CPE/EE 421 Microcomputers: 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.
  - A POR signal
- 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

Brownout Reset

Figure 2-2 POR Timing

Figure 2-3 Brownout Timing
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

(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

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 SCGO, 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**

- **Interrupt Vectors (offset from 0xFFE0)**

```c
#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  /* 0xFFFCA Non-maskable */
#define RESET_VECTOR 15 * 2  /* 0xFFFEC Reset [Highest Pr.] */
```

---

**Interrupt Service Routines**

- **Interrupt Service Routine declaration**

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

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

**EXAMPLE**

```c
interrupt[TIMERA0_VECTOR] void myISR (Void);
interrupt[TIMERA0_VECTOR] void myISR (Void)
{
  // ISR code
}
```

---

**Operating Modes**

(to be discussed later)

<table>
<thead>
<tr>
<th>SC/G</th>
<th>SC/0</th>
<th>CPUC</th>
<th>Modes</th>
<th>CP and Clock States</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Active</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>LPM</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>LPM</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>LPM</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>LPM</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>LPM</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>LPM</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>LPM</td>
</tr>
</tbody>
</table>

**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, internally 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
  - MCLK main system clock
  - SMCLK sub main system clock

Basic Clock System
- 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

Basic operation
- After POC (Power Up Clear)
  - MCLK and SCLK 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 used 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
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

<table>
<thead>
<tr>
<th>Register State</th>
<th>Short Form</th>
<th>Register Type</th>
<th>Address</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCO control</td>
<td>DCOCTL</td>
<td>Read/write</td>
<td>056h</td>
<td>060h</td>
</tr>
<tr>
<td>Basic clock</td>
<td>BCSCTL1</td>
<td>Read/write</td>
<td>057h</td>
<td>084h</td>
</tr>
<tr>
<td>system control</td>
<td>BCSCTL2</td>
<td>Read/write</td>
<td>058h</td>
<td>reset</td>
</tr>
</tbody>
</table>

### Basic Clock Module - Control Registers

**DCOCTL**

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

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

<table>
<thead>
<tr>
<th>DCOCTL</th>
<th>DCO.2</th>
<th>DCO.1</th>
<th>DCO.0</th>
<th>MOD.4</th>
<th>MOD.3</th>
<th>MOD.2</th>
<th>MOD.1</th>
<th>MOD.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>056H</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**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.

**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.

<table>
<thead>
<tr>
<th>BCSCTL1</th>
<th>XT2Off</th>
<th>XT5V</th>
<th>DIVA.1</th>
<th>DIVA.0</th>
<th>XT5V</th>
<th>Rsel.2</th>
<th>Rsel.1</th>
<th>Rsel.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>057h</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**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.

**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

### BCSCTL2

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

**Bit6, XT5V**: XT5V should always be reset.

**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.

**Bit8, DCO**: The DCO bit selects the resistor for injecting current into the dc generator.

Based on this current, the oscillator operates if activated.

**Bit9, DCOR**: The DCOR bit selects the resistor for injecting current into the dc generator.

Based on this current, the oscillator operates if activated.

**Bit10, DCOR**: The internal resistor is selected in eight different steps.

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

**Bit11, XT5V**: XT5V should always be reset.

**Bit12, DCOR**: The selected source for SMCLK is divided by:

- **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 MCCLK 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 MCCLK:
SELM.0 .. SELM.1
SELM = 0: Use the DCOCLK
SELM = 1: Use the XT2CLK (x13x and x14x devices)
or
Use the LFXT1CLK (x11x(1) devices)
SELM = 3: Use the LFXT1CLK

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.

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
- Comparator 2 Hardware 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
```

Fail Safe Operation

- Basic module incorporates an oscillator-fault detection fail-safe feature.
- The oscillator fault detector is an analog circuit that monitors the LPXT1CLK (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 new clock source

Basic Clock Module - Examples

- How to select the Crystal Clock
  ```
  void selectclock(void)
  {
    IFG2=0; /* reset interrupt flag register 1 */
    IFG1=0; /* reset interrupt flag register 2 */
    BCSCTL1|=XTS; /* attach HF crystal (4MHz) to XIN/XOUT */
    do {
      /* wait in loop until crystal is stable */
      IFG1&=~OFIFG;
      Delay();
    } while(OFIFG&IFG1);
    IFG1&=~OFIFG; /* Reset osc. fault flag again */
  }
  ```

- How to select a clock for MCLK
  ```
  BCSCTL2|=SELM0+SELM1; /* Then set MCLK same as LFXT1CLK*/
  TACTL=TASSEL0+TACLR+ID1; /* USE ACLK/4 AS TIMER_A INPUT CLOCK */
  ```

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.
Watchdog Timer-General

The primary function of the watchdog timer (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 ultra-low-power using the hold mode

Watchdog/Timer two functions:
- SW Watchdog Mode
- Interval Timer Mode

Watchdog Timer-Registers

- **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 with .w suffix. In both operating modes (watchdog or timer), it is only possible to write to WDTCNT using the correct password.

- **Watchdog Timer Control Register (WDTCTL)**

<table>
<thead>
<tr>
<th>SSEL</th>
<th>TMSEL</th>
<th>Bit 1</th>
<th>Bit 2</th>
<th>Interval [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.064 ACLK × 2^3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.9 ACLK × 2^1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8 ACLK × 2^3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>16.0 SMCLK × 2^15</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>250 ACLK × 2^7</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1000 ACLK × 2^8</td>
</tr>
</tbody>
</table>

Bit 1: This bit is latched and selects one of four taps from the WDTCNT, as described in the following table. Assuming f crystal = 32,768 Hz and f System = 1 MHz, the following intervals are possible:

- **NDWM = 0**: The clock multiplexer and counter are stopped. The WDTIE bit is used to disable the interrupt from the Watchdog Timer when it is being used in interval-timer mode. The WDTIFG flag is reset automatically when the interrupt is serviced.
- **NDWM = 1**: Bit 5 is held low, the internal signal is active (level sensitive). The WDTIFG 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 signal was caused by a different source. See chapter 3 for more details on the PUC and POR signals.
- **NDWM = 2**: A rising edge triggers an NMI interrupt. 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. The WDTIFG flag is reset automatically when the interrupt is serviced.
- **NDWM = 3**: A falling edge triggers an NMI interrupt. 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. The WDTIFG flag is reset automatically when the interrupt is serviced.

Watchdog Timer-Interrupt Function

The Watchdog Timer (WDT) uses two bits in the SFRs for interrupt control. The WDT interrupt enable (WDTIE) (located in IE1.0, initial state is reset)

<table>
<thead>
<tr>
<th>WDTCTL</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>WDTIE = 0; WDTIFG = 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>WDTIE = 1; WDTIFG = 0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>WDTIE = 0; WDTIFG = 1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>WDTIE = 1; WDTIFG = 1</td>
</tr>
</tbody>
</table>

Bit 0: This bit selects the clock source for WDTCNT. The clock multiplexer and counter are stopped. The WDTIFG 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 signal was caused by a different source. See chapter 3 for more details on the PUC and POR signals.

- **WDTIE = 0**: WDTIE is set 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 signal was caused by a different source. See chapter 3 for more details on the PUC and POR signals.

- **WDTIE = 1**: WDTIE is used to enable or disable the interrupt from the Watchdog Timer when it is being used in interval-timer mode. 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
  ```c
  /* 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
  ```c
  WDTCTL=WDTPW + WDTHOLD ;  // stop watchdog timer
  ```
- Assembly programming
  ```assembly
  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
  ```

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?

\[ P = A C V^2 f + z A V I_{\text{short}} f + V I_{\text{leak}} \]
Short-circuit Power Consumption

Finite slope of the input signal causes a direct current path between VDD 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

\[ P = \frac{ACV^2}{t} + \tau AVI_{\text{short}} f + V_{\text{leak}} \]

Reduce the supply voltage, V

\[ I_{\text{leak}} \propto \exp(-\frac{qV}{kT}) \]

Reduce threshold Vt

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
- \( I_{\text{av}} = \sum I(t) \times \frac{dt}{T} \)
- In most cases \( I_{\text{av}} = \Sigma I(t) \times \frac{dt}{T} \)

Low-Power Concept: Basic Conditions for Burst Mode

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

\[ I_{\text{avg}} = I_{\text{measure}} + I_{\text{calculate}} + I_{\text{RTC}} + I_{\text{display}} \]

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.

The sleep current dominates the current consumption!
### 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 = 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 VC)
- Asynchronous logic
  - completion signals
  - leading
- Architecture
  - Parallelism (increased area and wiring)
  - Speculation (branch prediction)
  - Memory systems
    - Memory accesses (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, SCG1, and SCG0 bits on the stack. Then, when the SR contents are popped from the stack upon RETI, the operating mode will be active mode (AM).

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
  - 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 do-generator remains enabled
  - ACLK remains active
- Low power mode 3 (LPM3); SCG1=1, SCG0=1, OscOff=0, CPUOff=1:
  - CPU is disabled
  - MCLK is disabled
  - SMCLK is disabled
  - ACLK remains active
  - DCO oscillator is disabled
- Low power mode 4 (LPM4); SCG1=1, SCG0=0, OscOff=1, CPUOff=1:
  - CPU is disabled
  - MCLK is disabled
  - SMCLK is disabled
  - ACLK is disabled
  - DCO oscillator is disabled
  - DCO’s do-generator is disabled
  - Crystal oscillator is stopped

Operating Modes-Examples

The following example describes clearing low-power mode 0.

```assembly
; Clear low-power mode 0
LDR R1, =0x04000000  ; Load base address
LDR R2, [R1, #0]    ; Load contents
AND R2, R2, #0x00000001 ; AND with 1
STR R2, [R1, #0]    ; Store result

data
reset
```

The following example describes clearing low-power mode 3.

```assembly
; Clear low-power mode 3
LDR R1, =0x04000000  ; Load base address
LDR R2, [R1, #0]    ; Load contents
AND R2, R2, #0x00000003 ; AND with 3
STR R2, [R1, #0]    ; Store result

data
reset
```

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 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 operation halts 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 RAM/registers are unchanged. Wake up is possible by enabled interrupts coming from active peripherals or RST/NMI.

The following example describes clearing low-power mode 0.

```assembly
; Clear low-power mode 0
LDR R1, =0x04000000  ; Load base address
LDR R2, [R1, #0]    ; Load contents
AND R2, R2, #0x00000001 ; AND with 1
STR R2, [R1, #0]    ; Store result

data
reset
```

The following example describes clearing low-power mode 3.

```assembly
; Clear low-power mode 3
LDR R1, =0x04000000  ; Load base address
LDR R2, [R1, #0]    ; Load contents
AND R2, R2, #0x00000003 ; AND with 3
STR R2, [R1, #0]    ; Store result

data
reset
```
C – programming msp430x14x.h

#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 */

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?

Digital I/O

- 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

MSP430: Digital I/O

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

**Output Registers PnOUT**
- Each bit in each PnOUT register reflects the value of the output for 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

**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**

**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/compare; 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**
Timer_A Counting Modes

Stop/Halt Mode
Timer is halted with the next +CLK.

UP Mode
Timer counts between 0 and CCR0.

Continuous Mode
Timer continuously counts up.

UP/DOWN Mode
Timer counts between 0 and CCR0 and 0.

Example shows three independent HW event captures. CCRx “stamps” time of event. Continuous-Mode is ideal.
Digital Motor Control

C Examples, CCR0 Upmode ISR, TA_0

```c
#include <msp430x14x.h>

void main(void)
{
    // MSP-FET430P140 Demo - Timer_A Toggle P1.0, CCR0 upmode ISR, 32kHz ACLK
    // Description; Toggle P1.0 using software and TA_0 ISR.
    // n/a, MCLK = SMCLK = TACLK = DCO~ 800k used only during TA_ISR.
    CCR0 = 50000;     // Add Offset to CCR0
    CCTL0 = CCIE;     // CCR0 interrupt enabled
    P1DIR |= 0x01;    // P1.0 output
    WDTCTL = WDTPW + WDTHOLD; // Stop WDT
    _BIS_SR(LPM0_bits + GIE); // Enter LPM0 w/ interrupt
    CCR0 += 50000;      // Add Offset to CCR1
    break;
}
```

C Examples, CCR1 Contmode ISR, TA_1

```c
#include <msp430x14x.h>

void main(void)
{
    // MSP-FET430P140 Demo - Timer_A Toggle P1.0, CCR1 contmode ISR
    // Description; Toggle P1.0 using software and TA_1 ISR.
    // ACLK = n/a, MCLK = SMCLK = TACLK = DCO~ 800k used only during TA_ISR.
    CCR0 = 50000;     // Add Offset to CCR0
    CCTL0 = CCIE;     // CCR0 interrupt enabled
    P1DIR |= 0x01;    // P1.0 output
    WDTCTL = WDTPW + WDTHOLD; // Stop WDT
    _BIS_SR(LPM0_bits + GIE); // Enter LPM0 w/ interrupt
    CCR1 += 50000;      // Add Offset to CCR1
    break;
}
```

C Examples, PWM, TA1-2 upmode

```c
#include <msp430x14x.h>

void main(void)
{
    // MSP-FET430P140 Demo - Timer_A Toggle P1.0, CCR0 upmode ISR, 32kHz ACLK
    // Description; Toggle P1.0 using software and TA_0 ISR.
    // ACLK = n/a, MCLK = SMCLK = TACLK = DCO~ 800k used only during TA_ISR.
    CCR0 = 50000;     // Add Offset to CCR0
    CCTL0 = CCIE;     // CCR0 interrupt enabled
    P1DIR |= 0x01;    // P1.0 output
    WDTCTL = WDTPW + WDTHOLD; // Stop WDT
    _BIS_SR(LPM0_bits + GIE); // Enter LPM0 w/ interrupt
    CCR1 += 50000;      // Add Offset to CCR1
    break;
}
```

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**

- 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 — \( T + \delta T \) instead of \( T \))
  - \( T/2 > (2N+1)/2 \)
  - \( \delta T/2 < 1/(2N+1) \)
  - \( \delta T/T < 100/(2N+1) \) as a percentage.

**RS-232 Interface Standard**

- **Bi-polar**:
  - \(+3\) to \(+12\)V (ON, 0-state, or SPACE condition)
  - \(-3\) to \(-12\)V (OFF, 1-state, or MARK condition)
- Modern computers accept 0V as MARK.
- “Dead area” between \(-3\)V and \(3\)V 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).
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

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

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal</th>
<th>Description</th>
<th>Source: DTE or DEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Data Carrier Detect</td>
<td>CD</td>
<td>from Modem</td>
</tr>
<tr>
<td>2</td>
<td>Receive Data</td>
<td>RD</td>
<td>from Modem</td>
</tr>
<tr>
<td>3</td>
<td>Transmit Data</td>
<td>TD</td>
<td>from Terminal/Computer</td>
</tr>
<tr>
<td>4</td>
<td>Data Set Ready</td>
<td>DSR</td>
<td>from Modem</td>
</tr>
<tr>
<td>5</td>
<td>Request to Send</td>
<td>RTS</td>
<td>from Terminal/Computer</td>
</tr>
<tr>
<td>6</td>
<td>Clear to Send</td>
<td>CTS</td>
<td>from Modem</td>
</tr>
<tr>
<td>7</td>
<td>Ring Indicator</td>
<td>RI</td>
<td>from Modem</td>
</tr>
</tbody>
</table>

Example: 9 to 25 pin cable layout for asynchronous data

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

The Minimal RS-232 Function

- DTE to DCE in simplex mode
- DTE to DCE in full-duplex mode
- DTE to DTE in simplex mode
- DTE to DTE in full-duplex mode
The Minimal RS-232 Function

DTE to DCE with remote control

<table>
<thead>
<tr>
<th>DTE</th>
<th>DCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TxD</td>
<td>RxD</td>
</tr>
<tr>
<td>RdD</td>
<td>2</td>
</tr>
<tr>
<td>RdD</td>
<td>TxD</td>
</tr>
<tr>
<td>RTS</td>
<td>4</td>
</tr>
<tr>
<td>CTS</td>
<td>5</td>
</tr>
</tbody>
</table>

DTE to DTE with remote control

<table>
<thead>
<tr>
<th>DTE</th>
<th>DTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TxD</td>
<td>RxD</td>
</tr>
<tr>
<td>RdD</td>
<td>2</td>
</tr>
<tr>
<td>RdD</td>
<td>TxD</td>
</tr>
<tr>
<td>RTS</td>
<td>4</td>
</tr>
<tr>
<td>CTS</td>
<td>5</td>
</tr>
</tbody>
</table>

Handshaking Between RTS and CTS

Null Modem

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

Null Modem Block Diagram

<table>
<thead>
<tr>
<th>DTE</th>
<th>DTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TxD</td>
<td>RxD</td>
</tr>
<tr>
<td>RdD</td>
<td>2</td>
</tr>
<tr>
<td>RdD</td>
<td>TxD</td>
</tr>
<tr>
<td>RTS</td>
<td>4</td>
</tr>
<tr>
<td>CTS</td>
<td>5</td>
</tr>
</tbody>
</table>

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:
  - Transmits/receives 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)

USART Block Diagram: UART mode

<table>
<thead>
<tr>
<th>UART Features</th>
</tr>
</thead>
</table>

- 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

<table>
<thead>
<tr>
<th>UART Features</th>
</tr>
</thead>
</table>
Initialization Sequence & Character Format

- Initialization Sequence
  - Set SWRST bit
  - Initialize all USART registers with SWRST = 1
  - Enable USART module via the MEx SFRs (URXEx and/or UTXEx)
  - Clear SWRST via software (releases the USART for operation)
  - Optional: enable interrupts via IEx SFRs

- Character format

---

**Initialization Sequence**

1. Set SWRST bit
2. Initialize all USART registers with SWRST = 1
3. Enable USART module via the MEx SFRs (URXEx and/or UTXEx)
4. Clear SWRST via software (releases the USART for operation)
5. Optional: enable interrupts via IEx SFRs

---

**Character Format**

---

## C Examples, UART 2400

```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
  IE2 |= URXIE1; // Enable USART1 RX interrupt

  // Mainloop
  for (;;)
  {
    _BIS_SR(LPM3_bits + GIE); // Enter LPM3
    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 SR
}
```

---

**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.
- Baud rate divider with 32768hz XTAL @2400 = 32768Hz/2400 = 13.65 (000Dh)
- An external watch crystal is required on XIN XOUT for ACLK.

---

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

---