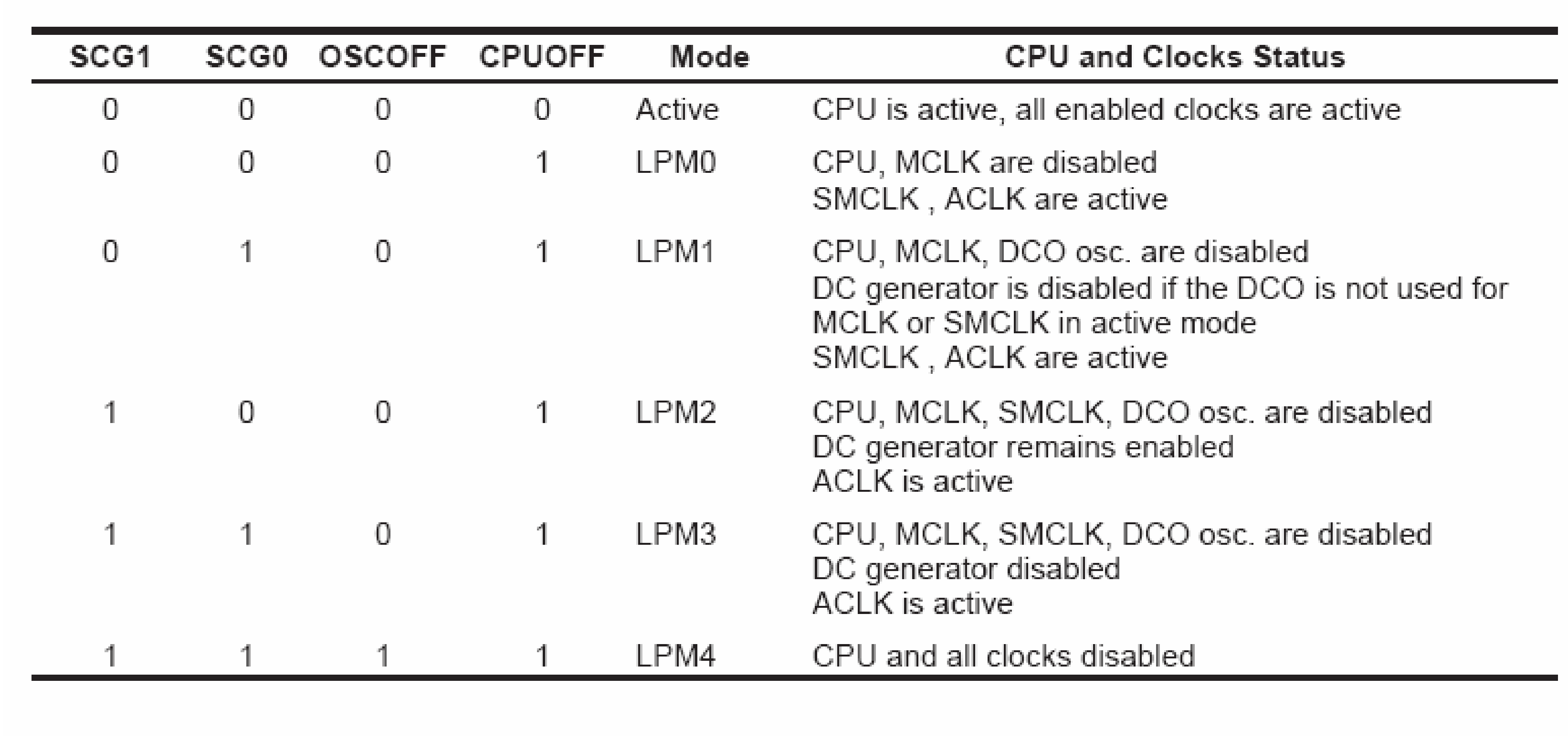
- **EXAMPLE**

```
Interrupt[TIMERA0_VECTOR] void myISR (Void);

Interrupt[TIMERA0_VECTOR] void myISR (Void)
{
//  ISR code
}
```

Interrupt Vectors

```
/************************************************************
* Interrupt Vectors (offset from 0xFFE0)
************************************************************/

#define PORT2_VECTOR        1 * 2  /* 0xFFE2 Port 2 */
#define UART1TX_VECTOR      2 * 2  /* 0xFFE4 UART 1 Transmit */
#define UART1RX_VECTOR      3 * 2  /* 0xFFE6 UART 1 Receive */
#define PORT1_VECTOR        4 * 2  /* 0xFFE8 Port 1 */
#define TIMERA1_VECTOR      5 * 2  /* 0xFFEA Timer A CC1-2, TA */
#define TIMERA0_VECTOR      6 * 2  /* 0xFFEC Timer A CC0 */
#define ADC_VECTOR          7 * 2  /* 0xFFEE ADC */
#define UART0TX_VECTOR      8 * 2  /* 0xFFF0 UART 0 Transmit */
#define UART0RX_VECTOR      9 * 2  /* 0xFFF2 UART 0 Receive */
#define WDT_VECTOR          10 * 2 /* 0xFFF4 Watchdog Timer */
#define COMPARATORA_VECTOR  11 * 2 /* 0xFFF6 Comparator A */
#define TIMERB1_VECTOR      12 * 2 /* 0xFFF8 Timer B 1-7 */
#define TIMERB0_VECTOR      13 * 2 /* 0xFFFA Timer B 0 */
#define NMI_VECTOR          14 * 2 /* 0xFFFC Non-maskable */
#define RESET_VECTOR        15 * 2 /* 0xFFFE Reset [Highest Pr.] */
```

Operating Modes (to be discussed later)



SCG1	SCG0	OSCOFF	CPUOFF	Mode	CPU and Clocks Status
0	0	0	0	Active	CPU is active, all enabled clocks are active
0	0	0	1	LPM0	CPU, MCLK are disabled SMCLK , ACLK are active
0	1	0	1	LPM1	CPU, MCLK, DCO osc. are disabled DC generator is disabled if the DCO is not used for MCLK or SMCLK in active mode SMCLK , ACLK are active
1	0	0	1	LPM2	CPU, MCLK, SMCLK, DCO osc. are disabled DC generator remains enabled ACLK is active
1	1	0	1	LPM3	CPU, MCLK, SMCLK, DCO osc. are disabled DC generator disabled ACLK is active
1	1	1	1	LPM4	CPU and all clocks disabled

MSP430: Basic Clock System

Basic Clock System

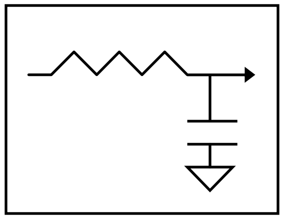
MSP430 Clock System

- **Low System Cost**
- **Low Power**

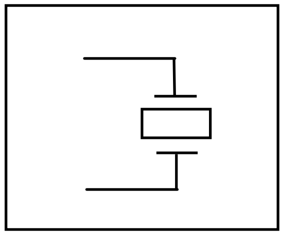
- Variety of operating modes driven by application, software selectable
- Support for the *Burst Mode* –
  when activated system starts and reacts rapidly
- Stability over voltage and temperature

Basic Clock System: MSP430x1xx

- One DCO, internal digitally controlled oscillator
  - Generated on-chip RC-type frequency controlled by SW + HW



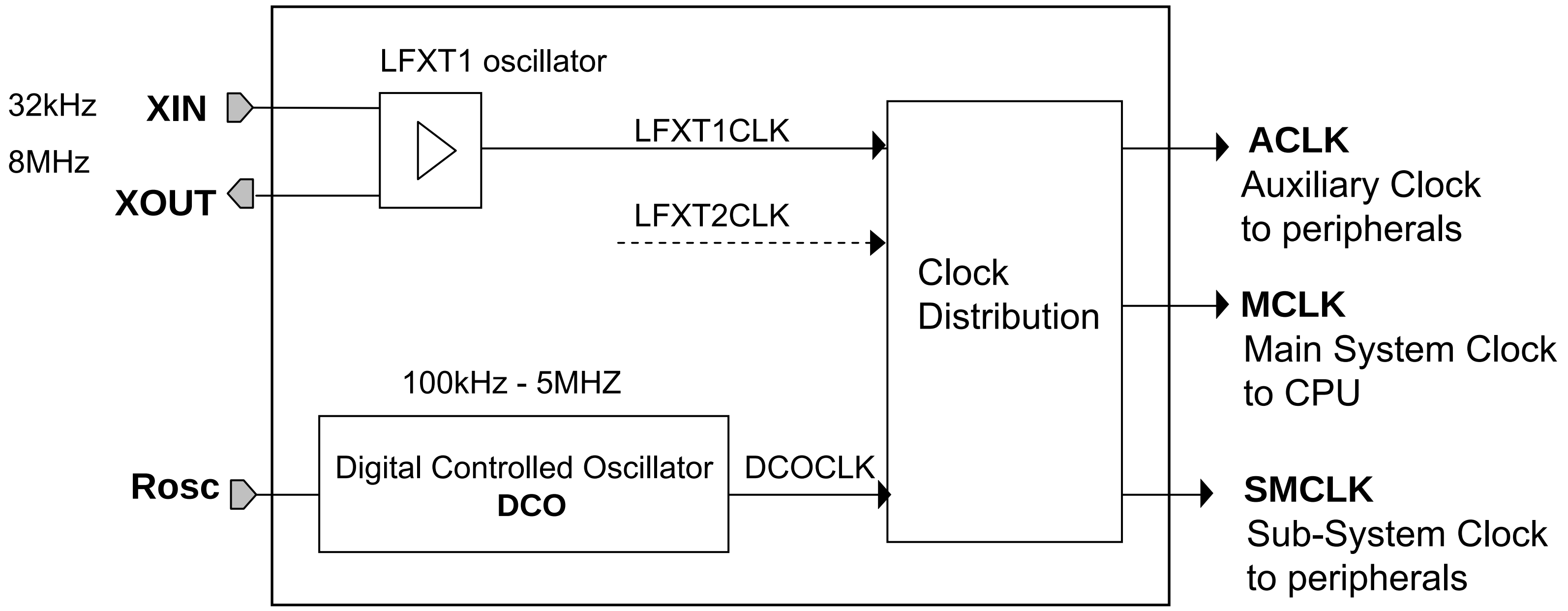
- One LF/XT oscillator
  - LF: 32768Hz
  - XT: 450kHz 8MHz



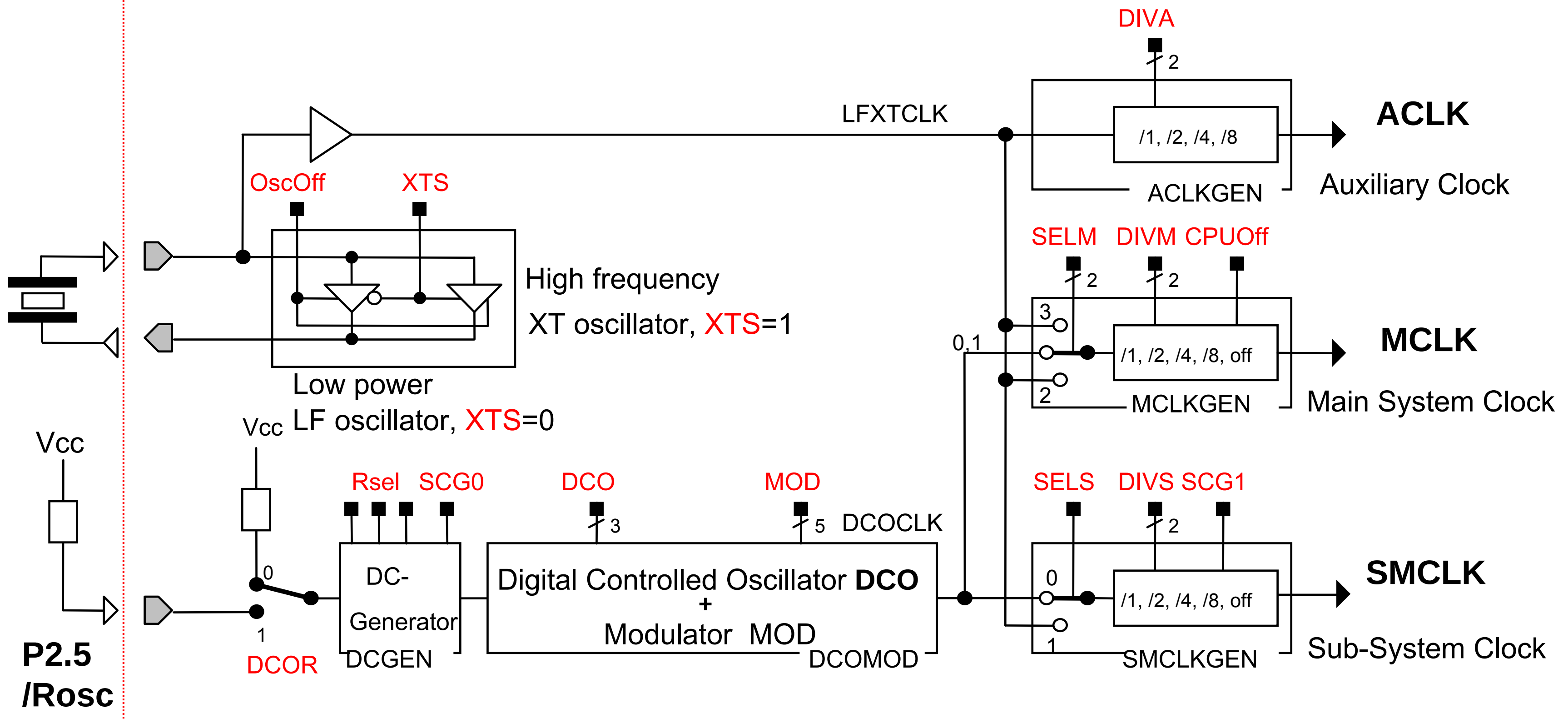
- Second LF/XT2 oscillator
  Optional XT: 450kHz 8MHz
- Clocks:
  - ACLK auxiliary clock ACLK
  - MCLK main system clock MCLK
  - SMCLK sub main system clock

Basic Clock System: MSP430x1xx

- DCOCLK Generated on-chip with 6μs start-up
- 32KHz Watch Crystal - or - High Speed Crystal / Resonator to 8MHz
  - (our system is 4MHz/8MHz high Speed Crystal)
- Flexible clock distribution tree for CPU and peripherals
- Programmable open-loop DCO Clock with internal and external current source



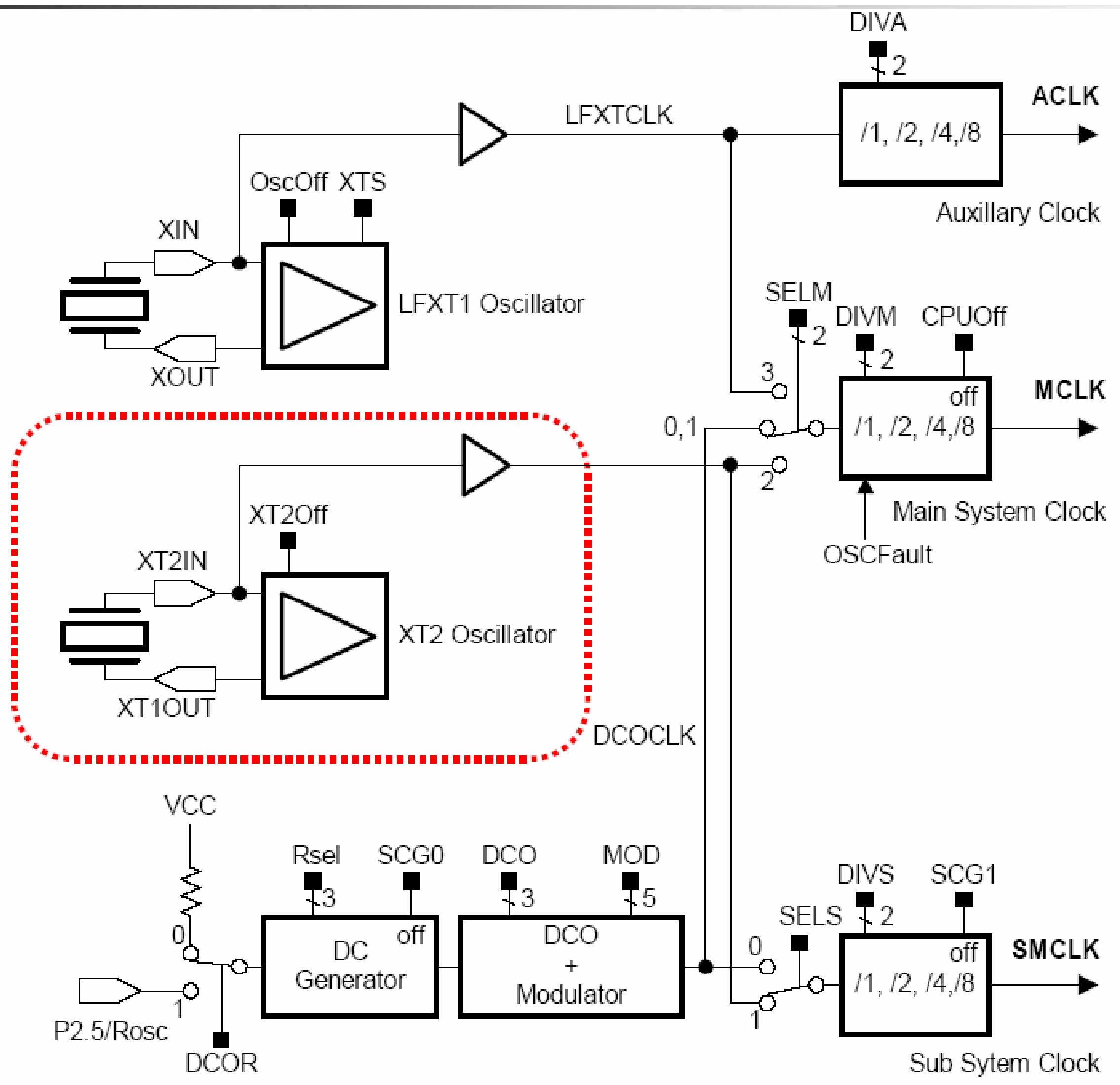
Basic Clock System – Block Diagram



The DCO-Generator is connected to pin P2.5/Rosc if DCOR control bit is set.

The port pin P2.5/Rosc is selected if DCOR control bit is reset (initial state).

Basic clock block diagram (MSP430x13x/14x/15x/16x)



Basic operation

- After POC (Power Up Clear)
  MCLK and SMCLK are sourced by DCOCLK (approx. 800KHz) and ACLK is sourced by LFXT1 in LF mode
- Status register control bits SCG0, SCG1, OSCOFF, and CPUOFF configure the MSP430 operating modes and enable or disable portions of the basic clock module
  - SCG1 - when set, turns off the SMCLK
  - SCG0 - when set, turns off the DCO dc generator
    (if DCOCLK is not used for MCLK or SMCLK)
  - OSCOFF - when set, turns off the LFXT1 crystal oscillator
    (if LFXT1CLK is not use for MCLK or SMCLK)
  - CPUOFF - when set, turns off the CPU
- DCOCTL, BCSCTL1, and BCSCTL2 registers
  configure the basic clock module
- The basic clock can be configured or reconfigured by software at any time during program execution

Basic Clock Module - Control Registers

The Basic Clock Module is configured using control registers DCOCTL, BCSCTL1, and BCSCTL2, and four bits from the CPU status register: SCG1, SCG0, OscOff, and CPUOFF.

User software can modify these control registers from their default condition at any time. The Basic Clock Module control registers are located in the byte-wide peripheral map and should be accessed with byte (.B) instructions.

Register State	**Short Form**	**Register Type**	**Address**	**Initial State**
DCO control register	DCOCTL	Read/write	056h	060h
Basic clock system control 1	BCSCTL1	Read/write	057h	084h
Basic clock system control 2	BCSCTL2	Read/write	058h	reset

Basic Clock Module - Control Registers

- Direct SW Control
- DCOCLK can be Set - Stabilized
- Stable DCOCLK over Temp/Vcc.

BCSCTL2

058h

SELM.1	SELM.0	DIVM.1	DIVM.0	SELS	DIVS.1	DIVS.0	DCOR
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

BCSCTL1

057h

XT2Off	XTS	DIVA.1	DIVA.0	XT5V	Rsel.2	Rsel.1	Rsel.0
rw-(1)	rw-(0)	rw-(0)	rw-(0)	rw-0	rw-1	rw-0	rw-0

Rsel.2–Rsel.0: **Selection of DCO nominal frequency**

DCOCTL

056h

DCO.2	DCO.1	DCO.0	MOD.4	MOD.3	MOD.2	MOD.1	MOD.0
rw-0	rw-1	rw-1	rw-0	rw-0	rw-0	rw-0	rw-0

DCO.2–DCO.0: Which of eight discrete DCO frequencies is selected

MOD.4–MOD.0: Define how often frequency f_{DCO+1} within the period of 32 DCOCLK cycles is used. Remaining clock cycles (32-MOD) the frequency f_{DCO} is mixed

RSEL.x Select DCO nominal frequency

DCO.x   and MOD.x set exact DCOCLK

…   select other clock tree options

DCOCTL

- **Digitally-Controlled Oscillator (DCO) Clock-Frequency Control**

DCOCTL is loaded with a value of 060h with a valid PUC condition.

	7							0
DCOCTL	DCO.2	DCO.1	DCO.0	MOD.4	MOD.3	MOD.2	MOD.1	MOD.0
056H	0	1	1	0	0	0	0	0

MOD.0 .. MOD.4: The MOD constant defines how often the discrete frequency f_{DCO+1} is used within a period of 32 DCOCLK cycles.

During the remaining clock cycles (32–MOD) the discrete frequency f $_{DCO}$ is used. When the DCO constant is set to seven, no modulation is possible since the highest feasible frequency has then been selected.

DCO.0 .. DCO.2: The DCO constant defines which one of the eight discrete frequencies is selected. The frequency is defined by the current injected into the dc generator.

BCSCTL1

- **Oscillator and Clock Control Register**

  BCSCTL1 is affected by a valid PUC or POR condition.

	7							**0**
BCSCTL1	XT2Off	XTS	DIVA.1	DIVA.0	XT5V	Rsel.2	Rsel.1	Rsel.0
057h	1	0	0	0	0	1	0	0

Bit0 to Bit2: The internal resistor is selected in eight different steps.

Rsel.0 to Rsel.2 The value of the resistor defines the nominal frequency.

The lowest nominal frequency is selected by setting Rsel=0.

Bit3, XT5V: XT5V should always be reset.

Bit4 to Bit5: The selected source for ACLK is divided by:

DIVA = 0: 1

DIVA = 1: 2

DIVA = 2: 4

DIVA = 3: 8

BCSCTL1

Bit6, XTS: The LFXT1 oscillator operates with a low-frequency or with a high-frequency crystal:

XTS = 0: The low-frequency oscillator is selected.

XTS = 1: The high-frequency oscillator is selected.

The oscillator selection must meet the external crystal’s operating condition.

Bit7, XT2Off: The XT2 oscillator is switched on or off:

XT2Off = 0: the oscillator is on

XT2Off = 1: the oscillator is off if it is not used for MCLK or SMCLK.

BCSCTL2

BCSCTL2 is affected by a valid PUC or POR condition.

	7							0
BCSCTL2 058h	SELM.1	SELM.0	DIVM.1	DIVM.0	SELS	DIVS.1	DIVS.0	DCOR

Bit0, DCOR: The DCOR bit selects the resistor for injecting current into the dc generator. Based on this current, the oscillator operates if activated.

DCOR = 0: Internal resistor on, the oscillator can operate. The fail-safe mode is on.

DCOR = 1: Internal resistor off, the current must be injected externally if the DCO output drives any clock using the DCOCLK.

Bit1, Bit2: The selected source for SMCLK is divided by:

DIVS.1 .. DIVS.0 DIVS = 0:1

DIVS = 1: 2

DIVS = 2: 4

DIVS = 3: 8

BCSCTL2

Bit3, SELS: Selects the source for generating SMCLK:

SELS = 0: Use the DCOCLK

SELS = 1: Use the XT2CLK signal (in three-oscillator systems)

or

LFXT1CLK signal (in two-oscillator systems)

Bit4, Bit5: The selected source for MCLK is divided by DIVM.0 .. DIVM.1

DIVM = 0: 1

DIVM = 1: 2

DIVM = 2: 4

DIVM = 3: 8

Bit6, Bit7: Selects the source for generating MCLK:

SELM.0 .. SELM.1

SELM = 0: Use the DCOCLK

SELM = 1: Use the DCOCLK

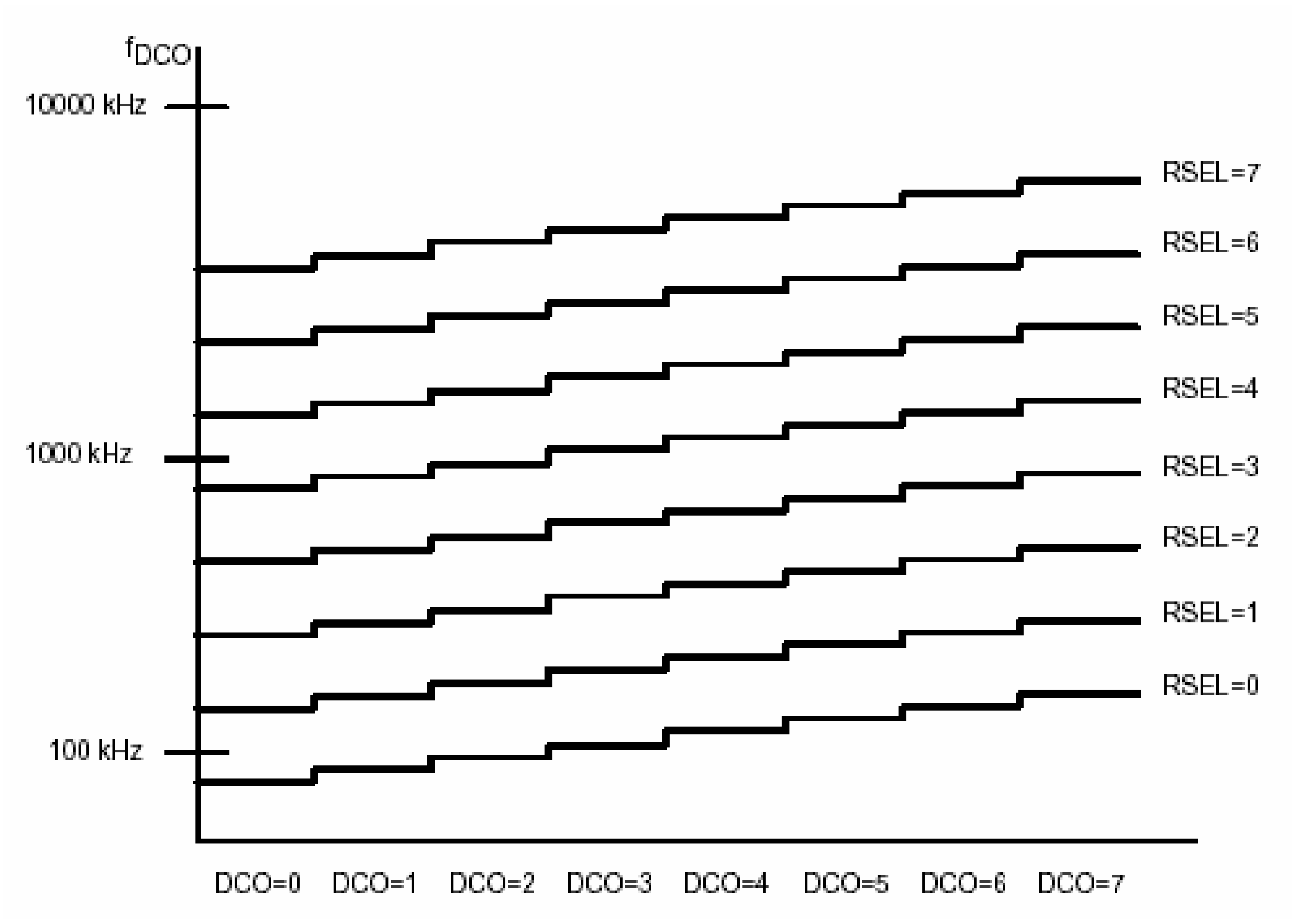
SELM = 2: Use the XT2CLK (x13x and x14x devices)

or

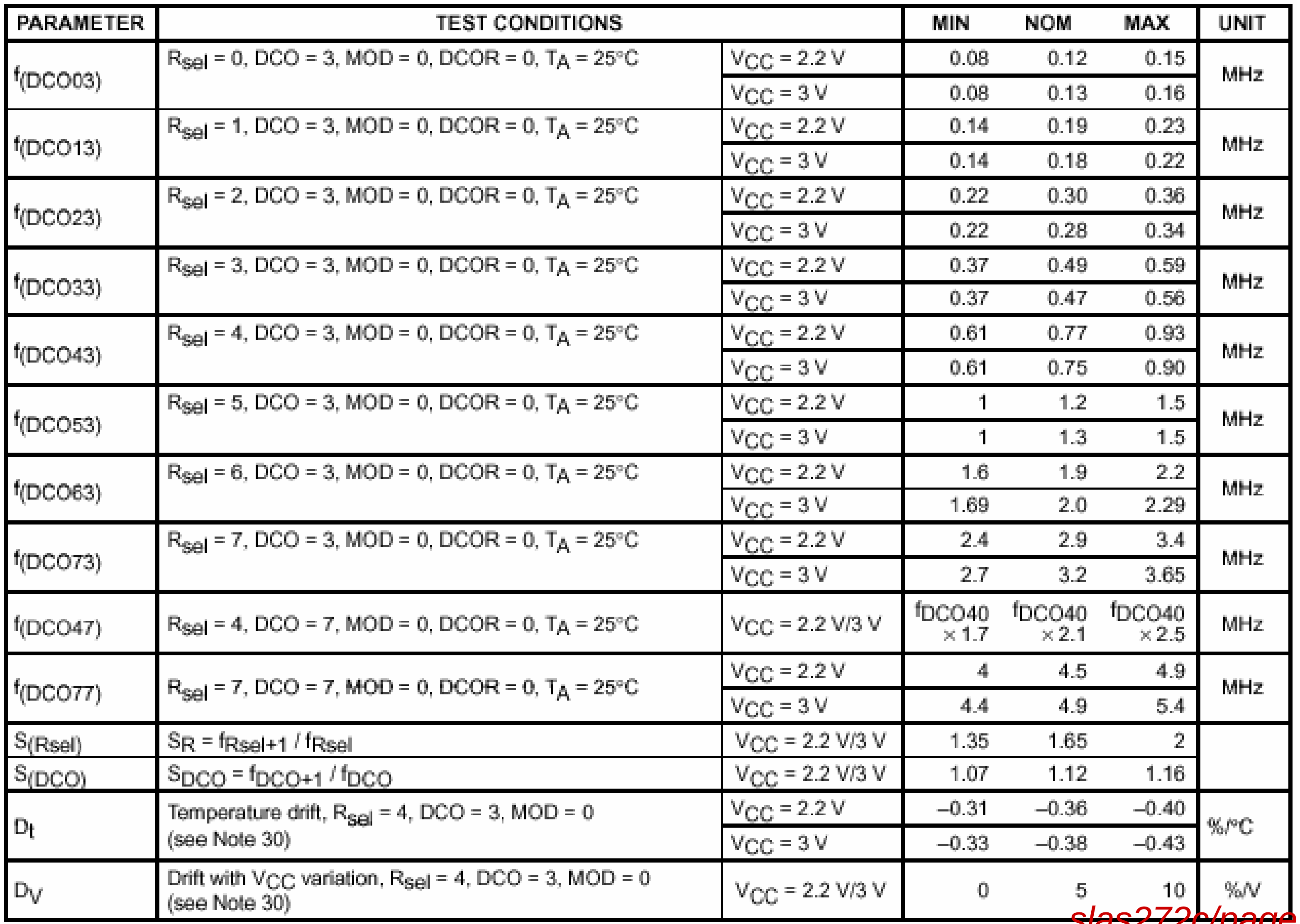
Use the LFXT1CLK (x11x(1) devices)

SELM = 3: Use the LFXT1CLK

Range (RSELx) and Steps (DCOx)



F149 default DCO clock setting

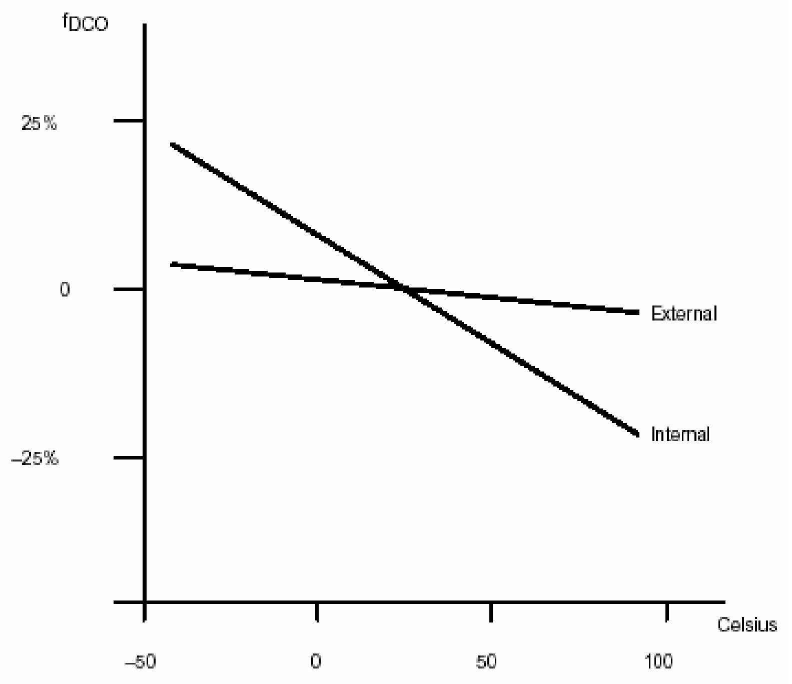


PARAMETER	TEST CONDITIONS		MIN	NOM	MAX	UNIT
$f_{(DCO03)}$	R_{sel} = 0, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	0.08	0.12	0.15	MHz
		V_{CC} = 3 V	0.08	0.13	0.16	
$f_{(DCO13)}$	R_{sel} = 1, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	0.14	0.19	0.23	MHz
		V_{CC} = 3 V	0.14	0.18	0.22	
$f_{(DCO23)}$	R_{sel} = 2, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	0.22	0.30	0.36	MHz
		V_{CC} = 3 V	0.22	0.28	0.34	
$f_{(DCO33)}$	R_{sel} = 3, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	0.37	0.49	0.59	MHz
		V_{CC} = 3 V	0.37	0.47	0.56	
$f_{(DCO43)}$	R_{sel} = 4, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	0.61	0.77	0.93	MHz
		V_{CC} = 3 V	0.61	0.75	0.90	
$f_{(DCO53)}$	R_{sel} = 5, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	1	1.2	1.5	MHz
		V_{CC} = 3 V	1	1.3	1.5	
$f_{(DCO63)}$	R_{sel} = 6, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	1.6	1.9	2.2	MHz
		V_{CC} = 3 V	1.69	2.0	2.29	
$f_{(DCO73)}$	R_{sel} = 7, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	2.4	2.9	3.4	MHz
		V_{CC} = 3 V	2.7	3.2	3.65	
$f_{(DCO47)}$	R_{sel} = 4, DCO = 7, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V/3 V	$f_{DCO40} \times 1.7$	$f_{DCO40} \times 2.1$	$f_{DCO40} \times 2.5$	MHz
$f_{(DCO77)}$	R_{sel} = 7, DCO = 7, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	4	4.5	4.9	MHz
		V_{CC} = 3 V	4.4	4.9	5.4	
$S_{(Rsel)}$	$S_R = f_{Rsel+1} / f_{Rsel}$	V_{CC} = 2.2 V/3 V	1.35	1.65	2	
$S_{(DCO)}$	$S_{DCO} = f_{DCO+1} / f_{DCO}$	V_{CC} = 2.2 V/3 V	1.07	1.12	1.16	
D_t	Temperature drift, R_{sel} = 4, DCO = 3, MOD = 0 (see Note 30)	V_{CC} = 2.2 V	–0.31	–0.36	–0.40	%/°C
		V_{CC} = 3 V	–0.33	–0.38	–0.43	
D_V	Drift with V_{CC} variation, R_{sel} = 4, DCO = 3, MOD = 0 (see Note 30)	V_{CC} = 2.2 V/3 V	0	5	10	%/V

slas272c/page 46

External Resistor

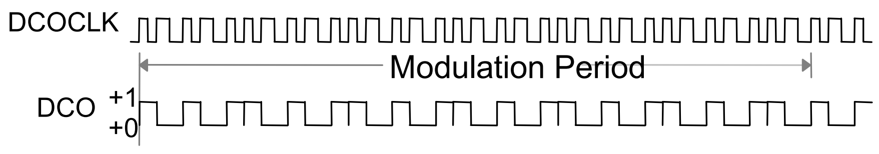
- The DCO temperature coefficient can be reduced by using an external resistor ROSC to source the current for the DC generator.
- ROSC also allows the DCO to operate at higher frequencies.
  - Internal resistor nominal value is approximately 200 kOhm => DCO to operate up to 5 MHz.
  - External ROSC of approximately 100 kOhm => the DCO can operate up to approximately 10 MHz.

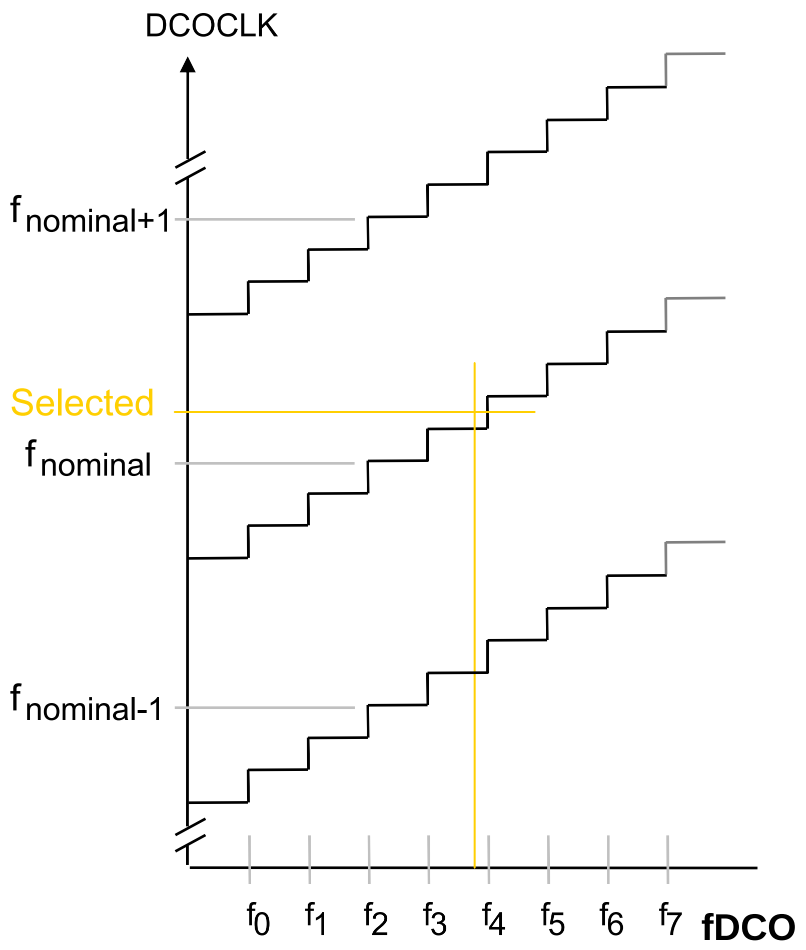


Basic Clock Systems-DCO TAPS

- **DCOCLK frequency control**
- nominal - injected current into DC generator
  1) internal resistors Rsel2, Rsel1 and Rsel0
  2) an external resistor at Rosc (P2.5/11x)
- Control bits DCO0 to DCO2 set fDCO tap
- Modulation bits MOD0 to MOD4 allow mixing of fDCO and fDCO+1 for precise frequency generation

Example Selected:	Frequency	Cycle time
	1000kHz	1000 nsec
f3:	943kHz	1060 nsec
f4:	1042kHz	960 nsec
MOD=19		

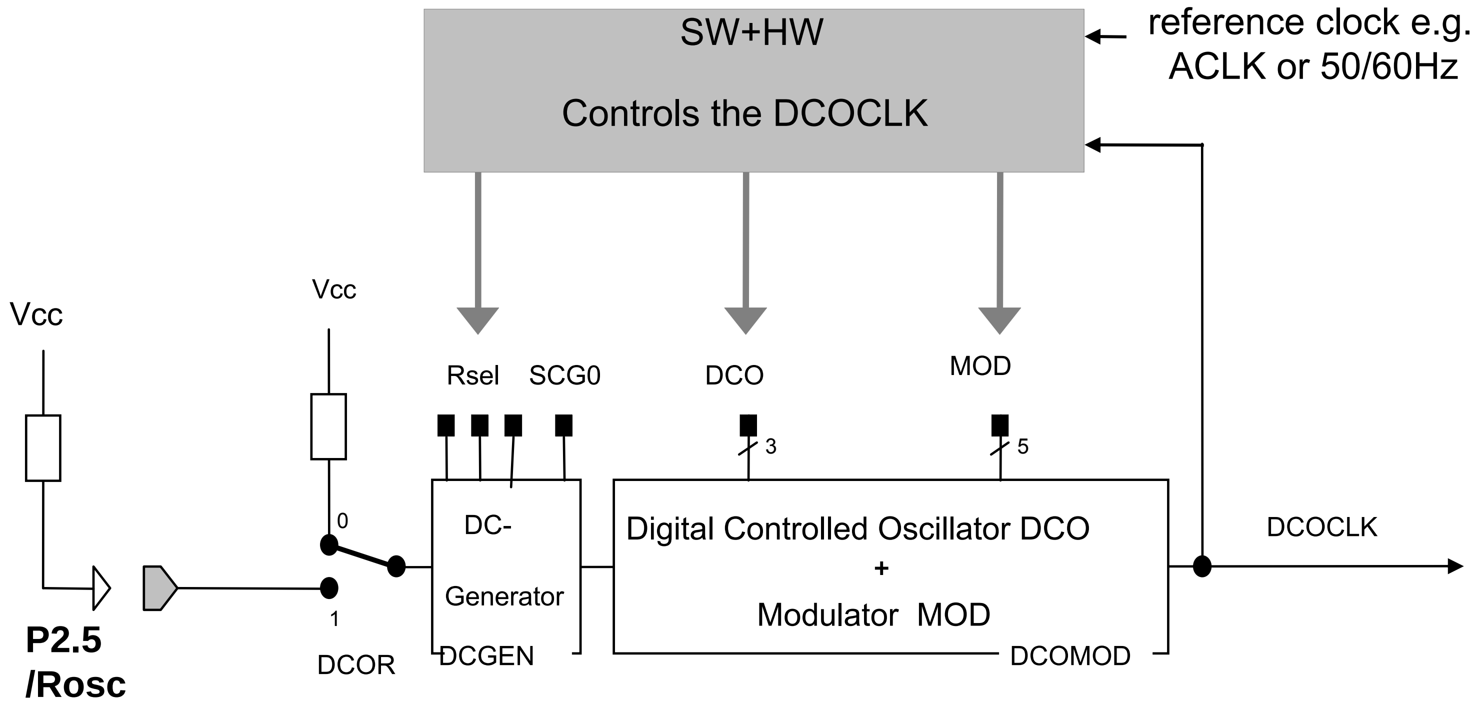




To produce an intermediate effective frequency between f_{DCO} and f_{DCO+1}
Cycle_time = $((32\text{-MOD})*t_{DCO}+\text{MOD}*t_{DCO+1})/32$ = 1000.625 ns, selected frequency ≈ 1 MHz.

Software FLL

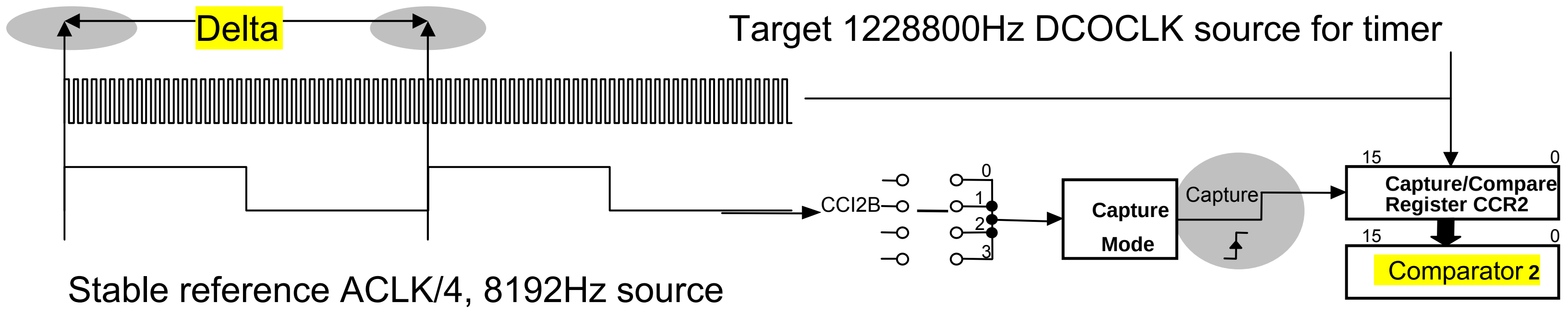
- **Basic Clock DCO is an open loop - close with SW+HW**
  - A reference frequency e.g. ACLK or 50/60Hz can be used to measure DCOCLK’s
  - **Initialization or Periodic** software set and stabilizes DCOCLK over reference clock
  - DCOCLK is programmable 100kHz - 5Mhz and stable over voltage and temperature



Software FLL Implementation

- **Example: Set DCOCLK= 1228800, ACLK= 32768**
  - ACLK/4 captured on CCI2B, DCOCLK is clock source for Timer_A
  - Comparator2 HW captures SMCLK (1228800Hz) in one ACLK/4 (8192Hz) period
  - Target Delta = 1228800/8192= 150

```
CCI2BInt …                        ; Compute Delta
        cmp     #150,Delta        ; Delta= 1228800/8192
        jlo     IncDCO            ; JMP to IncDCO
DecDCO  dec     &DCOCTL           ; Decrease DCOCLK
        reti
IncDCO  inc     &DCOCTL           ; Increase DCOCLK
        reti
```



Delta
Target 1228800Hz DCOCLK source for timer
CCI2B
0
1
2
3
Capture Mode
Capture
15
0
Capture/Compare Register CCR2
15
0
Comparator 2
Stable reference ACLK/4, 8192Hz source

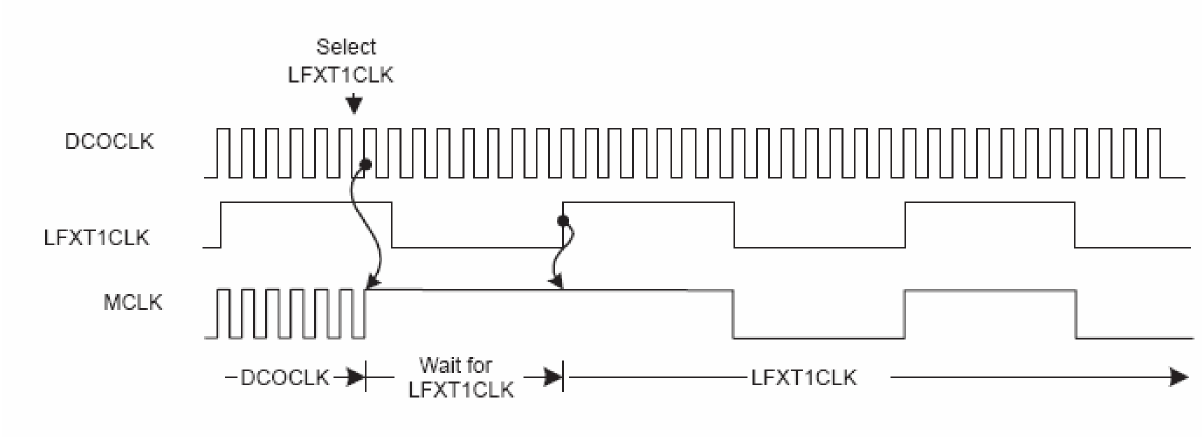
Fail Safe Operation

- Basic module incorporates an oscillator-fault detection fail-safe feature.
- The oscillator fault detector is an analog circuit that monitors the LFXT1CLK (in HF mode) and the XT2CLK.
- An oscillator fault is detected when either clock signal is not present for approximately 50 us.
  - When an oscillator fault is detected, and when MCLK is sourced from either LFXT1 in HF mode or XT2, MCLK is automatically switched to the DCO for its clock source.
- When OFIFG is set and OFIE is set, an NMI interrupt is requested. The NMI interrupt service routine can test the OFIFG flag to determine if an oscillator fault occurred. The OFIFG flag must be cleared by software.

Synchronization of clock signals

- When switching MCLK and SMCLK from one clock source to another
  => avoid race conditions
  - The current clock cycle continues until the next rising edge
  - The clock remains high until the next rising edge of
  - ues with

Figure 4−11. Switch MCLK from DCOCLK to LFXT1CLK



Basic Clock Module - Examples

- **How to select the Crystal Clock**

```
BCSCTL1 |= XTS;                                      // ACLK = LFXT1 = HF XTAL
BCSCTL1 |= DIVA0;                                    // ACLK = XT1 / 8
BCSCTL1 |= DIVA1;
do {
    IFG1 &= ~OFIFG;                                    // Clear OSCFault flag from SW
    for (i = 0xFF; i > 0; i--);                        // Time for flag to set by HW
 } while ((IFG1 & OFIFG));                             // OSCFault flag still set?
// clock is stable
BCSCTL2 |= SELM_3;                                   // MCLK = LFXT1 (safe)
```

Basic Clock Systems-Examples

- **Adjusting the Basic Clock**

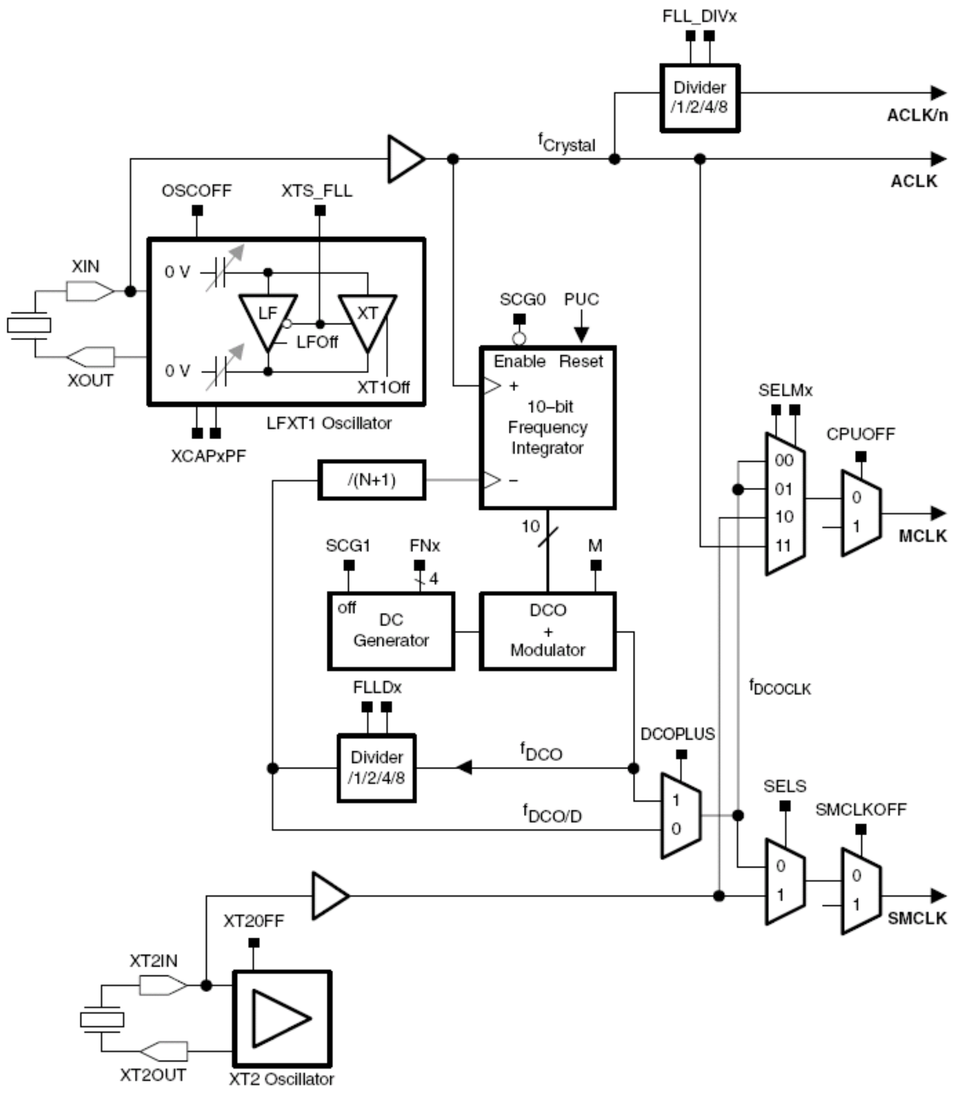
The control registers of the Basic Clock are under full software control. If clock requirements other than those of the default from PUC are necessary, the Basic Clock can be configured or reconfigured by software at any time during program execution.

- ACLKGEN from LFXT1 crystal, resonator, or external-clock source and divided by 1, 2, 4, or 8. If no LFXTCLK clock signal is needed in the application, the OscOff bit should be set in the status register.
- SCLKGEN from LFXTCLK, DCOCLK, or XT2CLK (x13x and x14x only) and divided by 1, 2, 4, or 8. The SCG1 bit in the status register enables or disables SMCLK.
- MCLKGEN from LFXTCLK, DCOCLK, or XT2CLK (x13x and x14x only) and divided by 1, 2, 4, or 8. When set, the CPUOff bit in the status register enables or disables MCLK.
- DCOCLK frequency is adjusted using the RSEL, DCO, and MOD bits. The DCOCLK clock source is stopped when not used, and the dc generator can be disabled by the SCG0 bit in the status register (when set).
- The XT2 oscillator sources XT2CLK (x13x and x14x only) by clearing the XT2Off bit.

FLL+ Clock Module (MSP430x4xx)

- FLL+ clock module:
  frequency-locked loop clock module
  - Low system cost
  - Ultra-low power consumption
  - Can operate with no external components
  - Supports one or two external crystals or resonators (LFXT1 and XT2)
  - Internal digitally-controlled oscillator with stabilization to a multiple of the LFXT1 watch crystal frequency
  - Full software control over 4 output clocks: ACLK, ACLK/n, MCLK, and SMCLK

MSP430x43x, MSP430x44x and MSP430x461x Frequency-Locked Loop



FLL+ Clock Module

- LFXT1CLK: Low-frequency/high-frequency oscillator that can be used
  - either with low-frequency 32768-Hz watch crystals, or
  - standard crystals or resonators in the 450-kHz to 8-MHz range.
- XT2CLK: Optional high-frequency oscillator that can be used with standard crystals, resonators, or external clock sources in the 450-kHz to 8-MHz range. In MSP430F47x devices the upper limit is 16 MHz.
- DCOCLK: Internal digitally controlled oscillator (DCO) with RC-type characteristics, stabilized by the FLL.
- Four clock signals are available from the FLL+ module:
  - ACLK: Auxiliary clock. The ACLK is the LFXT1CLK clock source. ACLK is software selectable for individual peripheral modules.
  - ACLK/n: Buffered output of the ACLK. The ACLK/n is ACLK divided by 1,2,4 or 8 and only used externally.
  - MCLK: Master clock. MCLK is software selectable as LFXT1CLK, XT2CLK (if available), or DCOCLK. MCLK can be divided by 1, 2, 4, or 8 within the FLL block. MCLK is used by the CPU and system.
  - SMCLK: Sub-main clock. SMCLK is software selectable as XT2CLK (if available), or DCOCLK. SMCLK is software selectable for individual peripheral modules.

FLL+ Clock Module Operation

- After a PUC, MCLK and SMCLK are sourced from DCOCLK at 32 times the ACLK frequency. When a 32,768-Hz crystal is used for ACLK, MCLK and SMCLK will stabilize to 1.048576 MHz.
- Status register control bits SCG0, SCG1, OSCOFF, and CPUOFF configure the MSP430 operating modes and enable or disable components of the FLL+ clock module.
- The SCFQCTL, SCFI0, SCFI1, FLL_CTL0, and FLL_CTL1 registers configure the FLL+ clock module. The FLL+ can be configured or reconfigured by software at any time during program execution.
- Example, MCLK = 64 × ACLK = 2097152

```
BIC #GIE,SR ; Disable interrupts
MOV.B #(64-1),&SCFQTL ; MCLK = 64 * ACLK, DCOPLUS=0
MOV.B #FN_2,&SCFIO ; Select DCO range
BIS #GIE,SR ; Enable interrupts
```

LFXT1 Oscillator

- Low-frequency (LF) mode (XTS_FLL=0) with 32,768 Hz watch crystal connected to XIN and XOUT
- High-frequency (HF) mode (XTS_FLL=1) with high-frequency crystals or resonators connected to XIN and XOUT (~450 KHz to 8 MHz)
- XCPxPF bits configure the internally provided load capacitance for the LFXT1 crystal (1, 6, 8, or 10 pF)
- OSCOFF bit can be set to disable LFXT1

XT2 Oscillator

- XT2 sources XT2CLK and its characteristics are identical to LFXT1 in HF mode, except it does not have internal load capacitors (must be provided externally)
- XT2OFF bit disables the XT2 oscillator if XT2CLK is not used for MCLK and SMCLK

DCO

- Integrated ring oscillator with RC-type characteristics
- DCO frequency is stabilized by the FLL to a multiple of ACLK as defined by N (the lowest 7 bits of the SCFQCTL register)
- DCOPLUS bit sets the f_{DCOCLK} to f_{DCO} or $f_{DCO/D}$ (divider). The FLLDx bits define the divider D to 1, 2, 4 or 8. By default DCOPLUS=0 and D=2, providing $f_{DCOCLK} = f_{DCO/2}$
- DCOPLUS = 0: $f_{DCOCLK} = (N + 1) \times f_{ACLK}$
- DCOPLUS = 1: $f_{DCOCLK} = D \times (N + 1) \times f_{ACLK}$

DCO Frequency Range

DCO

PARAMETER	TEST CONDITIONS	V_{CC}	MIN	TYP	MAX	UNIT
$f_{(DCOCLK)}$	$N_{(DCO)}$=01Eh, FN_8=FN_4=FN_3=FN_2=0, D = 2; DCOPLUS= 0	2.2 V/3 V		1		MHz
$f_{(DCO=2)}$	FN_8=FN_4=FN_3=FN_2=0 ; DCOPLUS = 1	2.2 V	0.3	0.65	1.25	MHz
		3 V	0.3	0.7	1.3	
$f_{(DCO=27)}$	FN_8=FN_4=FN_3=FN_2=0; DCOPLUS = 1	2.2 V	2.5	5.6	10.5	MHz
		3 V	2.7	6.1	11.3	
$f_{(DCO=2)}$	FN_8=FN_4=FN_3=0, FN_2=1; DCOPLUS = 1	2.2 V	0.7	1.3	2.3	MHz
		3 V	0.8	1.5	2.5	
$f_{(DCO=27)}$	FN_8=FN_4=FN_3=0, FN_2=1; DCOPLUS = 1	2.2 V	5.7	10.8	18	MHz
		3 V	6.5	12.1	20	
$f_{(DCO=2)}$	FN_8=FN_4=0, FN_3= 1, FN_2=x; DCOPLUS = 1	2.2 V	1.2	2	3	MHz
		3 V	1.3	2.2	3.5	
$f_{(DCO=27)}$	FN_8=FN_4=0, FN_3= 1, FN_2=x; DCOPLUS = 1	2.2 V	9	15.5	25	MHz
		3 V	10.3	17.9	28.5	
$f_{(DCO=2)}$	FN_8=0, FN_4= 1, FN_3= FN_2=x; DCOPLUS = 1	2.2 V	1.8	2.8	4.2	MHz
		3 V	2.1	3.4	5.2	
$f_{(DCO=27)}$	FN_8=0, FN_4=1, FN_3= FN_2=x; DCOPLUS = 1	2.2 V	13.5	21.5	33	MHz
		3 V	16	26.6	41	
$f_{(DCO=2)}$	FN_8=1, FN_4=FN_3=FN_2=x; DCOPLUS = 1	2.2 V	2.8	4.2	6.2	MHz
		3 V	4.2	6.3	9.2	
$f_{(DCO=27)}$	FN_8=1,FN_4=FN_3=FN_2=x; DCOPLUS = 1	2.2 V	21	32	46	MHz
		3 V	30	46	70	
S_n	Step size between adjacent DCO taps: $S_n = f_{DCO(Tap\ n+1)} / f_{DCO(Tap\ n)}$ (see Figure 16 for taps 21 to 27)	$1 < TAP \le 20$	1.06		1.11	
		TAP = 27	1.07		1.17	
D_t	Temperature drift, $N_{(DCO)}$ = 01Eh, FN_8=FN_4=FN_3=FN_2=0 D = 2; DCOPLUS = 0	2.2 V	–0.2	–0.3	–0.4	%/°C
		3 V	–0.2	–0.3	–0.4	
D_V	Drift with V_{CC} variation, $N_{(DCO)}$ = 01Eh, FN_8=FN_4=FN_3=FN_2=0 D = 2; DCOPLUS = 0		0	5	15	%/V

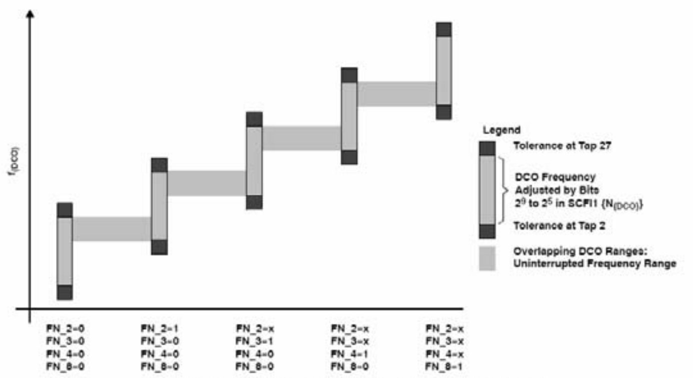

$f_{(DCO)}$
Legend
Tolerance at Tap 27
DCO Frequency Adjusted by Bits 2^9 to 2^5 in SCFI1 {$N_{(DCO)}$}
Tolerance at Tap 2
Overlapping DCO Ranges: Uninterrupted Frequency Range
FN_2=0 FN_3=0 FN_4=0 FN_8=0
FN_2=1 FN_3=0 FN_4=0 FN_8=0
FN_2=x FN_3=1 FN_4=0 FN_8=0
FN_2=x FN_3=x FN_4=1 FN_8=0
FN_2=x FN_3=x FN_4=x FN_8=1

Figure 17. Five Overlapping DCO Ranges Controlled by FN_x Bits

Frequency Locked Loop

- FLL continuously counts up or down a 10-bit frequency integrator.
- The output of the frequency integrator that drives the DCO can be read in SCFI1 and SCFI0. The count is adjusted +1 or −1 with each ACLK crystal period.
- Five of the integrator bits, SCFI1 bits 7-3, set the DCO frequency tap. Twenty-nine taps are implemented for the DCO (28, 29, 30, and 31 are equivalent), and each is approximately 10% higher than the previous. The modulator mixes two adjacent DCO frequencies to produce fractional taps.
- SCFI1 bits 2-0 and SCFI0 bits 1-0 are used for the modulator.
- The DCO starts at the lowest tap after a PUC or when SCFI0 and SCFI1 are cleared. Time must be allowed for the DCO to settle on the proper tap for normal operation. 32 ACLK cycles are required between taps requiring a worst case of 28 x 32 ACLK cycles for the DCO to settle

FLL+ Clock Module Registers

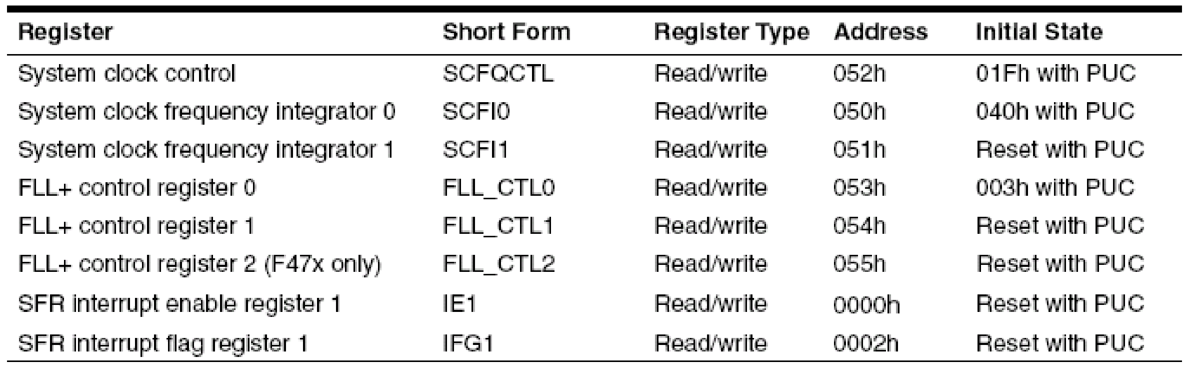


Table 5–2. FLL+ Registers

Register	Short Form	Register Type	Address	Initial State
System clock control	SCFQCTL	Read/write	052h	01Fh with PUC
System clock frequency integrator 0	SCFI0	Read/write	050h	040h with PUC
System clock frequency integrator 1	SCFI1	Read/write	051h	Reset with PUC
FLL+ control register 0	FLL_CTL0	Read/write	053h	003h with PUC
FLL+ control register 1	FLL_CTL1	Read/write	054h	Reset with PUC
FLL+ control register 2 (F47x only)	FLL_CTL2	Read/write	055h	Reset with PUC
SFR interrupt enable register 1	IE1	Read/write	0000h	Reset with PUC
SFR interrupt flag register 1	IFG1	Read/write	0002h	Reset with PUC

MSP430: Watchdog Timer

Watchdog Timer-General

General

The primary function of the watchdog-timer module (WDT) is to perform a controlled-system restart after a software problem occurs. If the selected time interval expires, a system reset is generated. If the watchdog function is not needed in an application, the module can work as an interval timer, to generate an interrupt after the selected time interval.

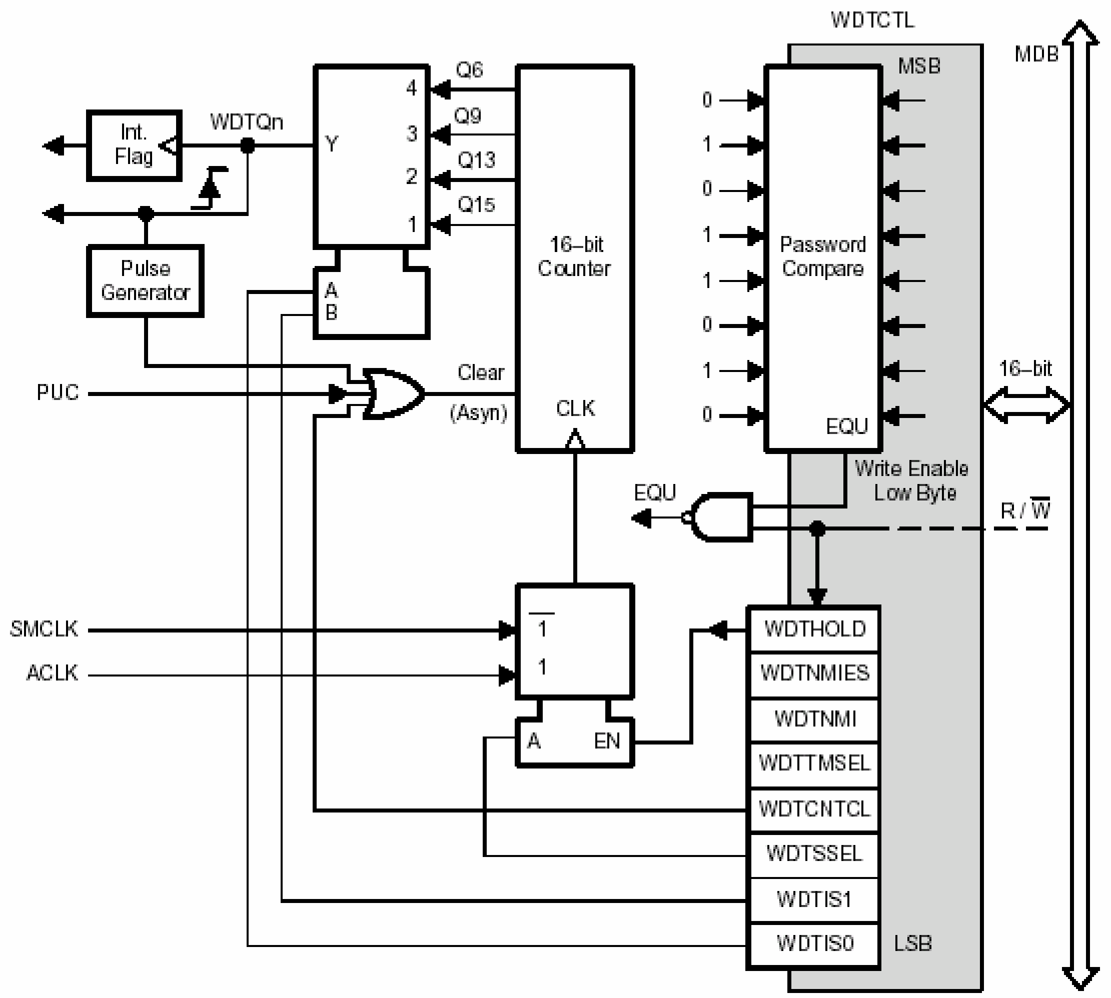
Features of the Watchdog Timer include:

- Eight software-selectable time intervals
- Two operating modes: as watchdog or interval timer
- Expiration of the time interval in watchdog mode, which generates a system reset; or in timer mode, which generates an interrupt request
- Safeguards which ensure that writing to the WDT control register is only possible using a password
- Support of ultralow-power using the hold mode

Watchdog/Timer two functions:

- SW Watchdog Mode
- Interval Timer Mode

Watchdog Timer-Diagram

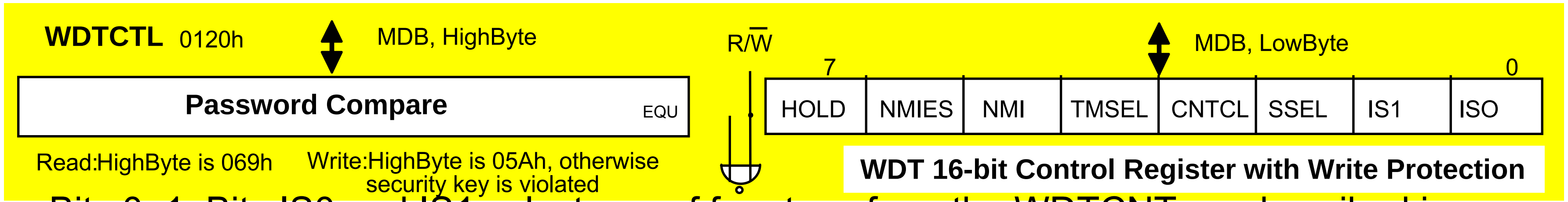


Watchdog Timer-Registers

- **Watchdog Timer Counter**

  The watchdog-timer counter (WDTCNT) is a 16-bit up-counter that 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 must be done using word instructions with no suffix or .w suffix. In both operating modes (watchdog or timer), it is only possible to write to WDTCTL using the correct password.

- **Watchdog Timer Control Register**



Bits 0, 1: Bits IS0 and IS1 select one of four taps from the WDTCNT, as described in following table. Assuming f crystal = 32,768 Hz and f System = 1 MHz, the following intervals are possible:

WDTCTL

Bits 0, 1: Bits IS0 and IS1 select one of four taps from the WDTCNT, as described in following table. Assuming f crystal = 32,768 Hz and f System = 1 MHz, the following intervals are possible:

Table: WDTCNT Taps

SSEL	IS1	IS0	Interval [ms]	
0	1	1	0.064	t SMCLK × 2 ^{6}
0	1	0	0.5	t SMCLK × 2 ^{9}
1	1	1	1.9	t ACLK × 2 ^{6}
0	0	1	8	t SMCLK × 2 ^{13}
1	1	0	16.0	t ACLK × 2 ^{9}
0	**0**	**0**	**32**	t SMCLK × 2 ^{15} <– Value after PUC (reset)
1	0	1	250	t ACLK × 2 ^{13}
1	0	0	1000	t ACLK × 2 ^{15}

Bit 2: The SSEL bit selects the clock source for WDTCNT.

SSEL = 0: WDTCNT is clocked by SMCLK .

SSEL = 1: WDTCNT is clocked by ACLK.

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

WDTCTL

Bit 4: The TMSEL bit 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 by the PUC signal.

NMI = 0: The RST/NMI input works as reset input.

As long as the RST/NMI pin is held low, the internal signal is active (level sensitive).

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

Bit 6: If the NMI function is selected, this bit selects the activating edge of the RST/NMI input. It is cleared by the PUC signal.

NMIES = 0: A rising edge triggers an NMI interrupt.

NMIES = 1: A falling edge triggers an NMI interrupt.

CAUTION: Changing the NMIES bit with software can generate an NMI interrupt.

Bit 7: This bit stops the operation of the watchdog counter. The clock multiplexer is disabled and the counter stops incrementing. It holds the last value until the hold bit is reset and the operation continues. It is cleared by the PUC signal.

HOLD = 0: The WDT is fully active.

HOLD = 1: The clock multiplexer and counter are stopped.

Watchdog Timer-Interrupt Function

The Watchdog Timer (WDT) uses two bits in the SFRs for interrupt control.

The WDT interrupt flag (WDTIFG) (located in IFG1.0, initial state is reset)

The WDT interrupt enable (WDTIE) (located in IE1.0, initial state is reset)

- When using the watchdog mode, the WDTIFG flag is used by the reset interrupt service routine to determine if the watchdog caused the device to reset. If the flag is set, then the Watchdog Timer initiated the reset condition (either by timing out or by a security key violation). If the flag is cleared, then the PUC was caused by a different source. See chapter 3 for more details on the PUC and POR signals.
- When using the Watchdog Timer in interval-timer mode, the WDTIFG flag is set after the selected time interval and a watchdog interval-timer interrupt is requested. The interrupt vector address in interval-timer mode is different from that in watchdog mode. In interval-timer mode, the WDTIFG flag is reset automatically when the interrupt is serviced.
- The WDTIE bit is used to enable or disable the interrupt from the Watchdog Timer when it is being used in interval-timer mode. Also, the GIE bit enables or disables the interrupt from the Watchdog Timer when it is being used in interval-timer mode.

Watchdog Timer-Timer Mode

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

- 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 it is serviced. The enable bit remains unchanged. In interval-timer mode, the WDT interrupt-enable bit and the GIE bit must be set to allow the WDT to request an interrupt. The interrupt vector address in timer mode is different from that in watchdog mode.

Watchdog Timer-Examples

- **How to select timer mode**

```
/* WDT is clocked by fACLK (assumed 32Khz) */
WDTCL=WDT_ADLY_250; // WDT 250MS/4 INTERVAL TIMER
IE1 |=WDTIE;         // ENABLE WDT INTERRUPT
```

- **How to stop watchdog timer**

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

- **Assembly programming**

```
WDT_key     .equ    05A00h                 ; Key to access WDT
WDTStop     mov    #(WDT_Key+80h),&WDTCTL ; Hold Watchdog
WDT250      mov    #(WDT_Key+1Dh),&WDTCTL ; WDT, 250ms Interval
```

MSP430x1xx Microcontrollers Low Power Modes

CPE/EE 421/521 Microcomputers

Power as a Design Constraint

Power becomes a first class architectural design constraint

- Why worry about power?
  - Battery life in portable and mobile platforms
  - Power consumption in desktops, server farms
    - Cooling costs, packaging costs, reliability, timing
    - Power density: 30 W/cm2 in Alpha 21364
      (3x of typical hot plate)
  - Environment?
    - IT consumes 10% of energy in the US

Where does power go in CMOS?

Dynamic power consumption

Power due to short-circuit current during transition

Power due to leakage current

$$P = \boxed{ACV^2 f} + \tau A V I_{short} f + V I_{leak}$$

Dynamic Power Consumption

C – Total capacitance seen by the gate's outputs Function of wire lengths, transistor sizes, ...

V – Supply voltage Trend: has been dropping with each successive fab

$$ACV^2f$$

A - Activity of gates How often on average do wires switch?

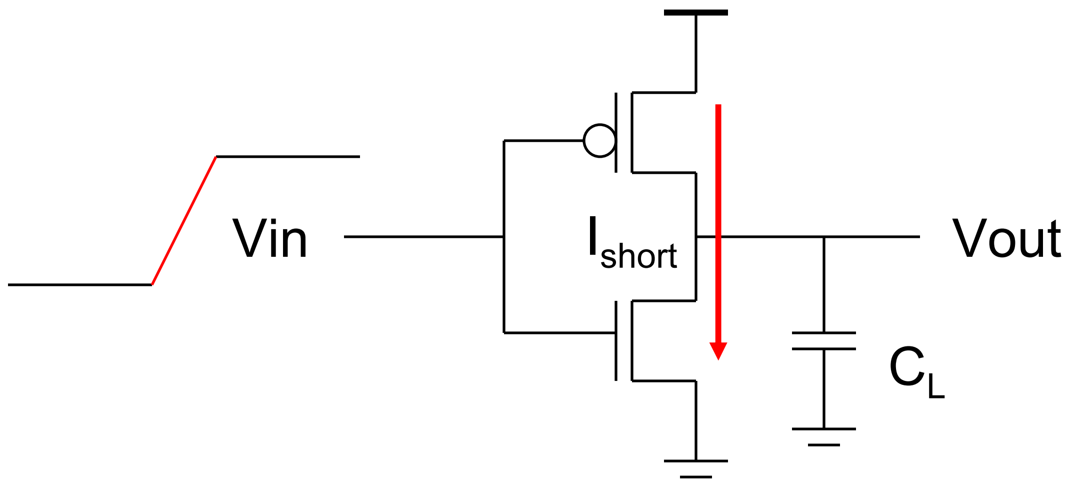
f – clock frequency Trend: increasing ...

Reducing Dynamic Power

1) **Reducing V has quadratic effect; Limits?**
2) **Lower C - shrink structures, shorten wires**
3) **Reduce switching activity - Turn off unused parts or use design techniques to minimize number of transitions**

Short-circuit Power Consumption

$$\tau A V I_{short} f$$

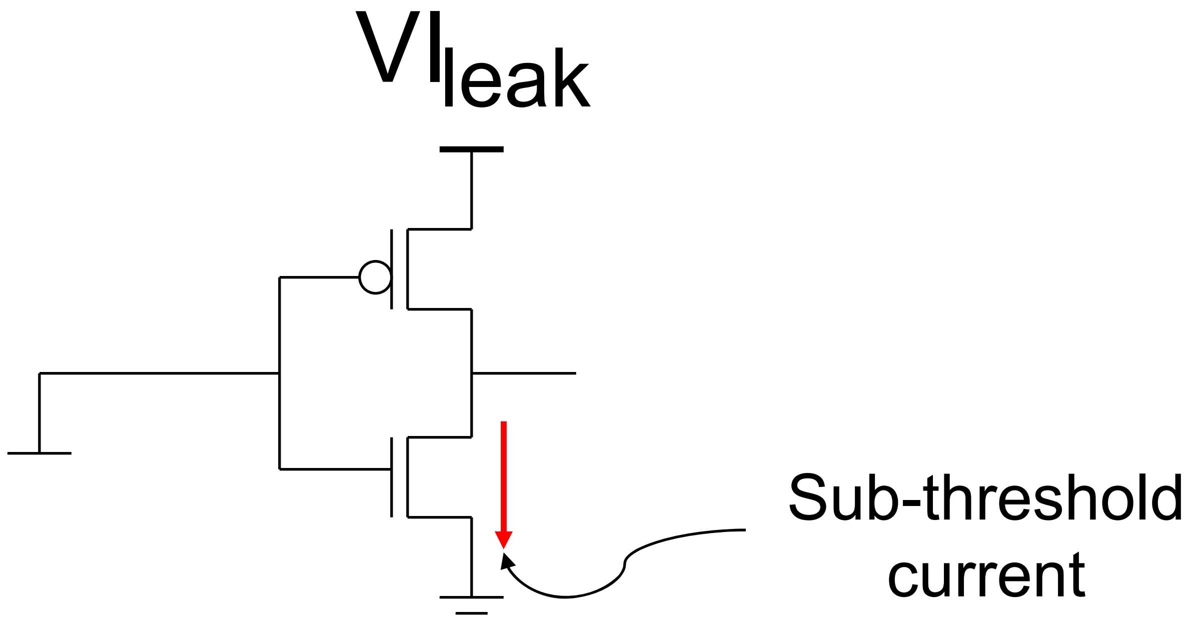


Finite slope of the input signal causes a direct current path between V_{DD} and GND for a short period of time during switching when both the NMOS and PMOS transistors are conducting

Reducing Short-circuit

1) **Lower the supply voltage V**
2) **Slope engineering – match the rise/fall time of the input and output signals**

Leakage Power



Sub-threshold current grows exponentially with increases in temperature and decreases in Vt

CMOS Power Equations

$$P = \boxed{ACV^2 f} + \tau AVI_{short} f + VI_{leak}$$

Reduce the supply voltage, V

$$f_{max} \propto \frac{(V - V_t)^2}{V}$$

Reduce threshold V_t

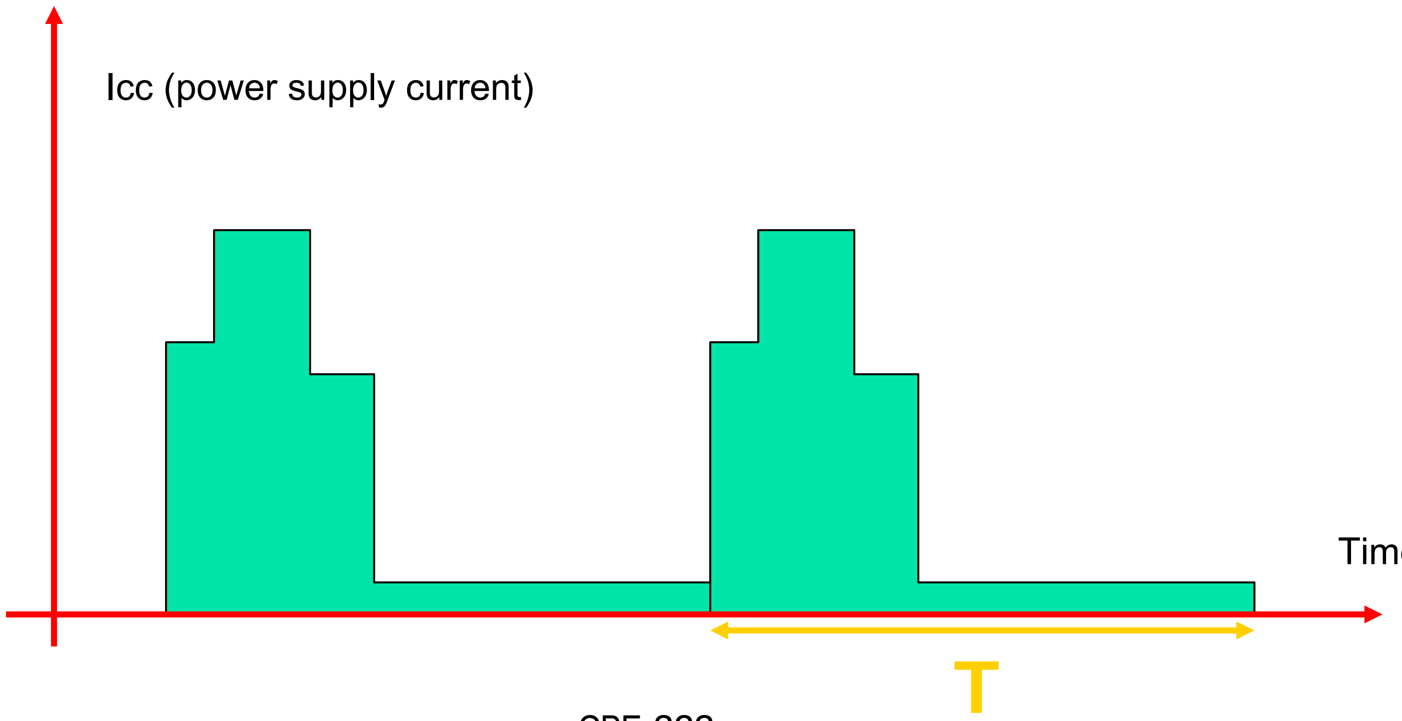
$$I_{leak} \propto \exp\left(-\frac{qV_t}{kT}\right)$$

How can we reduce power consumption?

- Dynamic power consumption
  - charge/discharge of the capacitive load on each gate’s output
  - frequency
- Control activity
  - reduce power supply voltage
  - reduce working frequency
  - turn off unused parts (module enables)
  - use low power modes
  - interrupt driven system
- Minimize the number of transitions
  - instruction formats, coding?

Average power consumption

- Dynamic power supply current
  - Set of modules that are periodically active
  - Typical situation – real time cycle T
  - Iave = ∫ Icc(t)dt /T
  - In most cases Iave = Σ Ii*ti/T



Low-Power Concept: Basic Conditions for Burst Mode

The example of the heat cost allocator shows that the current of the non-activity periode dominates the current consumption.

		Measure		Process data		Real-Time Clock		LCD Display
I_{AVG}	=	$I_{Measure}$	+	$I_{Calculate}$	+	I_{RTC}	+	$I_{Display}$
	=	$I_{ADC}{}^{*}\, t_{Measure}/T$	+	$I_{active} * t_{calc}/T$	+	$I_{active} * t_{RTC}/T$	+	$I_{Display}$
	=	3mA *200µs/60s	+	0.5mA * 10ms/60s	+	0.5mA * 0.5ms/60s	+	2.1µA
	=	10nA	+	83nA	+	4nA	+	2.1µA
I_{AVG}	≅							2.1µA

The ***sleep current dominates*** the current consumption!

The currents are related to the sensor and µC system. Additional current consumption of other system parts should be added for the total system current

Battery Life

- Battery Capacity BC – [mAh]
- Battery Life
  - BL = BC / Iave
- In the previous example, standard 800 mAh batteries will allow battery life of:
  - BL = 750 mAh / 2.1 μA $\approx$ 44 years !!!
- Conclusion:
  - Power efficient modes
  - Interrupt driven system with processor in idle mode

Power and Related metrics

- Peak power
  - Possible damage
- Dynamic power
  - Non-ideal battery characteristics
  - Ground bounce, di/dt noise
- Energy/operation ratio
  - MIPS/W
  - Energy x Delay

Reducing power consumption

- Logic
  - Clock tree (up to 30% of power)
  - Clock gating (turn off branches that are not used)
  - Half frequency clock (both edges)
  - Half swing clock (half of Vcc)
  - Asynchronous logic
    - completion signals
    - testing
- Architecture
  - Parallelism (increased area and wiring)
  - Speculation (branch prediction)
  - Memory systems
    - Memory access (dynamic)
    - Leakage
    - Memory banks (turn off unused)
  - Buses
    - 32-64 address/data, (15-20% of power)
    - Gray Code, Code compression

Reducing power consumption #2

- Operating System
  - Finish computation “when necessary”
  - Scale the voltage
    - Application driven
    - Automatic
- System Architecture
  - Power efficient and specialized processing cores
  - A “convergent” architecture
  - Trade-off
    - AMD K6 / 400MHz / 64KB cache – 12W
    - XScale with the same cache 450 mW @ 600 MHz (40mW@150MHz)
    - 24 processors? Parallelism?
- Other issues
  - Leakage current – Thermal runaway
  - Voltage clustering (low Vthreshold for high speed paths)

Operating Modes-General

The MSP430 family was developed for ultralow-power applications and uses different levels of operating modes. The MSP430 operating modes, give advanced support to various requirements for ultralow power and ultralow energy consumption. This support is combined with an intelligent management of operations during the different module and CPU states. An interrupt event wakes 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 ultra-low power system design which uses complementary metal-oxide semiconductor (CMOS) technology, takes into account three different needs:

- The desire for speed and data throughput despite conflicting needs for ultra-low power
- Minimization of individual current consumption
- Limitation of the activity state to the minimum required by the use of low power modes

Low power mode control

There are four bits that control the CPU and the main parts of the operation of the system clock generator:

- CPUOff,
- OscOff,
- SCG0, and
- SCG1.

These four bits support discontinuous active mode (AM) requests, to limit the time period of the full operating mode, and are located in the status register. The major advantage of including the operating mode bits in the status register is that the present state of the operating condition is saved onto the stack during an interrupt service request. 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.

Operating Modes-General

Another program flow may be selected by manipulating the data stored on the stack or the stack pointer. Being able to access the stack and stack pointer with the instruction set allows the program structures to be individually optimized, as illustrated in the following program flow:

Enter interrupt routine

The interrupt routine is entered and processed if an enabled interrupt awakens the MSP430:

- The SR and PC are stored on 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 modes are available to return from the interrupt service routine and continue the flow of operation:

- Return with low-power mode bits set. 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 low-power mode bits reset. When returning from the interrupt, the program continues at the address following the instruction that set the OscOff or CPUOff-bit in the status register. To use this mode, the interrupt service routine must reset the OscOff, CPUOff, SCGO, and SCG1 bits on the stack. Then, when the SR contents are popped from the stack upon RETI, the operating mode will be active mode (AM).

Operating Modes – Software configurable

There are six operating modes that the software can configure:

- Active mode AM; SCG1=0, SCG0=0, OscOff=0, CPUOff=0: CPU clocks are active
- Low power mode 0 (LPM0); SCG1=0, SCG0=0, OscOff=0, CPUOff=1:
  - CPU is disabled
  - MCLK is disabled
  - SMCLK and ACLK remain active
- Low power mode 1 (LPM1); SCG1=0, SCG0=1, OscOff=0, CPUOff=1:
  - CPU is disabled
  - MCLK is disabled
  - DCO’s dc generator is disabled if the DCO is not used for MCLK or SMCLK when in active mode. Otherwise, it remains enabled.
  - SMCLK and ACLK remain active
- Low power mode 2 (LPM2); SCG1=1, SCG0=0, OscOff=0, CPUOff=1:
  - CPU is disabled
  - MCLK is disabled
  - SMCLK is disabled
  - DCO oscillator automatically disabled because it is not needed for MCLK or SMCLK
  - DCO’s dc-generator remains enabled
  - ACLK remains active

Operating Modes #2

- Low power mode 3 (LPM3); SCG1=1, SCG0=1, OscOff=0, CPUOff=1:
  - CPU is disabled
  - MCLK is disabled
  - SMCLK is disabled
  - DCO oscillator is disabled
  - DCO’s dc-generator is disabled
  - ACLK remains active

- Low power mode 4 (LPM4); SCG1=X, SCG0=X, OscOff=1, CPUOff=1:
  - CPU is disabled
  - ACLK is disabled
  - MCLK is disabled
  - SMCLK is disabled
  - DCO oscillator is disabled
  - DCO’s dc-generator is disabled
  - Crystal oscillator is stopped

Operating Modes-Low Power Mode in details

- **Low-Power Mode 0 and 1 (LPM0 and LPM1)**

Low power mode 0 or 1 is selected if 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 stops. The operation of the CPU halts and all internal bus activities stop until an interrupt request or reset occurs. The system clock generator continues operation, and the clock signals MCLK, SMCLK, and ACLK stay active depending on the state of the other three status register bits, SCG0, SCG1, and OscOff.

The peripherals are enabled or disabled with their individual control register settings, and with the module enable registers in the SFRs. All I/O port pins and RAM/registers are unchanged. Wake up is possible through all enabled interrupts.

- **Low-Power Modes 2 and 3 (LPM2 and LPM3)**

Low-power mode 2 or 3 is selected if bits CPUOff and SCG1 in the status register are set. Immediately after the bits are set, CPU, MCLK, and SMCLK operations halt and all internal bus activities stop until an interrupt request or reset occurs.

Peripherals that operate with the MCLK or SMCLK signal are inactive because the clock signals are inactive. Peripherals that operate with the ACLK signal are active or inactive according with the individual control registers and the module enable bits in the SFRs. All I/O port pins and the RAM/registers are unchanged. Wake up is possible by enabled interrupts coming from active peripherals or RST/NMI.

Operating Modes - Low Power Mode in details

- **Low-Power Mode 4 (LPM4)**

System Resets, Interrupts, and Operating Modes In low power mode 4 all activities cease; only the RAM contents, I/O ports, and registers are maintained. Wake up is only possible by enabled external interrupts.

Before activating LPM4, the software should consider the system conditions during the low power mode period . The two most important conditions are environmental (that is, temperature effect on the DCO), and the clocked operation conditions.

The environment defines whether the value of the frequency integrator should be held or corrected. A correction should be made when ambient conditions are anticipated to change drastically enough to increase or decrease the system frequency while the device is in LPM4.

Operating Modes-Examples

- **The following example describes entering into low-power mode 0.**

```
;===Main program flow with switch to CPUOff Mode==============
BIS #18h,SR ;Enter LPM0 + enable general interrupt GIE
             ;(CPUOff=1, GIE=1). The PC is incremented
             ;during execution of this instruction and
             ;points to the consecutive program step.
......       ;The program continues here if the CPUOff
             ;bit is reset during the interrupt service
             ;routine. Otherwise, the PC retains its
             ;value and the processor returns to LPM0.
```

- **The following example describes clearing low-power mode 0.**

```
;===Interrupt service routine=================================
......                ;CPU is active while handling interrupts
BIC #10h,0(SP)        ;Clears the CPUOff bit in the SR contents
                      ;that were stored on the stack.
RETI                  ;RETI restores the CPU to the active state
                      ;because the SR values that are stored on
                      ;the stack were manipulated. This occurs
                      ;because the SR is pushed onto the stack
                      ;upon an interrupt, then restored from the
                      ;stack after the RETI instruction.
```

Operating Modes C Examples

- C – programming msp430x14x.h

```
/************************
* STATUS REGISTER BITS
************************/

#define C       0x0001
#define Z       0x0002
#define N       0x0004
#define V       0x0100
#define GIE     0x0008
#define CPUOFF  0x0010
#define OSCOFF  0x0020
#define SCG0    0x0040
#define SCG1    0x0080

/* Low Power Modes coded with
    Bits 4-7 in SR */
/* Begin #defines for assembler */
#ifndef __IAR_SYSTEMS_ICC
#define LPM0    CPUOFF
#define LPM1    SCG0+CPUOFF
#define LPM2    SCG1+CPUOFF
#define LPM3    SCG1+SCG0+CPUOFF
#define LPM4    SCG1+SCG0+OSCOFF+CPUOFF
/* End #defines for assembler */

#else /* Begin #defines for C */
#define LPM0_bits   CPUOFF
#define LPM1_bits   SCG0+CPUOFF
#define LPM2_bits   SCG1+CPUOFF
#define LPM3_bits   SCG1+SCG0+CPUOFF
#define LPM4_bits   SCG1+SCG0+OSCOFF+CPUOFF
```

- …

```
#include "In430.h"

#define LPM0      _BIS_SR(LPM0_bits) /* Enter LP Mode 0 */
#define LPM0_EXIT _BIC_SR(LPM0_bits) /* Exit LP Mode 0 */
#define LPM1      _BIS_SR(LPM1_bits) /* Enter LP Mode 1 */
#define LPM1_EXIT _BIC_SR(LPM1_bits) /* Exit LP Mode 1 */
#define LPM2      _BIS_SR(LPM2_bits) /* Enter LP Mode 2 */
#define LPM2_EXIT _BIC_SR(LPM2_bits) /* Exit LP Mode 2 */
#define LPM3      _BIS_SR(LPM3_bits) /* Enter LP Mode 3 */
#define LPM3_EXIT _BIC_SR(LPM3_bits) /* Exit LP Mode 3 */
#define LPM4      _BIS_SR(LPM4_bits) /* Enter LP Mode 4 */
#define LPM4_EXIT _BIC_SR(LPM4_bits) /* Exit LP Mode 4 */
#endif /* End #defines for C */
```

```
/* - in430.h -
    Intrinsic functions for the MSP430
*/

unsigned short _BIS_SR(unsigned short);
unsigned short _BIC_SR(unsigned short);
```

C Examples

```
//******************************************************************************
//  MSP-FET430P140 Demo - WDT Toggle P1.0, Interval ISR, 32kHz ACLK
//
//  Description; Toggle P1.0 using software timed by WDT ISR.
//  Toggle rate is exactly 250ms based on 32kHz ACLK WDT clock source.
//  In this example the WDT is configured to divide 32768 watch-crystal(2^15)
//  by 2^13 with an ISR triggered @ 4Hz.
//  ACLK= LFXT1= 32768, MCLK= SMCLK= DCO~ 800kHz
//  //*External watch crystal installed on XIN XOUT is required for ACLK*
//
//
//               MSP430F149
//             -----------------
//         /|\|              XIN|-
//          | |                 | 32kHz
//          --|RST          XOUT|-
//            |                 |
//            |             P1.0|-->LED
//
//  M.Buccini
//  Texas Instruments, Inc
//  August 2003
//  Built with IAR Embedded Workbench Version: 1.26B
//  December 2003
//  Updated for IAR Embedded Workbench Version: 2.21B
//************************************************************
```

```
#include <msp430x14x.h>

void main(void)
{
 // WDT 250ms, ACLK, interval timer
  WDTCTL = WDT_ADLY_250;
  IE1 |= WDTIE;   // Enable WDT
    interrupt
  P1DIR |= 0x01; // Set P1.0 to
    output direction
 // Enter LPM3 w/interrupt
  _BIS_SR(LPM3_bits + GIE);
}

// Watchdog Timer interrupt service
    routine
interrupt[WDT_TIMER] void
    watchdog_timer(void)
{
  P1OUT ^= 0x01;   // Toggle P1.0
    using exclusive-OR
}
```

C Examples

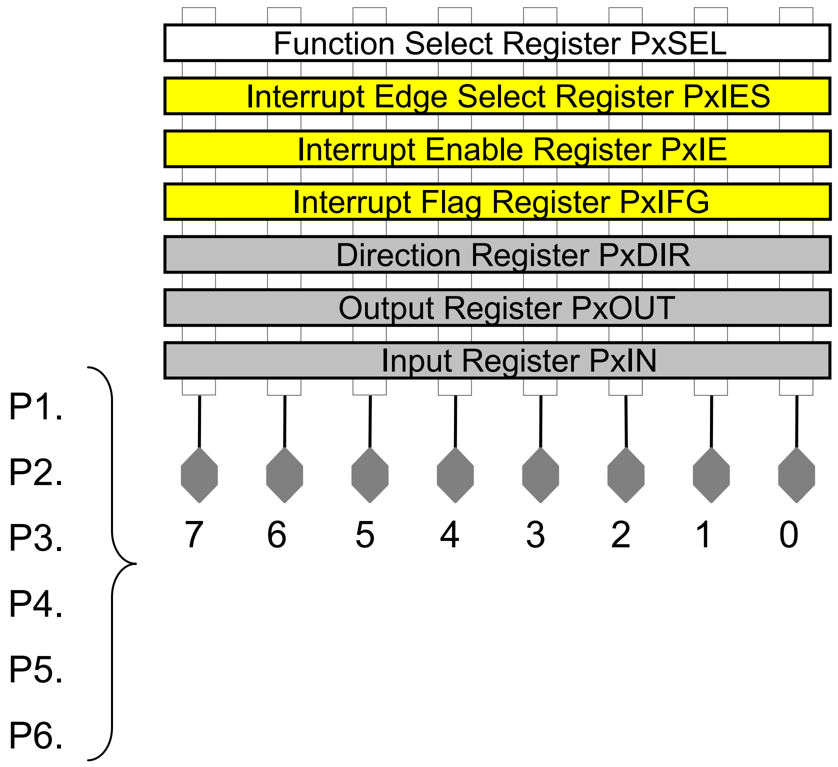
```
....
_BIS_SR(LPM0_bits + GIE);          // Enter LPM0 w/ interrupt
// program stops here
```

QQ?

Your program is in LPM0 mode and it is woke up by an interrupt. What should be done if you do not want to go back to LPM0 after servicing the interrupt request, but rather you would let the main program re-enter LMP0, based on current conditions?

MSP430: Digital I/O

Digital I/O


Function Select Register PxSEL
Interrupt Edge Select Register PxIES
Interrupt Enable Register PxIE
Interrupt Flag Register PxIFG
Direction Register PxDIR
Output Register PxOUT
Input Register PxIN
P1.
P2.
P3.
P4.
P5.
P6.
7
6
5
4
3
2
1
0

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

Chapter 9, User's Manual
pages 9-1 to 9-7

Digital I/O Introduction

- MSP430 family – up to 6 digital I/O ports implemented, P1-P6
- MSP430F14x – all 6 ports implemented
- Ports P1 and P2 have interrupt capability.
- Each interrupt for the P1 and P2 I/O lines can be individually enabled and configured to provide an interrupt on a rising edge or falling edge of an input signal.

- The digital I/O features include:
  - Independently programmable individual I/Os
  - Any combination of input or output
  - Individually configurable P1 and P2 interrupts
  - Independent input and output data registers

- The digital I/O is configured with user software

Digital I/O Registers Operation

- Input Register PnIN
  - Each bit in each PnIN register reflects the value of the input signal at the corresponding I/O pin when the pin is configured as I/O function.
    - Bit = 0: The input is low
    - Bit = 1: The input is high

Do not write to PxIN. It will result in increased current consumption

- Output Registers PnOUT
  - Each bit in each PnOUT register is the value to be output on the corresponding I/O pin when the pin is configured as I/O function and output direction.
    - Bit = 0: The output is low
    - Bit = 1: The output is high

Digital I/O Operation

- Direction Registers PnDIR
  - Bit = 0: The port pin is switched to input direction
  - Bit = 1: The port pin is switched to output direction

- Function Select Registers PnSEL
  - Port pins are often multiplexed with other peripheral module functions.
    - Bit = 0: I/O Function is selected for the pin
    - Bit = 1: Peripheral module function is selected for the pin

Digital I/O Operation

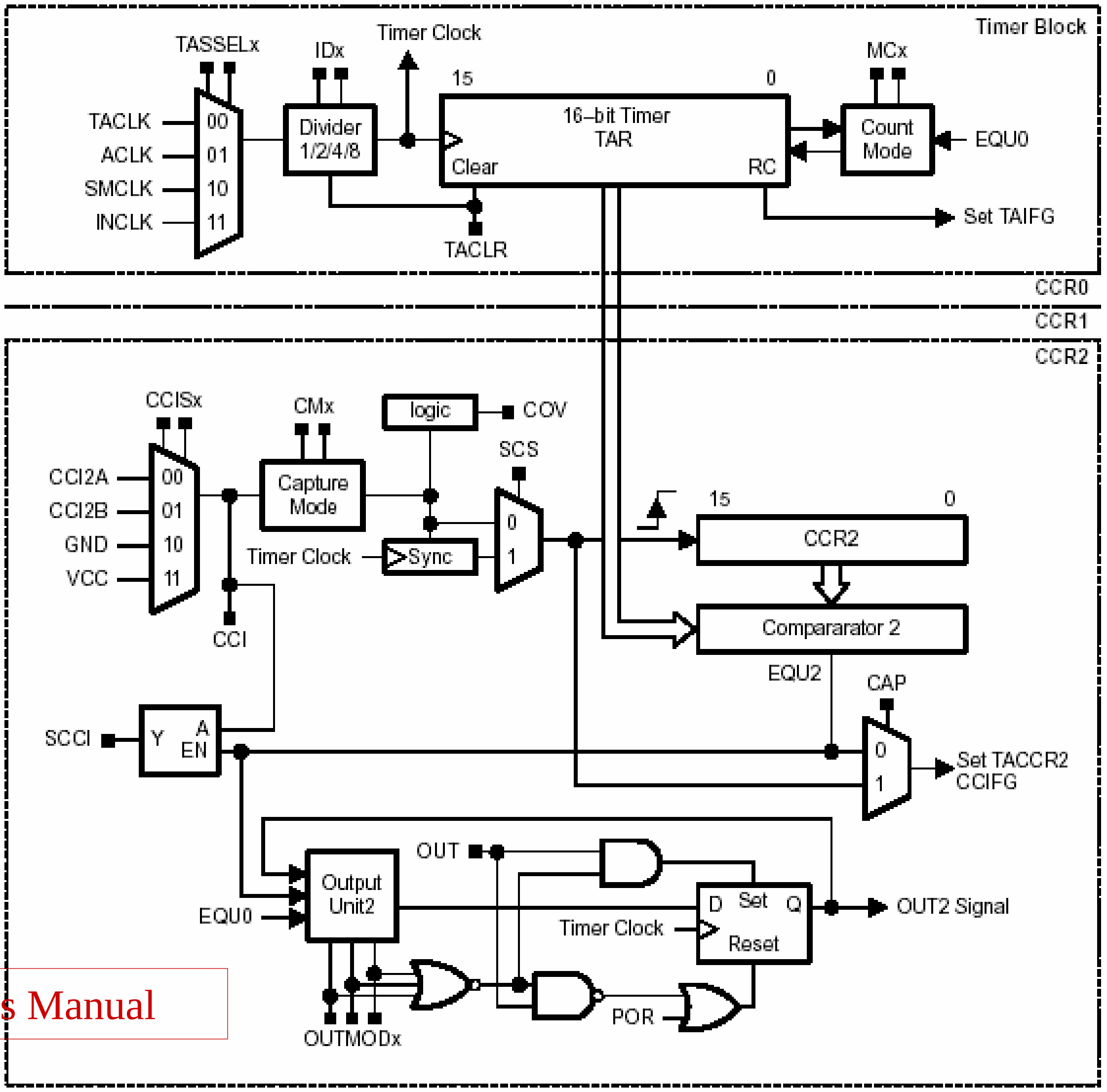
- Interrupt Flag Registers P1IFG, P2IFG (only for P1 and P2)
  - Bit = 0: No interrupt is pending
  - Bit = 1: An interrupt is pending
- Only transitions, not static levels, cause interrupts
- Interrupt Edge Select Registers P1IES, P2IES
  - (only for P1 and P2)
- Each PnIES bit selects the interrupt edge for the corresponding I/O pin.
  - Bit = 0: The PnIFGx flag is set with a low-to-high transition
  - Bit = 1: The PnIFGx flag is set with a high-to-low transition

MSP430: Timer_A

Timer_A MSP430x1xx

- 16-bit counter with 4 operating modes
- Selectable and configurable clock source
- Three (or five) independently configurable capture/compare registers with configurable inputs
- Three (or five) individually configurable output modules with 8 output modes
- multiple, simultaneous, timings; multiple capture/compares; multiple output waveforms such as PWM signals; and any combination of these.
- Interrupt capabilities
  - each capture/compare block individually configurable

Timer_A5 - MSP430x1xx Block Diagram



Page 11-3, User's Manual

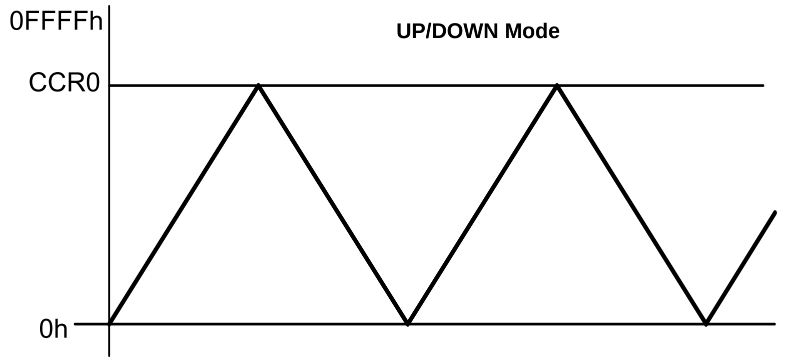
Timer_A Counting Modes

Stop/Halt Mode

Timer is halted with the next +CLK

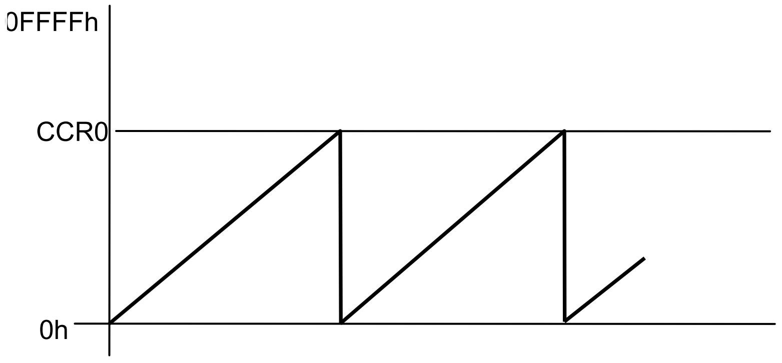
UP/DOWN Mode

Timer counts between 0 and CCR0 and 0



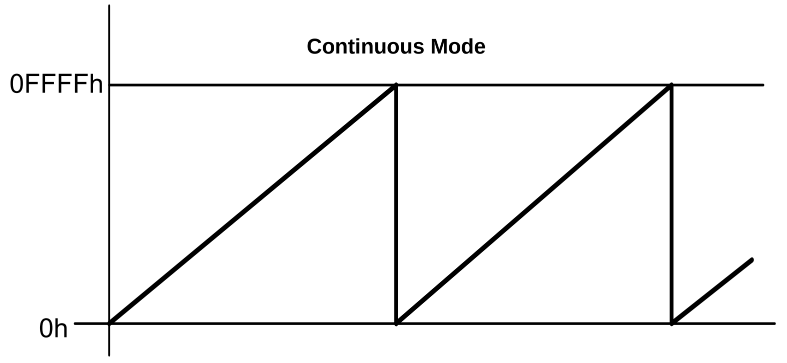
UP Mode

Timer counts between 0 and CCR0

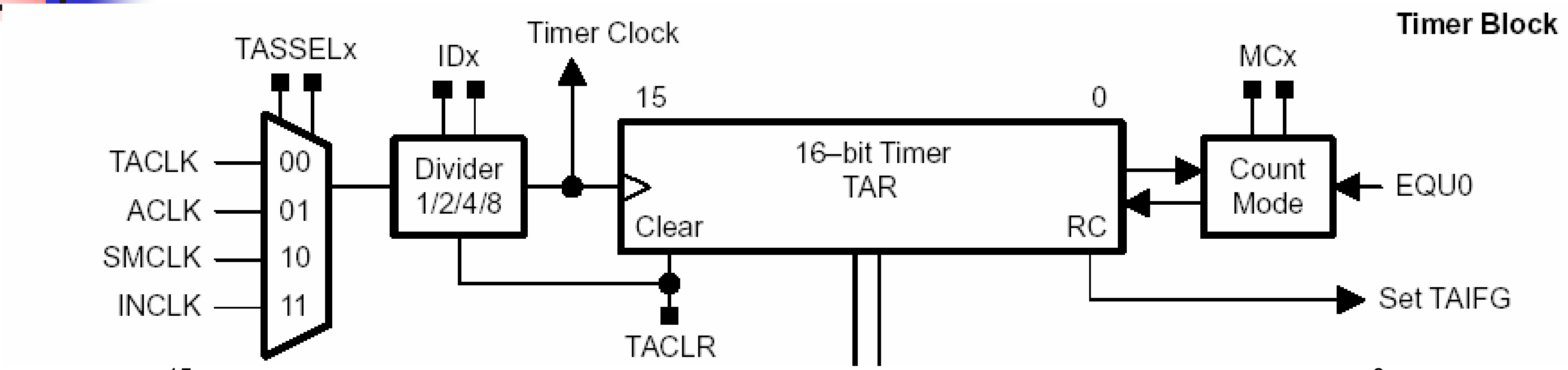


Continuous Mode

Timer continuously counts up



Timer_A 16-bit Counter



TACTL

160h

15															0
unused						Input Select		Input Divider		Mode Control		un-used	CLR	TAIE	TAIFG
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)

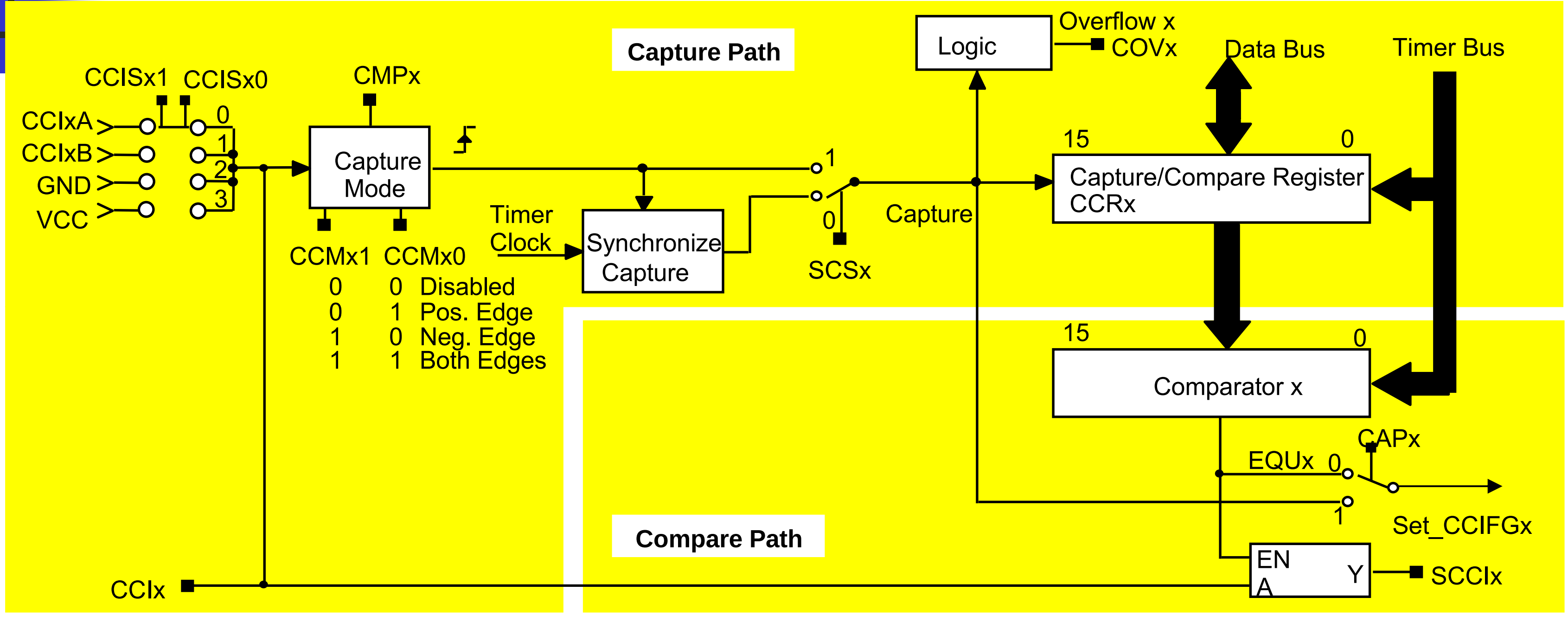
MC1	MC0	
0	0	Stop Mode
0	1	Up Mode
1	0	Continuous Mode
1	1	Up/Down Mode

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

SSEL1	SSEL0	
0	0	TACLK
0	1	ACLK
1	0	MCLK
1	1	INCLK

Page 11-12, User's Manual

Timer_A Capture Compare Blocks



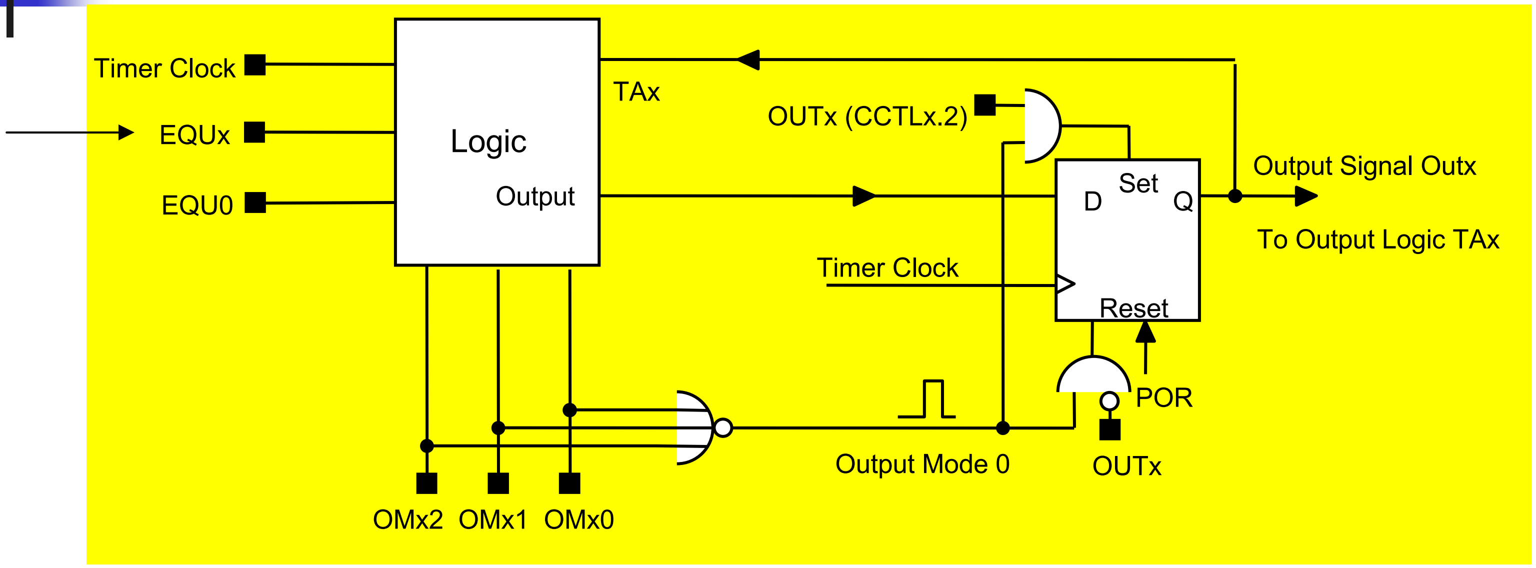
CCRx
0172h
to
017Eh

15															0
2^{15}															2^{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)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

CCTLx
162h
to
16Eh

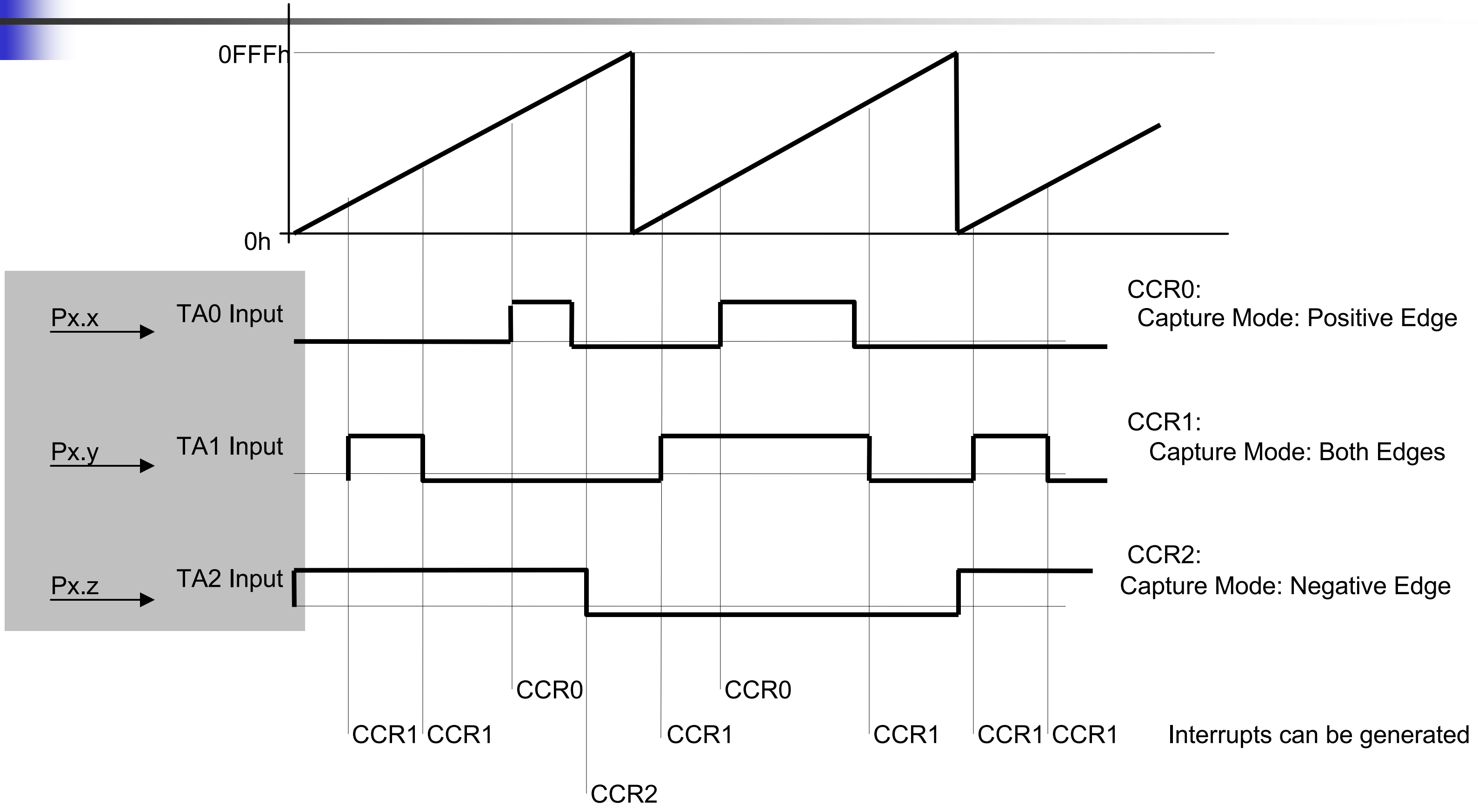
15															0
CAPTURE MODE		INPUT SELECT		SCS	SCCI	un-used	CAP	OUTMODx			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

Timer_A Output Units



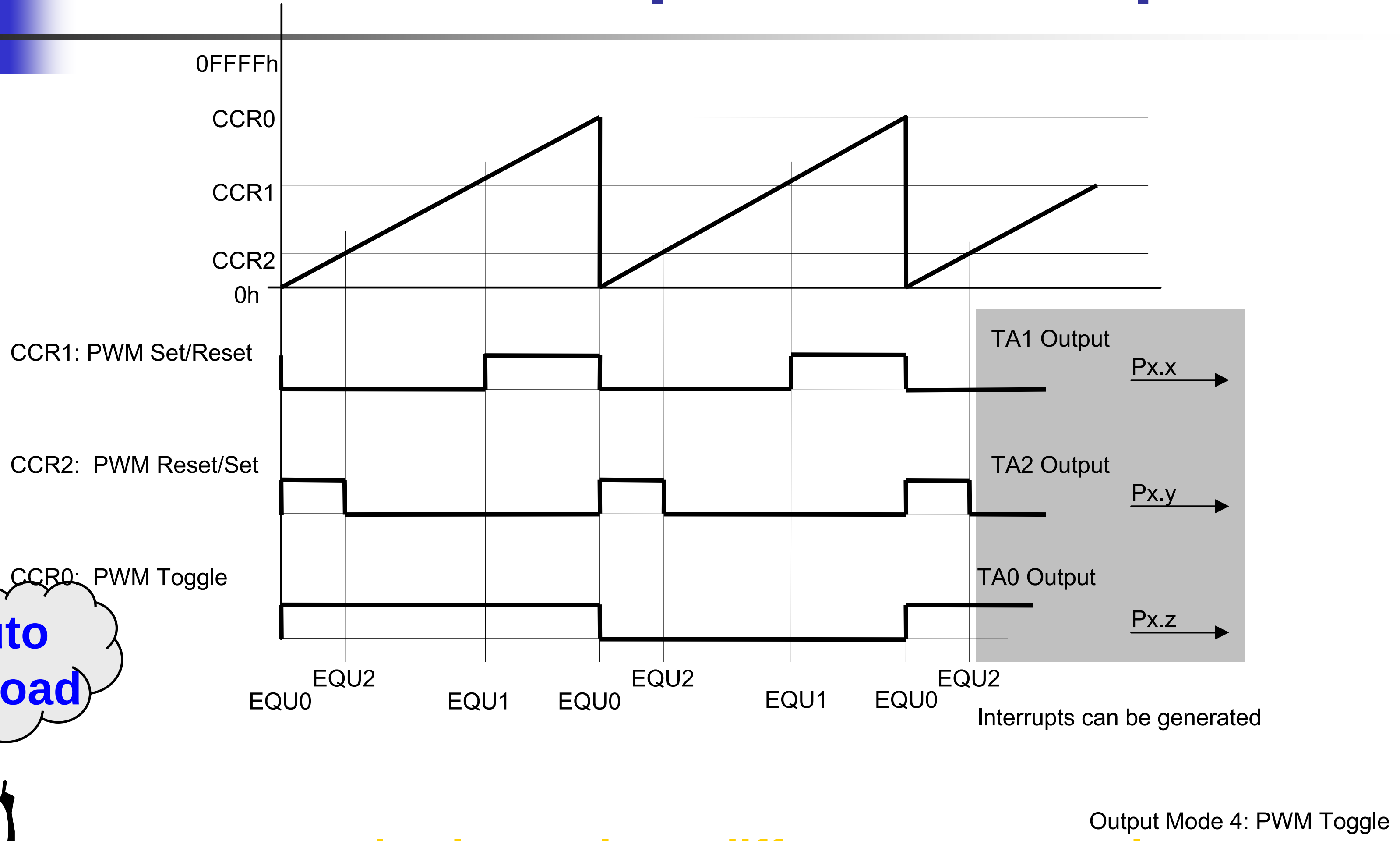
OMx2	OMx1	OMx0	Function	Operational Conditions
0	0	0	Output Mode	Outx signal is set according to Outx bit
0	0	1	Set	EQUx sets Outx signal clock synchronous with timer clock
0	1	0	PWM Toggle/Reset	EQUx toggles Outx signal, reset with EQU0, clock sync. with timer clock
0	1	1	PWM Set/Reset	EQUx sets Outx signal, reset with EQU0, clock synchronous with timer clock
1	0	0	Toggle	EQUx toggles Outx signal, clock synchronous with timer clock
1	0	1	Reset	EQUx resets Outx signal clock synchronous with timer clock
1	1	0	PWM Toggle/Reset	EQUx toggles Outx signal, set with EQU0, clock synchronous with timer clock
1	1	1	PWM Set/Reset	EQUx resets Outx signal, set with EQU0, clock synchronous with timer clock

Timer_A Continuous-Mode Example



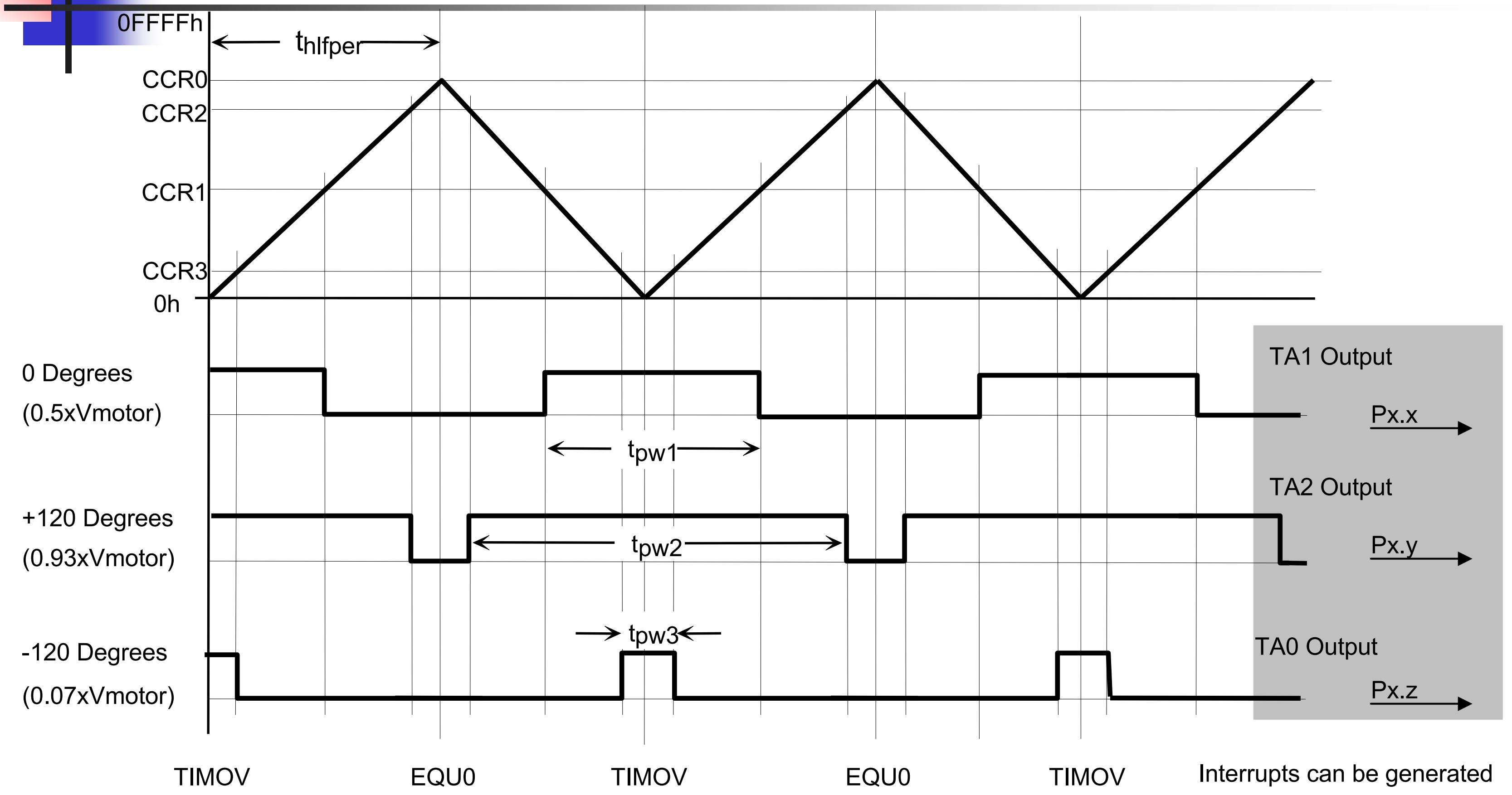
Example shows three independent HW event captures. CCRx "stamps" time of event - Continuous-Mode is ideal.

Timer_A PWM Up-Mode Example



Example shows three different asymmetric PWM-Timings generated with the Up-Mode

Timer_A PWM Up/Down Mode Example



Example shows Symmetric PWM Generation - Digital Motor Control

C Examples, CCR0 Contmode ISR, TA_0 ISR

```
//*****************************************************************
//  MSP-FET430P140 Demo - Timer_A Toggle P1.0,
//  CCR0 Contmode ISR, DCO SMCLK
//  Description; Toggle P1.0 using software and TA_0 ISR. Toggle rate is
//  set at 50000 DCO/SMCLK cycles. Default DCO frequency used for TACLK.
//  Durring the TA_0 ISR P0.1 is toggled and 50000 clock cycles are added to
//  CCR0.  TA_0 ISR is triggered exactly 50000 cycles. CPU is normally off
    and
//  used only durring TA_ISR.
//  ACLK = n/a, MCLK = SMCLK = TACLK = DCO~ 800k
//
//
//           MSP430F149
//         ---------------
//     /|\|            XIN|-
//      | |               |
//      --|RST        XOUT|-
//        |               |
//        |           P1.0|-->LED
//
//  M. Buccini
//  Texas Instruments, Inc
//  September 2003
//  Built with IAR Embedded Workbench Version: 1.26B
//  December 2003
//  Updated for IAR Embedded Workbench Version: 2.21B
//***********************************************************************
```

```
#include <msp430x14x.h>

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;             // Stop WDT
  P1DIR |= 0x01;                        // P1.0 output
  CCTL0 = CCIE;          // CCR0 interrupt enabled
  CCR0 = 50000;
  TACTL = TASSEL_2 + MC_2; // SMCLK, contmode

  _BIS_SR(LPM0_bits + GIE); // Enter LPM0 w/ interrupt
}

// Timer A0 interrupt service routine
interrupt[TIMERA0_VECTOR] void TimerA(void)
{
  P1OUT ^= 0x01; // Toggle P1.0
  CCR0 += 50000; // Add Offset to CCR0
}
```

C Examples, CCR0 Upmode ISR, TA_0

```
//*****************************************************************************
//  MSP-FET430P140 Demo - Timer_A Toggle P1.0, CCR0 upmode ISR, 32kHz ACLK
//
//  Description; Toggle P1.0 using software and the TA_0 ISR. Timer_A is
//  configured in an upmode, thus the the timer will overflow when TAR
     counts
//  to CCR0. In this example, CCR0 is loaded with 1000-1.
//  Toggle rate = 32768/(2*1000) = 16.384
//  ACLK = TACLK = 32768, MCLK = SMCLK = DCO~ 800k
//  //*An external watch crystal on XIN XOUT is required for ACLK*//

//
//           MSP430F149
//         ---------------
//     /|\|            XIN|-
//      | |               | 32kHz
//      --|RST        XOUT|-
//        |               |
//        |           P1.0|-->LED
//
//  M. Buccini
//  Texas Instruments, Inc
//  October 2003
//  Built with IAR Embedded Workbench Version: 1.26B
//  December 2003
//  Updated for IAR Embedded Workbench Version: 2.21B
//*****************************************************************************
```

```
#include <msp430x14x.h>

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD; // Stop WDT
  P1DIR |= 0x01;  // P1.0 output
  CCTL0 = CCIE;   // CCR0 interrupt enabled
  CCR0 = 1000-1;
  TACTL = TASSEL_1 + MC_1; // ACLK, upmode

  _BIS_SR(LPM3_bits + GIE); // Enter LPM3 w/
     interrupt
}

// Timer A0 interrupt service routine
#pragma vector=TIMERA0_VECTOR
Interrupt[TIMERA0_VECTOR] void Timer_A (void)
{
  P1OUT ^= 0x01; // Toggle P1.0
}
```

C Examples, CCR1 Contmode ISR, TA_1

```
//*******************************************************************
//  MSP-FET430P140 Demo –
// Timer_A Toggle P1.0, CCR1 Contmode ISR, CO SMCLK
//  Description; Toggle P1.0 using using software and TA_1 ISR.
// Toggle rate is set at 50000 DCO/SMCLK cycles.
// Default DCO frequency used for TACLK.
//  Durring the TA_1 ISR P0.1 is toggled and
// 50000 clock cycles are added to CCR1.
// TA_1 ISR is triggered exactly 50000 cycles.
// CPU is normally off and used only durring TA_ISR.
//  ACLK = n/a, MCLK = SMCLK = TACLK = DCO ~ 800k
//  Proper use of TAIV interrupt vector generator demonstrated.
//
//           MSP430F149
//         ---------------
//     /|\|            XIN|-
//      | |               |
//      --|RST        XOUT|-
//        |               |
//        |           P1.0|-->LED
//
//  M. Buccini
//  Texas Instruments, Inc
//  September 2003
//  Built with IAR Embedded Workbench Version: 1.26B
//  December 2003
//  Updated for IAR Embedded Workbench Version: 2.21B
//****************************************************************
```

```
#include <msp430x14x.h>

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD; // Stop WDT
  P1DIR |= 0x01; // P1.0 output
  CCTL1 = CCIE;  // CCR1 interrupt enabled
  CCR1 = 50000;
  TACTL = TASSEL_2 + MC_2; // SMCLK, Contmode

  _BIS_SR(LPM0_bits + GIE); // Enter LPM0 w/
    interrupt
}
// Timer_A3 Interrupt Vector (TAIV) handler
#pragma vector=TIMERA1_VECTOR
__interrupt void Timer_A(void)
{
  switch( TAIV )
  {
  case  2:                  // CCR1
    {
    P1OUT ^= 0x01;          // Toggle P1.0
    CCR1 += 50000;          // Add Offset to CCR1
    }
           break;
  case  4: break;           // CCR2 not used
  case 10: break;           // overflow not used
 }
}
```

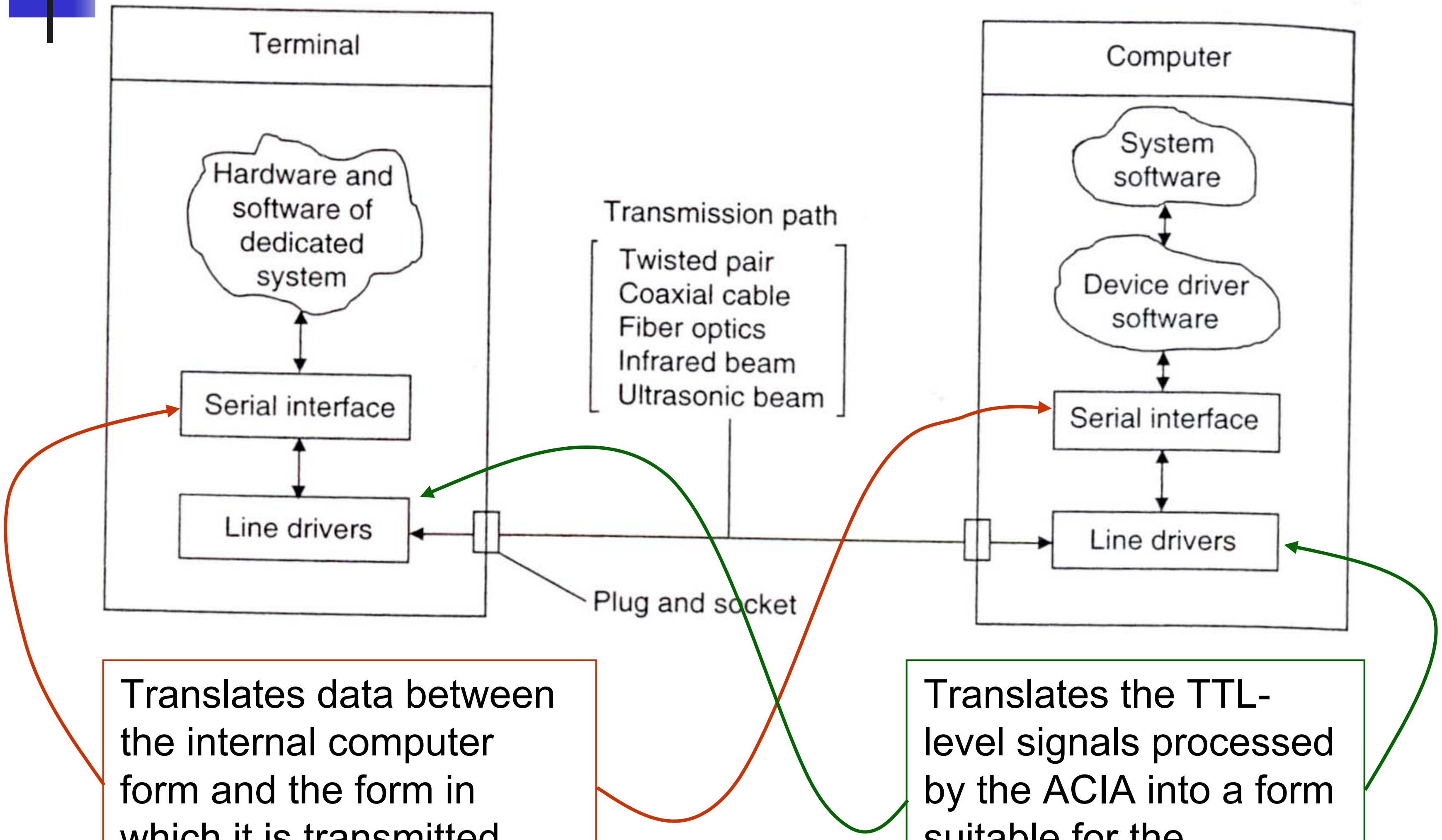
C Examples, PWM, TA1-2 upmode

```
//******************************************************************************
//  MSP-FET430P140 Demo - Timer_a PWM TA1-2 upmode, DCO SMCLK
//
//  Description; This program will generate a two PWM outputs on P1.2/1.3 using
//  Timer_A in an upmode.  The value in CCR0, defines the period and the
//  values in CCR1 and CCR2 the duty PWM cycles.  Using ~ 800kHz SMCLK as TACLK,
//  the timer period is ~ 640us with a 75% duty cycle on P1.2 and 25% on P1.3.
//  ACLK = na, SMCLK = MCLK = TACLK = default DCO ~ 800kHz.
//
//               MSP430F149
//            -----------------
//        /|\|              XIN|-
//         | |                 |
//         --|RST          XOUT|-
//           |                 |
//           |             P1.2|--> CCR1 - 75% PWM
//           |             P1.3|--> CCR2 - 25% PWM
//
//  M.Buccini
//  Texas Instruments, Inc
//  September 2003
//  Built with IAR Embedded Workbench Version: 1.26B
//  January 2004
//  Updated for IAR Embedded Workbench Version: 2.21B
//******************************************************
```

```
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;   // Stop WDT
  P1DIR |= 0x0C;   // P1.2 and P1.3 output
  P1SEL |= 0x0C; // P1.2 and P1.3 TA1/2 options
  CCR0 = 512-1;          // PWM Period
  CCTL1 = OUTMOD_7;      // CCR1 reset/set
  CCR1 = 384;            // CCR1 PWM duty cycle
  CCTL2 = OUTMOD_7;      // CCR2 reset/set
  CCR2 = 128;            // CCR2 PWM duty cycle
  TACTL = TASSEL_2 + MC_1;  // SMCLK, up mode

  _BIS_SR(LPM0_bits);         // Enter LPM0
}
```

Serial Communication

Serial I/O Interface
Functional Units



Translates data between the internal computer form and the form in which it is transmitted over the data link

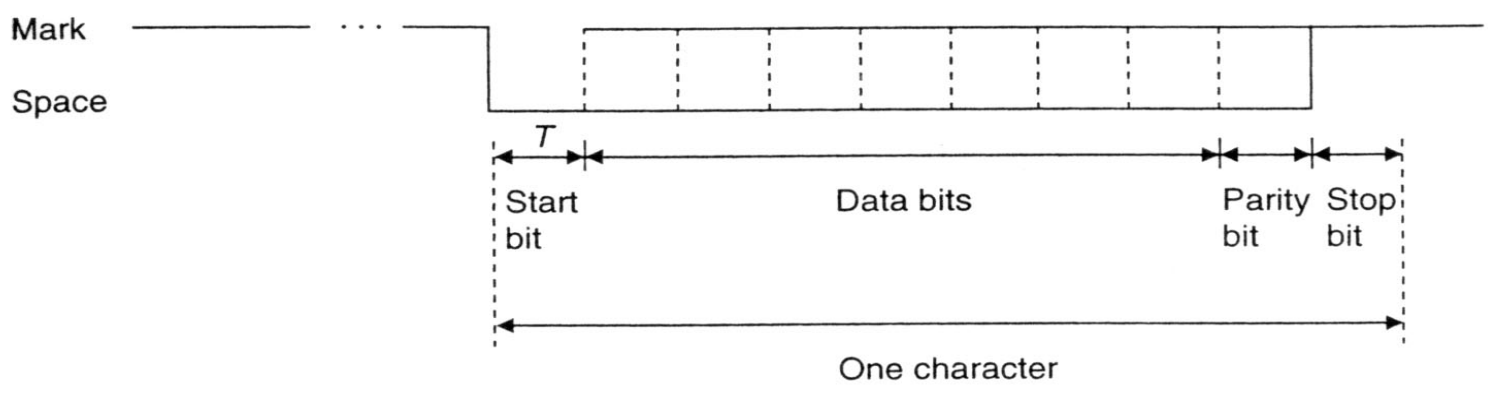
Translates the TTL-level signals processed by the ACIA into a form suitable for the transmission path

Asynchronous Serial Interface

- Asynchronous
  - Transmitted and received data are not synchronized over any extended period
  - No synchronization between receiver and transmitter clocks
- Serial
  - Usually character oriented
  - Data stream divided into individual bits at the transmitter side
  - Individual bits are grouped into characters at the receiving side
- Information is usually transmitted as ASCII-encoded characters
  - 7 or 8 bits of information plus control bits

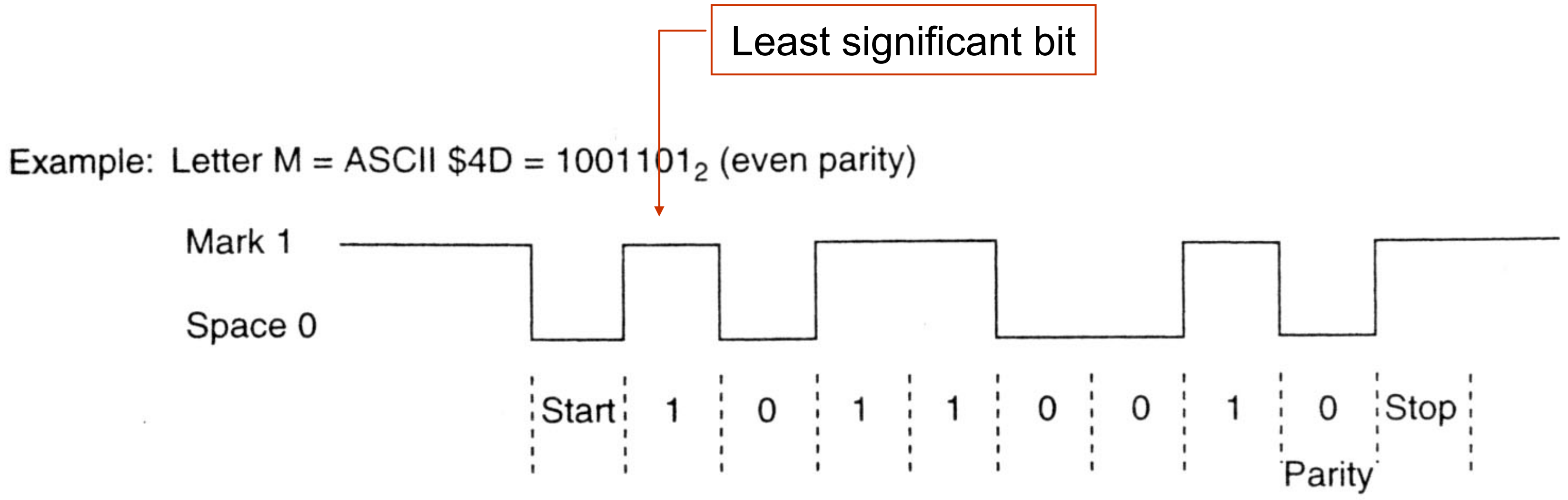
Asynchronous Serial Interface, cont'd

- MARK level (or OFF, or 1-state, or 1-level)
  - This is also the idle state (before the transfer begins)
- SPACE level (or ON, or 0-state, or 0-level)
- One character:
  - Start bit: space level
  - Data bits
  - Optional parity bit
  - Optional stop bit

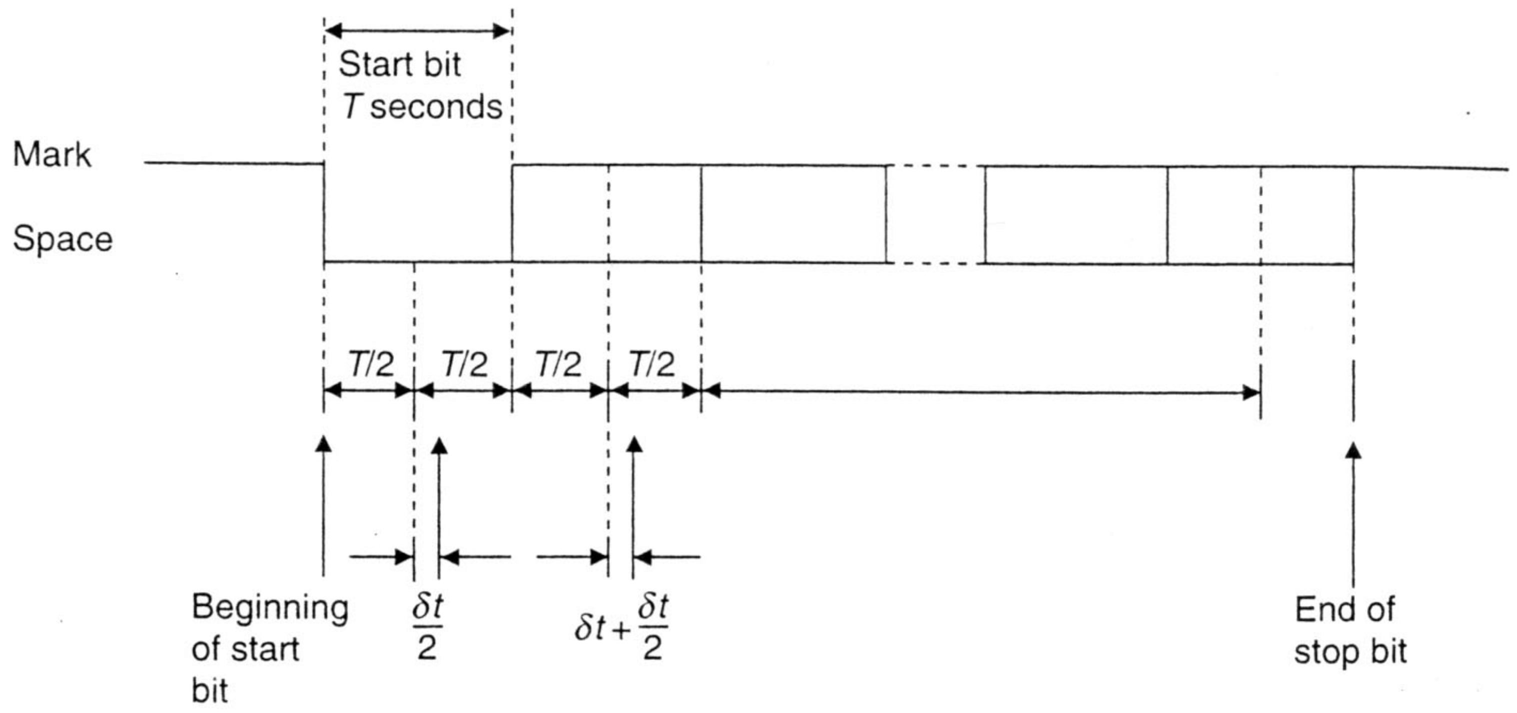


Asynchronous Serial Interface, cont'd

- 12 possible basic formats:
  - 7 or 8 bits of data
  - Odd, even, or no parity
  - 1 or 2 stop bits
  - Others exist also: no stop bits, 4/5/6 data bits, 1.5 stop bits, etc.



Receiver Clock Timing



- For N=9 bits (7 data + parity + stop) maximum tolerable error is 5% (assume that the receiver clock is slow -- [T + δt] instead of T)

  T/2 > (2N+1)δt/2

  δt/2 < 1/(2N+1)

  δt/T < 100/(2N+1) as a percentage

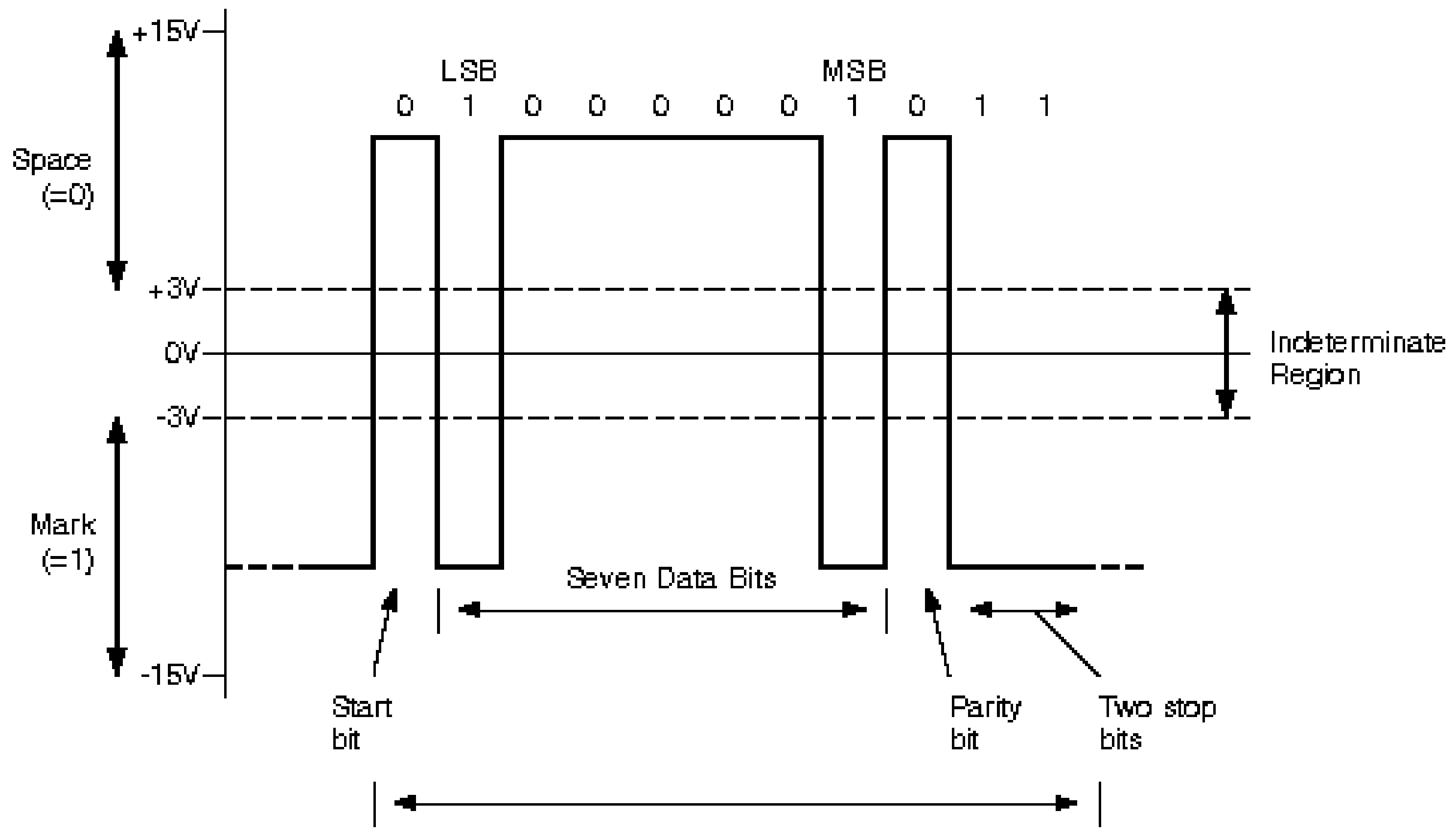
RS-232 Interface Standard

- Bi-polar:
  - +3 to +12V (ON, 0-state, or SPACE condition)
  - -3 to –12V (OFF, 1-state, or MARK condition)
- Modern computers accept 0V as MARK
- “Dead area” between –3V and 3V is designed to absorb line noise
- Originally developed as a standard for communication between computer equipment and modems
- From the point of view of this standard:
  - MODEM: data communications equipment (DCE)
  - Computer equipment: data terminal equipment (DTE)
- Therefore, RS-232C was intended for DTE-DCE links (not for DTE-DTE links, as it is frequently used now)

RS-232 Interface Standard

- Each manufacturer may choose to implement only a subset of functions defined by this standard
- Two widely used connectors: DB-9 and DB-25
- Three types of link
  - Simplex
  - Half-duplex
  - Full-duplex
- Basic control signals
  - RTS (Request to send):
    DTE indicates to the DCE that it wants to send data
  - CTS (Clear to send):
    DCE indicates that it is ready to receive data
  - DSR (Data set ready):
    indication from the DCE (i.e., the modem) that it is on
  - DTR (Data terminal ready):
    indication from the DTE that it is on

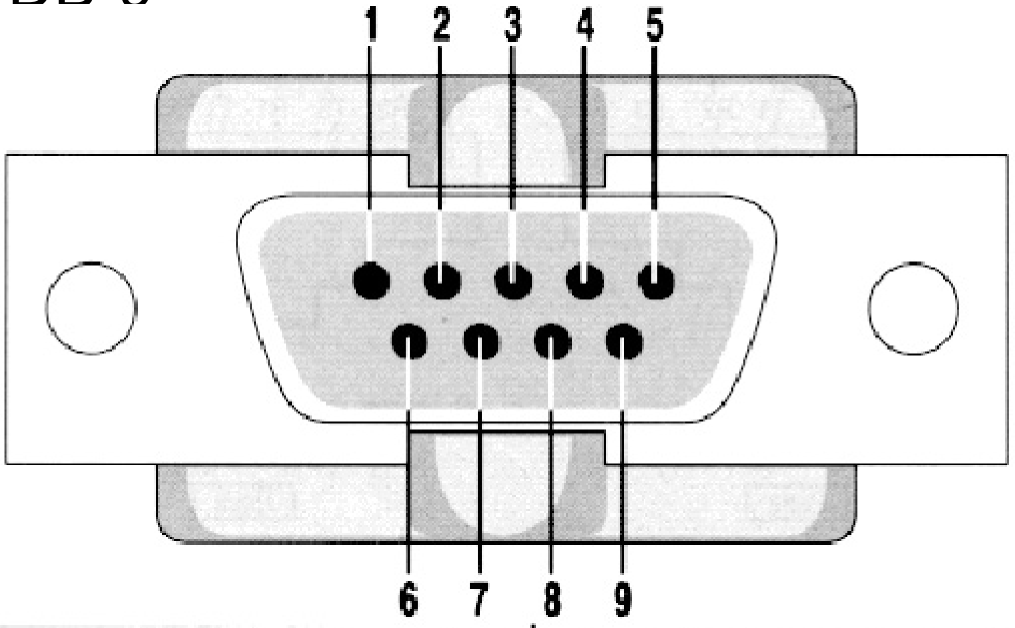
RS-232 Interface Standard, another example



- DTR (Data terminal ready): indication from the DTE that it is on

RS-232 Interface Standard

- DB-25 connector is described in the book; let's take a look at DB-9



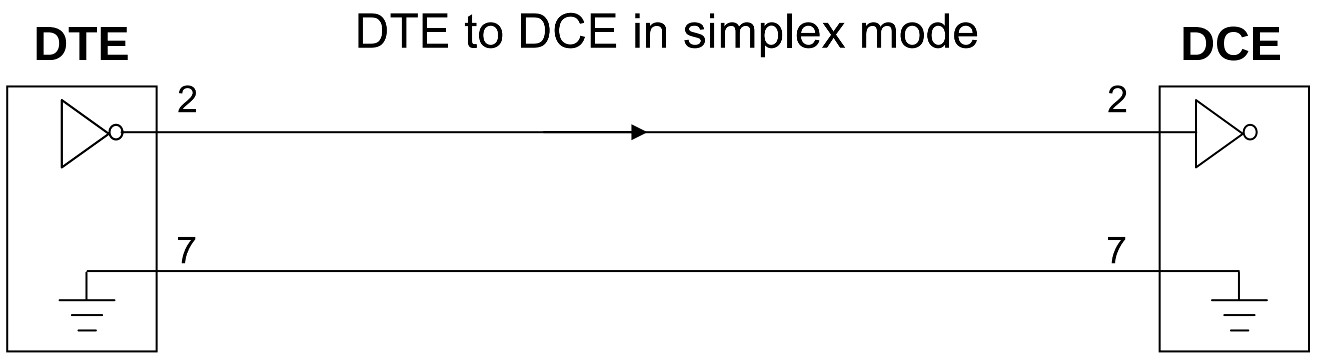
Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

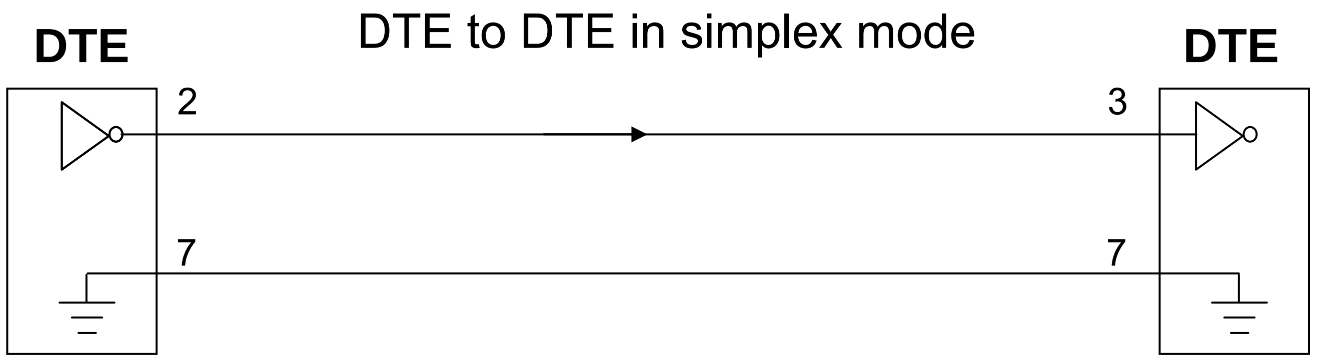
RS-232 Interface Standard

Example: 9 to 25 pin cable layout for asynchronous data

Description	Signal	9-pin DTE	25-pin DCE	Source DTE or DEC
Carrier Detect	CD	1	8	from Modem
Receive Data	RD	2	3	from Modem
Transmit Data	TD	3	2	from Terminal/Computer
Data Terminal Ready	DTR	4	20	from Terminal/Computer
Signal Ground	SG	5	7	from Modem
Data Set Ready	DSR	6	6	from Modem
Request to Send	RTS	7	4	from Terminal/Computer
Clear to Send	CTS	8	5	from Modem
Ring Indicator	RI	9	22	from Modem

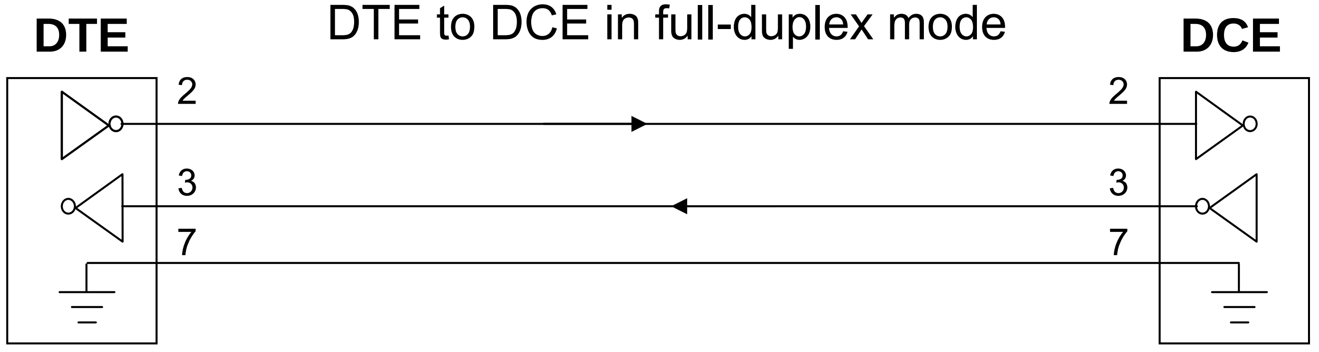
The Minimal RS-232 Function



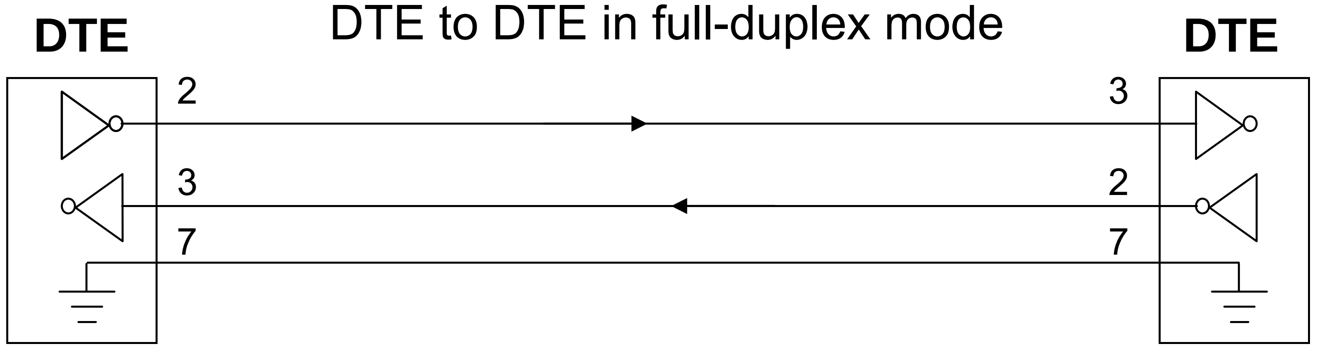


The Minimal RS-232 Function

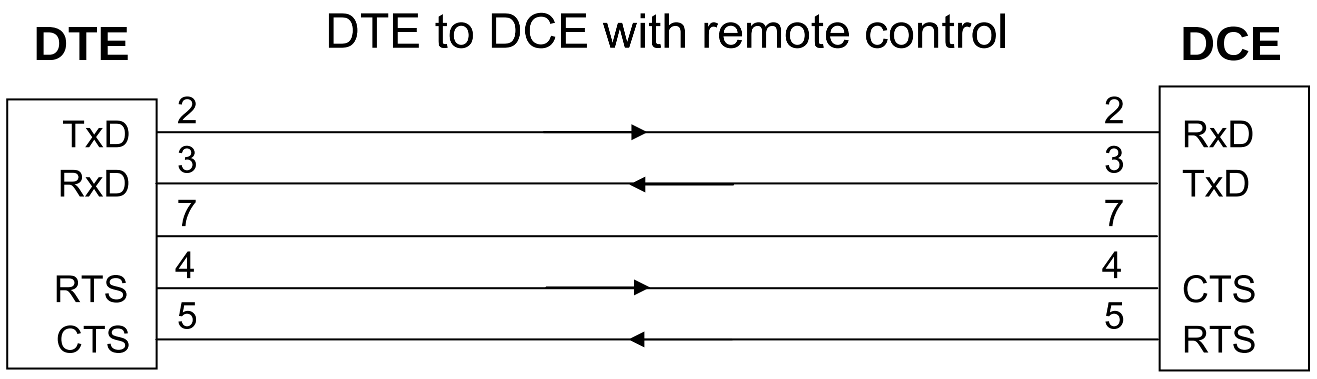
DTE to DCE in full-duplex mode

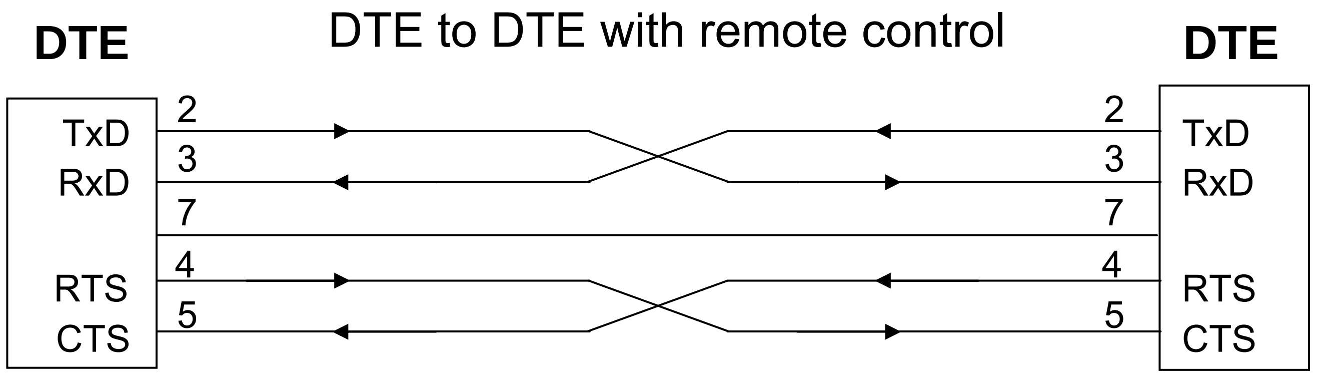


DTE to DTE in full-duplex mode

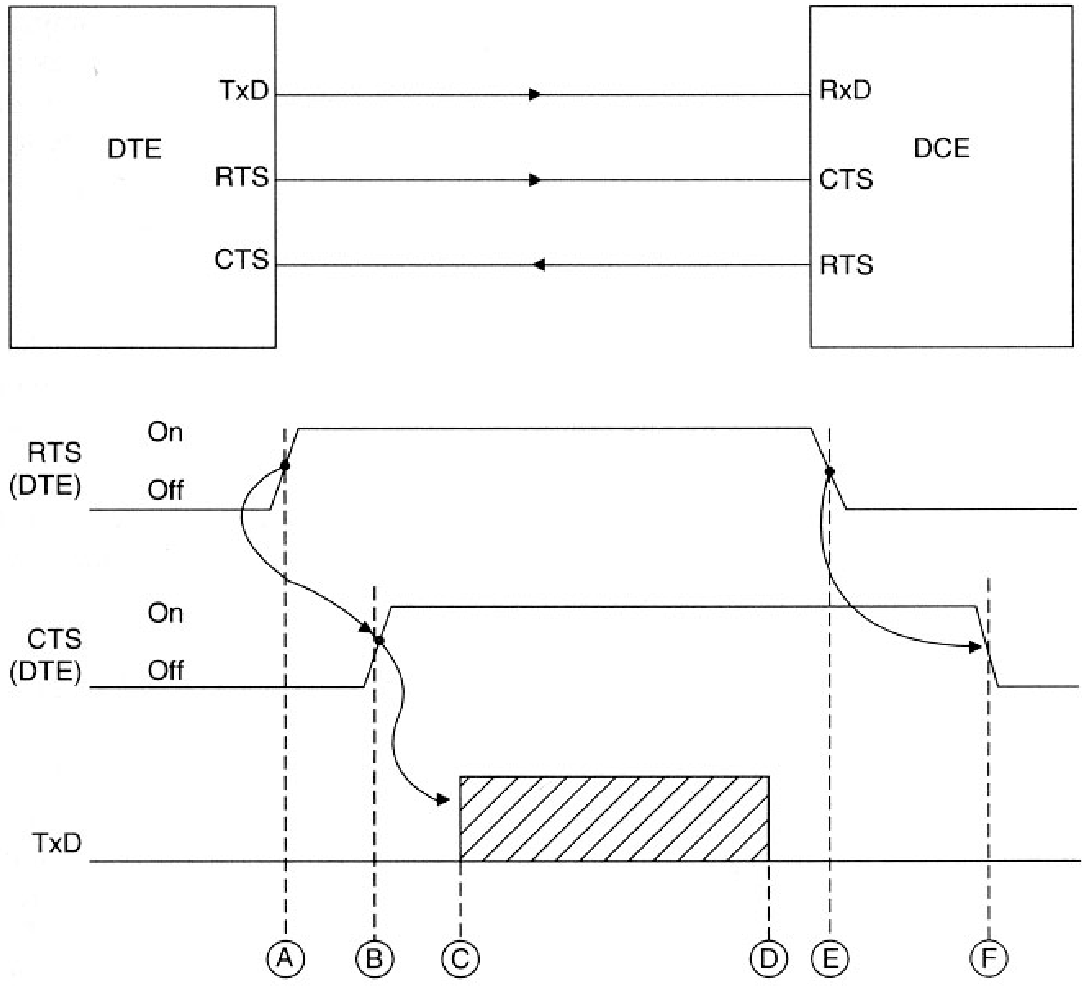


The Minimal RS-232 Function



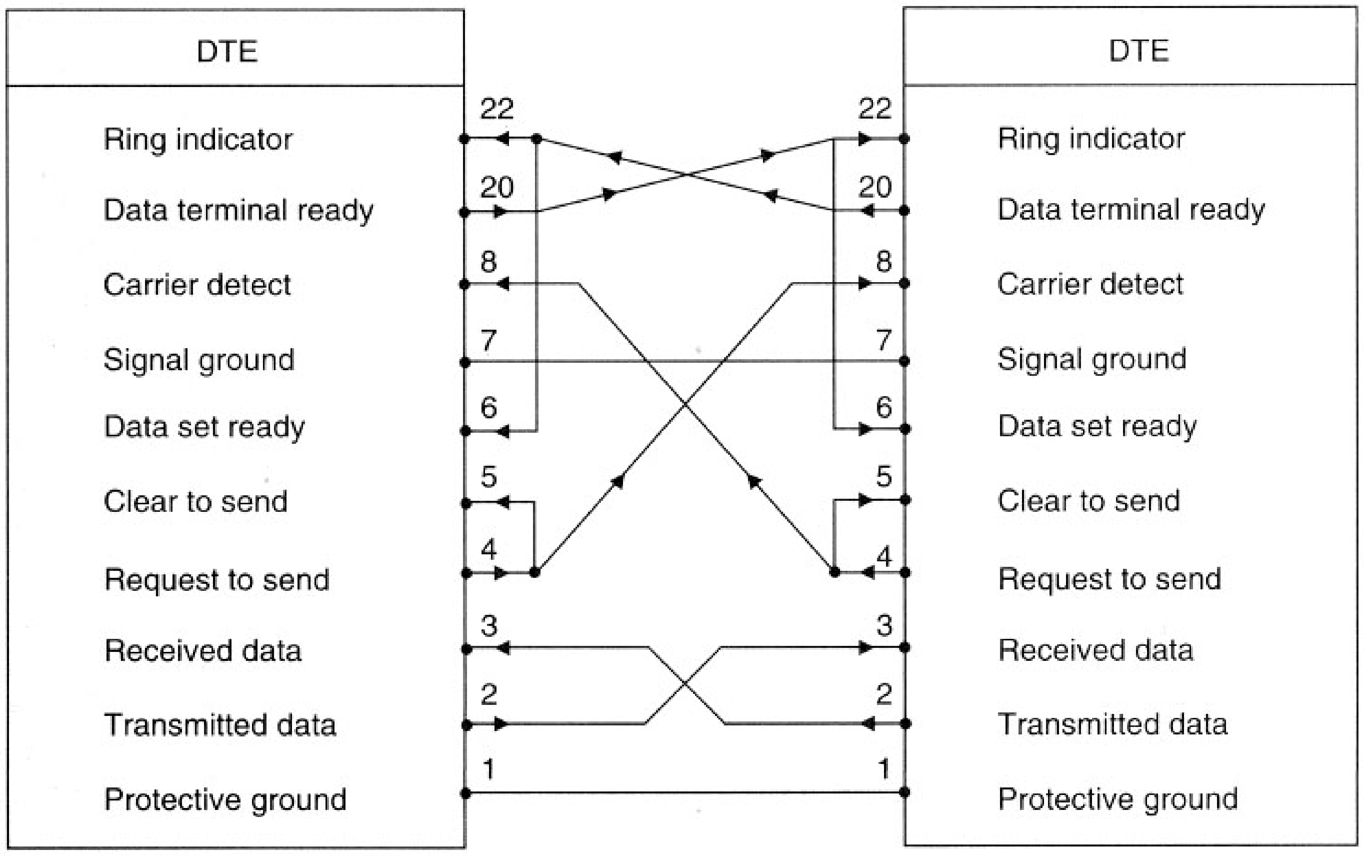


Handshaking Between RTS and CTS



Null Modem

- Null-modem simulates a DTE-DCE-DCE-DTE circuit



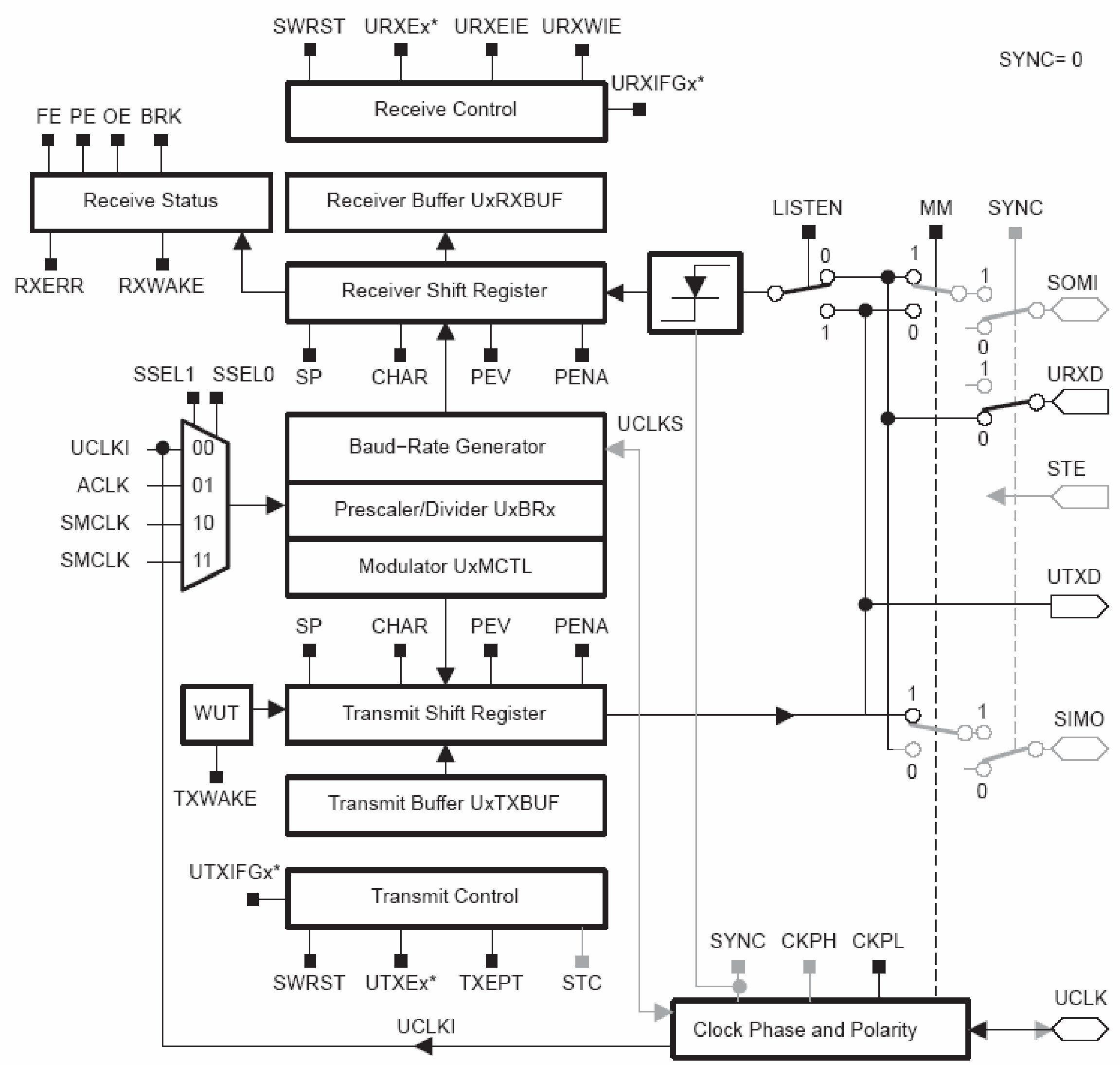
USART Peripheral Interface

- Universal Synchronous/Asynchronous Receive/Transmit (USART) peripheral interface supports two modes
  - Asynchronous UART mode (User manual, Ch. 13)
  - Synchronous Peripheral Interface, SPI mode (User manual, Ch. 14)
- UART mode:
  - Transmit/receive characters at a bit rate asynchronous to another device
  - Connects to an external system via two external pins URXD and UTXD (P3.4, P3.5)
  - Timing is based on selected baud rate (both transmit and receive use the same baud rate)

UART Features

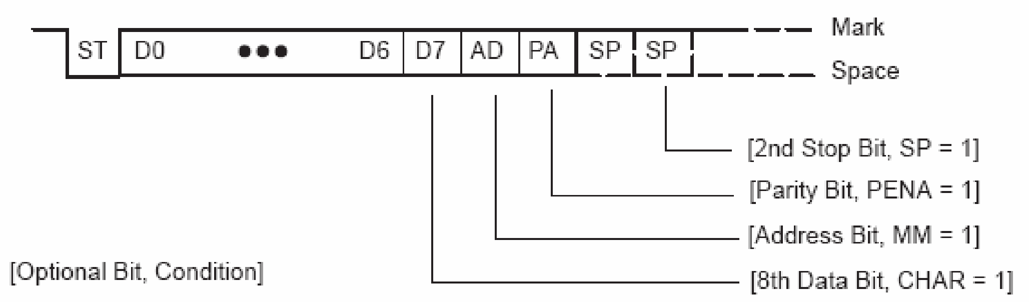
- 7- or 8-bit data width; odd, even, or non-parity
- Independent transmit and receive shift reg.
- Separate transmit and receive buffer registers
- LSB-first data transmit and receive
- Built-in idle-line and address-bit communication protocols for multiprocessor systems
- Receiver start-edge detection for auto-wake up from LPMx modes
- Programmable baud rate with modulation for fractional baud rate support
- Status flags for error detection
- Independent interrupt capability for transmit and receive

USART Block Diagram: UART mode

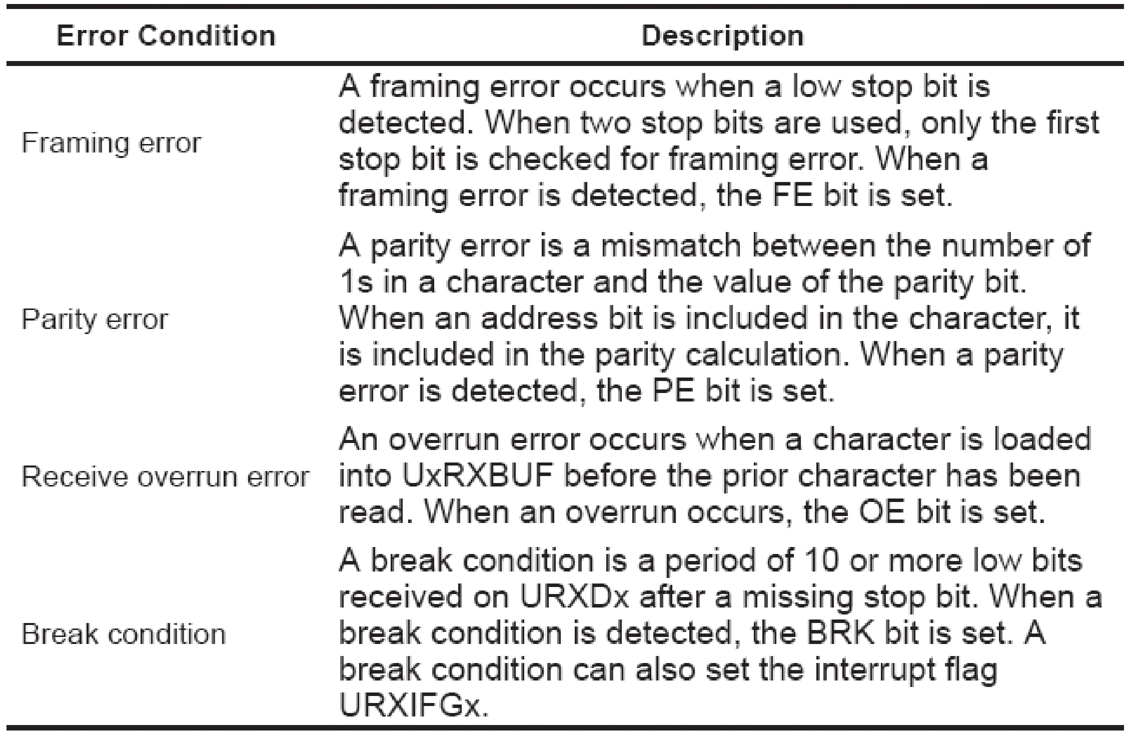


Initialization Sequence & Character Format

- Initialization Sequence
  - *1) Set SWRST (BIS.B #SWRST,&UxCTL)*
  - *2) Initialize all USART registers with SWRST = 1 (including UxCTL)*
  - *3) Enable USART module via the MEx SFRs (URXEx and/or UTXEx)*
  - *4) Clear SWRST via software (BIC.B #SWRST,&UxCTL)*
  - *5) Enable interrupts (optional) via the IEx SFRs (URXIEx and/or UTXIEx)*
  - *Note: Failure to follow this process may result in unpredictable USART behavior.*
- Character format



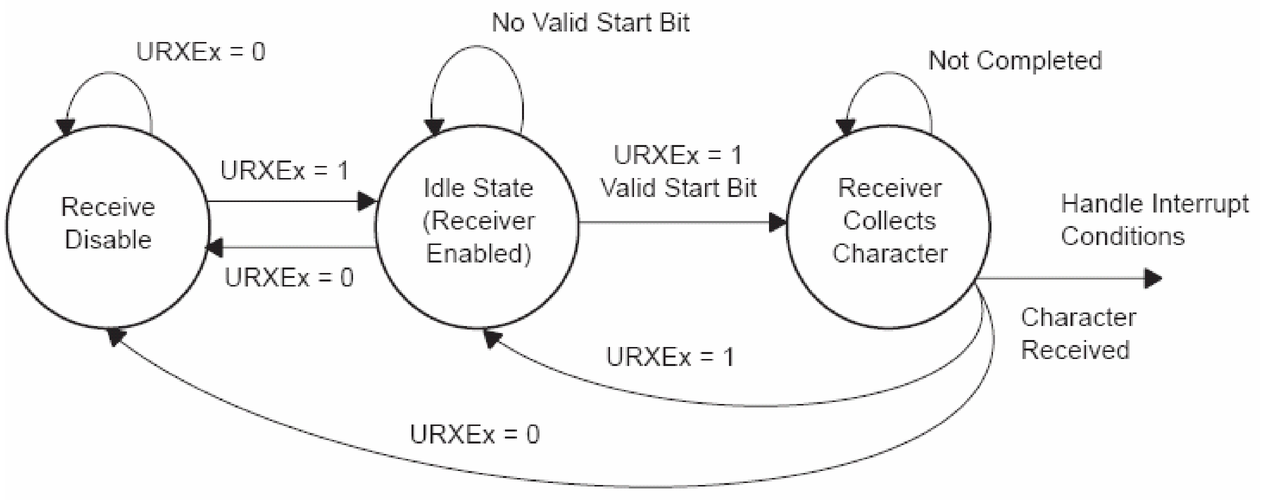
Automatic Error Detection



Error Condition	Description
Framing error	A framing error occurs when a low stop bit is detected. When two stop bits are used, only the first stop bit is checked for framing error. When a framing error is detected, the FE bit is set.
Parity error	A parity error is a mismatch between the number of 1s in a character and the value of the parity bit. When an address bit is included in the character, it is included in the parity calculation. When a parity error is detected, the PE bit is set.
Receive overrun error	An overrun error occurs when a character is loaded into UxRXBUF before the prior character has been read. When an overrun occurs, the OE bit is set.
Break condition	A break condition is a period of 10 or more low bits received on URXDx after a missing stop bit. When a break condition is detected, the BRK bit is set. A break condition can also set the interrupt flag URXIFGx.

UART Receive Enable

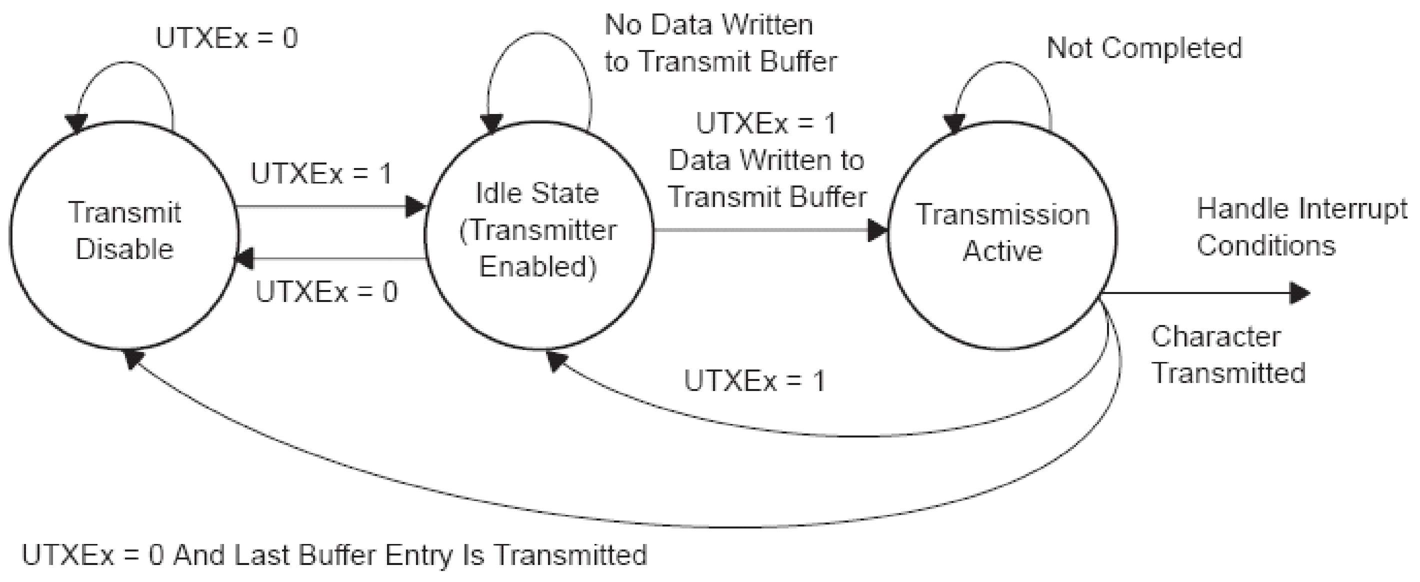
- The receive enable bit, URXEx, enables or disables data reception on URXDx
- Disabling the USART receiver stops the receive operation following completion of any character currently being received or immediately if no receive operation is active
- The receive-data buffer, UxRXBUF, contains the character moved from the RX shift register after the character is received



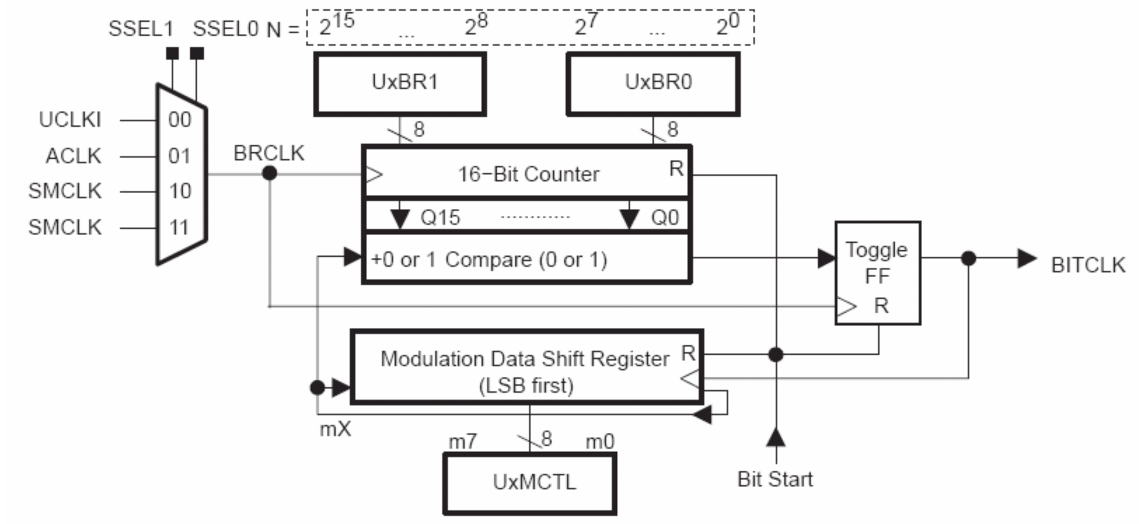
UART Transmit Enable

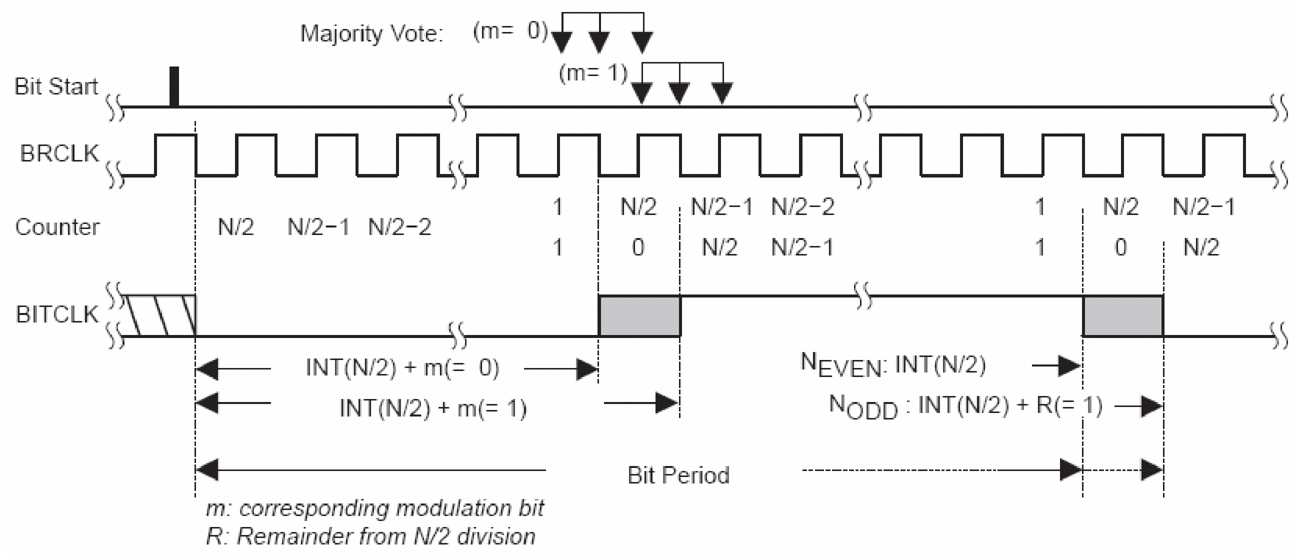
- When UTXEx is set (UTXEx=1), the UART transmitter is enabled
  - Transmission is initiated by writing data to UxTXBUF
  - Data is then moved to the transmit shift register (TX shift) on the next BITCLK after the TX shift register is empty, and transmission begins
  - Data should not be written to UxTXBUF unless it is ready for new data indicated by UTXIFGx = 1. Violation can result in an erroneous transmission if data in UxTXBUF is modified as it is being moved into the TX shift register.
- When the UTXEx bit is reset the transmitter is stopped
  - Any data in UxTXBUF and any active transmission prior to clearing UTXEx will continue until all data transmission is completed
  - It is recommended to disable transmitter (UTXEx = 0) only after completion of any active transmission. This is indicated by a set transmitter empty bit (TXEPT = 1).
  - Any data written to UxTXBUF while the transmitter is disabled will be held in the buffer but won’t be moved to the TX shift register. Once UTXEx=1, the data in is immediately loaded into the TX shift and character transmission resumes

UART Transmit Enable: State Diagram


UTXEx = 0
No Data Written to Transmit Buffer
Not Completed
UTXEx = 1
UTXEx = 1 Data Written to Transmit Buffer
Transmit Disable
Idle State (Transmitter Enabled)
Transmission Active
Handle Interrupt Conditions
UTXEx = 0
Character Transmitted
UTXEx = 1
UTXEx = 0 And Last Buffer Entry Is Transmitted

UART Baud Rate Generation





USART Interrupt Vectors

- The USART has one interrupt vector for transmission and one interrupt vector for reception
- Transmit:
  - The UTXIFGx interrupt flag is set by the transmitter to indicate that UxTXBUF is ready to accept another character. An interrupt request is generated if UTXIEx and GIE are also set. UTXIFGx is automatically reset if the interrupt request is serviced or if a character is written to UxTXBUF.
  - UTXIFGx is set after a PUC or when SWRST = 1. UTXIEx is reset after a PUC or when SWRST = 1.
- Receive:
  - The URXIFGx interrupt flag is set each time a character is received and loaded into UxRXBUF. An interrupt request is generated if URXIEx and GIE are also set. URXIFGx and URXIEx are reset by a system reset PUC signal or when SWRST = 1. URXIFGx is automatically reset if the pending interrupt is served (when URXSE = 0) or when UxRXBUF is read.

Control Registers

Table 13−3. USART0 Control and Status Registers

Register	Short Form	Register Type	Address	Initial State
USART control register	U0CTL	Read/write	070h	001h with PUC
Transmit control register	U0TCTL	Read/write	071h	001h with PUC
Receive control register	U0RCTL	Read/write	072h	000h with PUC
Modulation control register	U0MCTL	Read/write	073h	Unchanged
Baud rate control register 0	U0BR0	Read/write	074h	Unchanged
Baud rate control register 1	U0BR1	Read/write	075h	Unchanged
Receive buffer register	U0RXBUF	Read	076h	Unchanged
Transmit buffer register	U0TXBUF	Read/write	077h	Unchanged
SFR module enable register 1†	ME1	Read/write	004h	000h with PUC
SFR interrupt enable register 1†	IE1	Read/write	000h	000h with PUC
SFR interrupt flag register 1†	IFG1	Read/write	002h	082h with PUC

Does not apply to ’12xx devices. Refer to the register definitions for registers and bit positions for these devices.

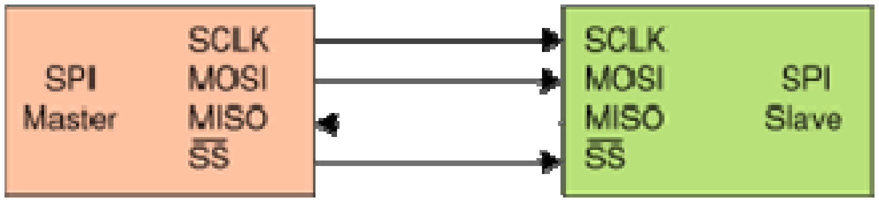
C Examples, UART 2400

```c
//******************************************************************************
//  MSP-FET430P140 Demo - USART1 UART 2400 Ultra-low Power Echo ISR, 32kHz ACLK
//
//  Description; Echo a received character, RX ISR used. In the Mainloop UART1
//  is made ready to receive one character with interrupt active. The Mainloop
//  waits in LPM3. The UART1 ISR forces the Mainloop to exit LPM3 after
//  receiving one character which echo's back the received character.
//  ACLK = UCLK1 = LFXT1 = 32768, MCLK = SMCLK = DCO~ 800k
//  Baud rate divider with 32768hz XTAL @2400 = 32768Hz/2400 = 13.65 (000Dh)
//  //*An external watch crystal is required on XIN XOUT for ACLK*//
//
//                MSP430F149
//            -----------------
//        /|\|              XIN|-
//         | |                 | 32kHz
//         --|RST          XOUT|-
//           |                 |
//           |             P3.6|----------->
//           |                 | 2400 - 8N1
//           |             P3.7|<-----------
//
//
//  M. Buccini
//  Texas Instruments, Inc
//  October 2003
//  Built with IAR Embedded Workbench Version: 1.26B
//  January 2004
//  Updated for IAR Embedded Workbench Version: 2.21B
//******************************************************************************
```

```c
#include <msp430x14x.h>

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;       // Stop WDT
  P3SEL |= 0xC0; // P3.6,7 = USART1 option select
  ME2 |= UTXE1 + URXE1; // Enable USART1 TXD/RXD
  UCTL1 |= CHAR;     // 8-bit character
  UTCTL1 |= SSEL0;   // UCLK = ACLK
  UBR01 = 0x0D;      // 32k/2400 - 13.65
  UBR11 = 0x00;
  UMCTL1 = 0x6B;     // Modulation
  UCTL1 &= ~SWRST;   // Initialize USART state
     machine
  IE2 |= URXIE1;     // Enable USART1 RX interrupt

// Mainloop
  for (;;)
  {
  _BIS_SR(LPM3_bits + GIE);   // Enter LPM3
     w/interrupt
  while (!(IFG2 & UTXIFG1));  // USART1 TX buffer
     ready?
  TXBUF1 = RXBUF1;            // RXBUF1 to TXBUF1
  }
}
// UART1 RX ISR will for exit from LPM3 in
     Mainloop
interrupt[UART1RX_VECTOR] void usart1_rx (void)
{
  _BIC_SR_IRQ(LPM3_bits);// Clear LPM3 bits from
     0(SR)
}
```

Serial Peripheral Interface

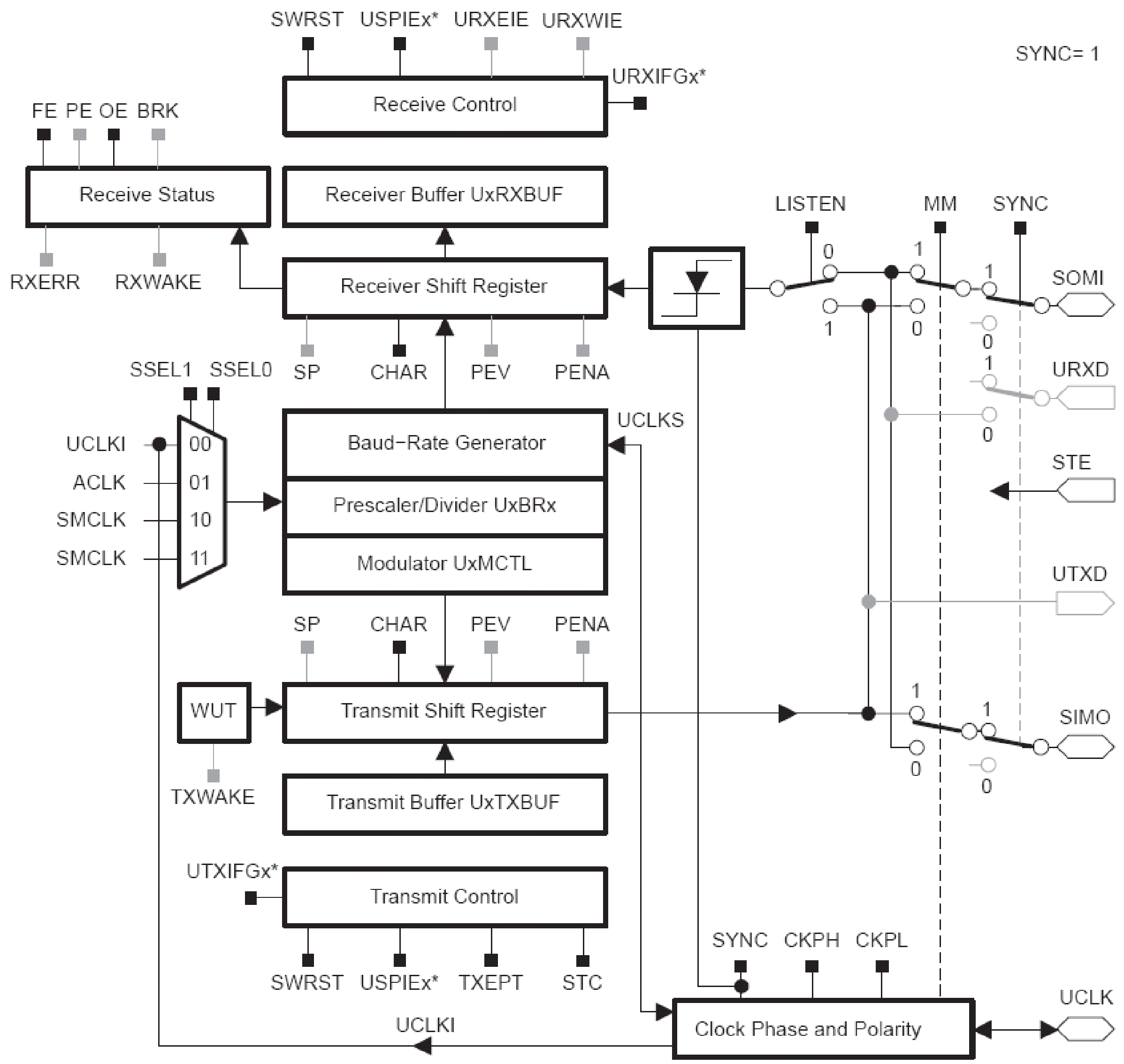
- Serial Peripheral Interface – SPI
  - It is a synchronous serial data link standard named by Motorola that operates in full duplex mode
  - Devices communicate in master/slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select (chip select) lines.
- The SPI bus specifies four logic signals.
  - SCLK — Serial Clock (output from master)
  - MOSI/SIMO — Master Output, Slave Input (output from master)
  - MISO/SOMI — Master Input, Slave Output (output from slave)
  - SS — Slave Select (active low; output from master)



SPI Mode: Signal Definition

- SIMO Slave in, master out
  - Master mode: SIMO is the data output line.
  - Slave mode: SIMO is the data input line.
- SOMI Slave out, master in
  - Master mode: SOMI is the data input line.
  - Slave mode: SOMI is the data output line.
- UCLK USART SPI clock
  - Master mode: UCLK is an output.
  - Slave mode: UCLK is an input.
- STE Slave transmit enable. Used in 4-pin mode to allow multiple masters on a single bus. Not used in 3-pin mode.
  - 4-Pin master mode:
    - When STE is high, SIMO and UCLK operate normally.
    - When STE is low, SIMO and UCLK are set to the input direction.
  - 4-pin slave mode:
    - When STE is high, RX/TX operation of the slave is disabled and SOMI is forced to the input direction.
    - When STE is low, RX/TX operation of the slave is enabled and SOMI operates normally.

USART: SPI Mode

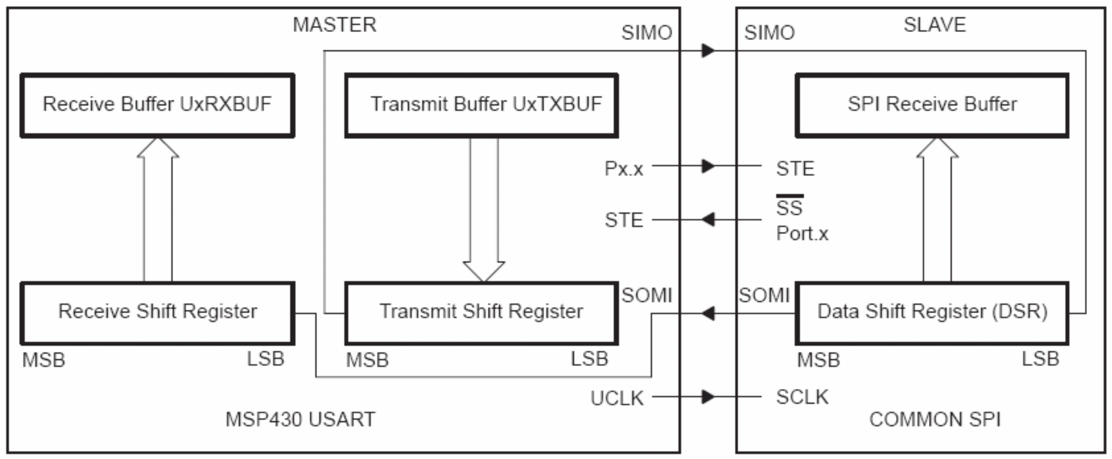


SPI Mode: Initialization Sequence

- 1) Set SWRST (BIS.B #SWRST,&UxCTL)
- 2) Initialize all USART registers with SWRST=1 (including UxCTL)
- 3) Enable USART module via the MEx SFRs (USPIEx)
- 4) Clear SWRST via software (BIC.B #SWRST,&UxCTL)
- 5) Enable interrupts (optional) via the IEx SFRs (URXIEx and/or UTXIEx)
- Note: Failure to follow this process may result in unpredictable USART behavior.

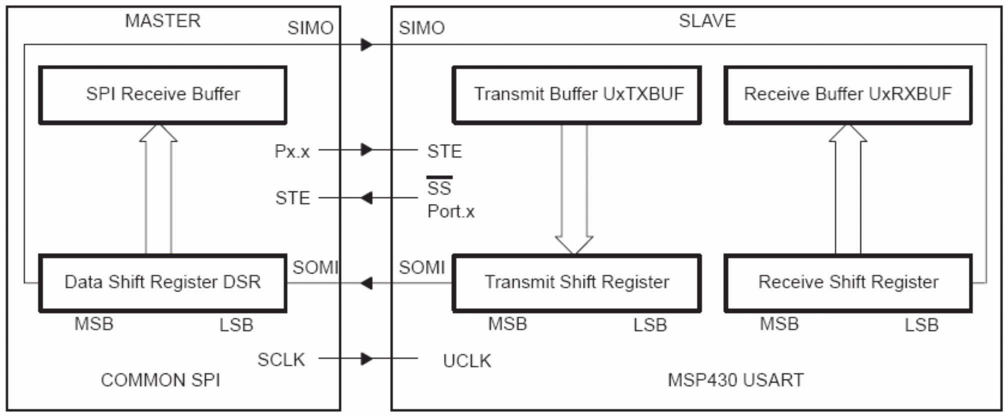
SPI Master Mode

- The USART initiates data transfer when data is moved to the UxTXBUF. The UxTXBUF data is moved to the TX shift register when the TX shift register is empty, initiating data transfer on SIMO starting with the MSB. Data on SOMI is shifted into the receive shift register on the opposite clock edge, starting with the most-significant bit.
- When the character is received, the receive data is moved from the RX shift register to the UxRXBUF and the receive interrupt flag, URXIFGx, is set, indicating the RX/TX operation is complete.
- A set transmit interrupt flag, UTXIFGx, indicates that data has moved from UxTXBUF to the TX shift register and UxTXBUF is ready for new data. It does not indicate RX/TX completion.
- To receive data into the USART in master mode, data must be written to UxTXBUF because receive and transmit operations operate concurrently.
- In 4-pin master mode, STE is used to prevent conflicts with another master. The master operates normally when STE is high. When STE is low:
  - SIMO and UCLK are set to inputs and no longer drive the bus
  - The error bit FE is set indicating a communication integrity violation to be handled by the user



SPI Slave Mode

- UCLK is used as the input for the SPI clock and must be supplied by the external master. The data-transfer rate is determined by this clock and not by the internal baud rate generator.
- Data written to UxTXBUF and moved to the TX shift register before the start of UCLK is transmitted on SOMI. Data on SIMO is shifted into the receive shift register on the opposite edge of UCLK and moved to UxRXBUF when the set number of bits are received
- When data is moved from the RX shift register to UxRXBUF, the URXIFGx interrupt flag is set, indicating that data has been received. The overrun error bit, OE, is set when the previously received data is not read from UxRXBUF before new data is moved to UxRXBUF.
- In 4-pin slave mode, STE is used by the slave to enable the transmit and receive operations and is provided by the SPI master. When STE is low, the slave operates normally. When STE is high:
  - Any receive operation in progress on SIMO is halted
  - SOMI is set to the input direction



C Examples, SPI Full-Duplex

```c
//******************************************************************************
//  MSP-FET430P140 Demo - USART0, SPI Full-Duplex 3-Wire Slave P1.x Exchange
//
//  Description: SPI Master communicates at fast as possible, full-duplex with
//  SPI Slave using 3-wire mode. The level on P1.4/5 is TX'ed and RX'ed to P1.0
//  and P1.1. Master will pulse slave Reset on init to insure synch start.
//  Slave normal mode is LPM4.
//  ACLK = n/a, MCLK = SMCLK = DCO ~ 800kHz, ULCK = external
//
//             fet140_slav0               fet140_mstr0
//             MSP430F169 Slave           MSP430F169 Master
//             -----------------          -----------------
//            |              XIN|-    /|\|              XIN|-
//            |                 |      | |                 |
//            |             XOUT|-     --|RST          XOUT|-
//            |                 | /|\    |                 |
//            |              RST|--+<----|P3.0             |
//      LED <-|P1.0             |        |             P1.4|<-
//      LED <-|P1.1             |        |             P1.5|<-
//          ->|P1.4             |        |             P1.0|-> LED
//          ->|P1.5             |        |             P1.1|-> LED
//            |       SIMO0/P3.1|<-------|P3.1             |
//            |       SOMI0/P3.2|------->|P3.2             |
//            |        UCLK/P3.3|<-------|P3.3             |
//
//  M. Buccini
//  Texas Instruments Inc.
//  Feb 2005
//  Built with IAR Embedded Workbench Version: 3.21A
//******************************************************************************
```

```c
#include  <msp430x14x.h>

void main(void)
{

  WDTCTL = WDTPW + WDTHOLD;  // Stop watchdog
  P1OUT = 0x00; // P1.0 setup for LED output
  P1DIR |= 0x03;
  P3SEL |= 0x0E;    // P3.1,2,3 SPI option select
  U0CTL = CHAR + SYNC + SWRST;   // 8-bit, SPI
  U0TCTL = CKPL + STC;       // Polarity, 3-wire
  U0BR0 = 0x02;               // SPICLK = SMCLK/2
  U0BR1 = 0x00;
  U0MCTL = 0x00;
  ME1 |= USPIE0;         // Module enable
  U0CTL &= ~SWRST;         // SPI enable
  IE1 |= URXIE0 + UTXIE0; // RX and TX int. enable
  _BIS_SR(LPM4_bits + GIE); // Enter LPM4 w/ int.
}
#pragma vector=USART0RX_VECTOR
__interrupt void SPI0_rx (void) {
  P1OUT = RXBUF0; // RXBUF0 to TXBUF0
}
#pragma vector=USART0TX_VECTOR
__interrupt void SPI0_tx (void) {
  unsigned int i;

  i = P1IN;
  i = i >> 4;
  TXBUF0 = i;     // Transmit character
}
```