CPE/EE 421 Microcomputers
Instructor: Dr. Aleksandar Milenkovic
Lecture Note
S15

Course Administration

- Instructor: Aleksandar Milenkovic
  milenka@ece.uah.edu
  www.ece.uah.edu/~milenka
  EB 217-L
  Mon. 5:30 PM – 6:30 PM, Wed. 12:30 – 13:30 PM
- URL: http://www.ece.uah.edu/~milenka/cpe421-05F
- TA: Joel Wilder
- Labs: Lab #3 is on.
- Test I: Graded. Solutions are in scr/.
- Text: Microprocessor Systems Design: 68000 Hardware, Software, and Interfacing
- Review: M68K (Chapter 1; Chapter 2; Chapter 3), MSP430 (Introduction, Arch., Basic Clock System)
- Today: MSP430 WDT, Low Power Modes

Review: Basic Clock Systems-detail

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.

<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 060h</td>
<td></td>
</tr>
<tr>
<td>Basic clock</td>
<td>BCSCTL1</td>
<td>Read/write</td>
<td>057h 084h</td>
<td></td>
</tr>
<tr>
<td>system control 1</td>
<td>BCSCTL2</td>
<td>Read/write</td>
<td>058h reset</td>
<td></td>
</tr>
</tbody>
</table>

Review: Basic Clock Systems-control registers

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

Selection of DCO nominal frequency
- Which of eight discrete DCO frequencies is selected
- Define how often frequency fDCO+1 within the period of 32 DCOCLK cycles is used
- Remaining clock cycles (32-MOD) the frequency fDCO is mixed

RSEL.x Select DCO nominal frequency
- DCO.x and MOD.x set exact DCOCLK
- ... select other clock tree options

Range (RSELx) and Steps (DCOx)
The DCO temperature coefficient can be reduced by using an external resistor ROSC to source the current for the DC generator.

External Resistor

1) Internal resistors Rsel2, Rsel1, and Rsel0

2) External resistor at Rosc (P2.5/11x)

Control bits DCO0 to DCO2 set DC0 tap

Modulation bits MOD0 to MOD4 allow mixing of DC0 and DC0+1 for precise frequency generation

Example

Selected:

f3:
1000kHz

f4:
943kHz

Frequency Cycle Time

1000 nsec

960 nsec

MOD=19

DC0CLK

DC0 +0

+1

Modulation Period

f0 f1 f2 f3 f4 f5 f6 f7

To produce an intermediate effective frequency between fDC0 and fDC0+1

Cycle_time = ((32-MOD)*tDC0+MOD*tDC0+1)/32 = 1000.625 ns, selected frequency ≈ 1 MHz.
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 the new clock
  ➤ The new clock source is selected and continues with a full high period

Interrupt Service Routines

➤ Interrupt Service Routine declaration

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

EXAMPLE

```c
Interrupt(TIMERA0_VECTOR) void myISR (Void);
```

CPE/EE 421/521 Microcomputers 13

Basic Clock Systems-Examples

➤ How to select the Crystal Clock

```c
void selectClock(void)
{
  IFG2 = 0; /* reset interrupt flag register 1 */
  IFG1 = 0; /* reset interrupt flag register 2 */
  BCSCTL1 |= XT2S; /*attach HF crystal (4MHz) to XIN/XOUT */
  do {
    /*wait in loop until crystal is stable*/
    IFG1 &= ~OFIFG;
    Delay();
    IFG1 &= ~OFIFG; /*Reset osc. fault flag again*/
  } while(OFIFG & IFG1);
}
```

➤ How to select a clock for MCLK

```c
BCSCTL2 = SELM0+SELM1; /*Then set MCLK same as LFXT1CLK*/
TACTL = TASSEL0+TACLR+ID1; /*USE ACLK/4 AS TIMER_A INPUT CLOCK (1MHz)*/
```

CPE/EE 421/521 Microcomputers 14

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 OsfOff 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 SGC0 bit in the status register enables or disables SCLK.
- 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 SGC0 bit in the status register (when set).
- The XT2 oscillator sources XT2CLK (x13x and x14x only) by clearing the XT2Off bit.

CPE/EE 421/521 Microcomputers 15

Basic Clock Systems-Examples

➤ MSP430 interrupt vectors (int_vector)

```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 /* 0xFFFFC Non-maskable */
#define RESET_VECTOR 15 * 2 /* 0xFFFFE Reset [Highest Pr.] */
```

CPE/EE 421/521 Microcomputers 16

Interrupt Service Routines

➤ MSP430: Watchdog Timer

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: 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

<table>
<thead>
<tr>
<th>SSEL</th>
<th>IS0</th>
<th>IS1</th>
<th>Interval (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.064</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.9</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>32</td>
</tr>
</tbody>
</table>

- Bit 0: The SSEL bit selects the clock source for WDTCNT.
- Bit 1: When using the Watchdog Timer in interval-timer mode, the WDTIFG flag is reset automatically when the interrupt is serviced.
- Bit 2: 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.

- Bit 4: The TMSEL bit selects the operating mode: watchdog or timer.
- Bit 5: The NMI bit selects the function of the RST/NMI input pin. It is

Watchdog Timer-Diagram

Watchdog Timer-Interrupt Function

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

- 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. 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 bitCNTCL 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; NDT, 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?

- Dynamic power consumption
- Power due to short-circuit current during transition
- Power due to leakage current

\[ P = (ACV^2f) + \tau AV_{\text{short}} + V_{\text{leak}} \]

Dynamic Power Consumption

\[ ACV^2f \]

- 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
- 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

MSP430x1xx Microcontrollers
Low Power Modes
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

\[ P = ACV^2 I_t + nAV_{\text{short}} f + V_{\text{leak}} \]

Reduce the supply voltage, V

\[ f_{\text{max}} = \frac{(V - V_t)^2}{V} \]

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
  - \( I_{\text{ave}} = \int I_{\text{cc}}(t) dt / T \)
  - In most cases \( I_{\text{ave}} = \sum I_i t_i / T \)

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.

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 $BC = [\text{mAh}]$
- Battery Life
  - $BL = BC / I_{ave}$
- In the previous example, standard 800 mAh batteries will allow battery life of:
  - $BL = 750 \text{ mAh} / 2.1 \mu\text{A} = 44 \text{ years} !!!$
- Conclusion:
  - Power efficient modes
  - Interrupt driven system with processor in idle mode

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 K5 / 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)

Low power - features

- 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 #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 K5 / 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
- SCGO, and
- SCGL.

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.
Return with low-power mode bits set. When returning from the interrupt, the program counter
Two different modes are available to return from the interrupt service routine and continue the flow of
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

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.

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 awaken 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

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, 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 SRPs. All I/O port pins and RAM/registers are unchanged. Wake up is possible through all enabled interrupts.

Low-power Mode 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.

The 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 SRPs. 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.

Low-power Mode 4 (LPM4)
System Reset, 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 DCVC), 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 C Examples

---

```c
#include <msp430x14x.h>

#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
#endif /* End #defines for C */

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

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

---