Advanced Agricultural Electronics Networks Development

Lecture No. 7, Reset, Interrupts and Operating Modes

BAE 5030-352

Spring 2010

Dr. Marvin Stone
Biosystems and Agricultural Engineering
Oklahoma State University

Low-power microcontroller features

- To accommodate low power operation, low-power microcontrollers normally include:
  - Support for simultaneously response to interrupt and enablement of clocking systems
    - This support allows “wake-up” from low-power mode to a higher power more functional mode
    - Automated support for this type of automated mode switching aids in achieving lower power operation
  - Support for detection and management of low-voltage condition (brown-out)
    - Low-power microcontrollers usually operate from some type of stored energy and need to efficiently manage the situation where the potential energy available is less that that needed for operation.
  - Features on the MSP 430 will be reviewed as a model for considering all low-power microcontrollers

MSP430 Reset triggers

- Power On Reset (POR)
  - Device reset
  - Triggered by any of the following three events:
    - Powering up the µC
    - A low signal on RST/NMI pin when configured in reset mode
    - An SVS (voltage supervisor) low condition when PORON = 1
- Power Up Clear (PUC)
  - Triggered by any of the following five events:
    - A POR signal
    - Watchdog timer expiration when in watchdog mode
    - Watchdog timer security key violation
    - Flash memory security key violation
    - CPU instruction fetch from address range 0h − 01FFh
- Brown-Out Reset (BOR)
  - Triggers when voltage rises and has a hysteresis delay on drop-out

Initialization result of a POR on an MSP430

- RST/NMI pin is configured in the reset mode
- I/O pins are switched to input mode
- Peripheral modules and registers are initialized
  - Module specific, See user guide for each module
- Status register (SR) is reset
- Watchdog timer active in watchdog mode
- Program counter (PC) is loaded with address contained at reset vector location (0FFFEh)
  - The compiler sets this vector to the entry point address for “main”
  - If the reset vectors content is 0FFFFh the device will be disabled for minimum power consumption

User Code Responsibility after Reset of an MSP430

- User software must:
  - Initialize the Stack Pointer (SP), typically to the top of RAM.
  - Initialize the watchdog timer
  - Configure peripherals
  - Handle any issues regarding the Reset source
    - Source of Reset indicated by:
      - Watchdog timer flag
      - Oscillator fault flag
      - Flash memory flag

Interrupts on an MSP430

- Interrupt types:
  - System reset (RESET)
  - Non-maskable (NMI)
    - May not be disabled
  - Maskable
    - May be disabled by the user
- The interrupt service routine for a particular interrupt event is executed when:
  - Maskable Interrupts:
    - That peripheral's interrupt enable bit and the GIE bit are set
  - Non-Maskable Interrupts:
    - That NMI's interrupt enable bit is set

Non-Maskable Interrupts (NMI) on an MSP430

- Not masked by the general interrupt enable bit (GIE) in the Status Register
- Enabled by:
  - Non-Maskable Interrupt Enable (NMIIE), bit 4 in the interrupt Enable Register 1 (IE1)
  - Flash memory access violation interrupt enable (ACCVIE), bit 5 in the interrupt Enable Register 1 (IE1)
  - Flash security key violation (KEYV) generates a PUC
  - Oscillator fault interrupt enable (OFIE), bit 1 in the interrupt Enable Register 1 (IE1)
- Occurance of an interrupt is indicated by:
  - The same bit position above, but in the Interrupt Flag Register 1 (IFG1)
- NMIIE, OFIE and ACCVIE
  - Clear the Flag bits in the ISR. Do not change the Enable bits in the ISR as they are automatically handled by the CPU

Maskable Interrupts on an MSP430

- Maskable interrupts are caused by peripherals with interrupt capability including the watchdog timer overflow in interval-timer mode, A/D converter, USART, etc.
- All maskable interrupts can be disabled by the general interrupt enable (GIE) bit in the status register (SR)
- Interrupt source can be disabled individually by an interrupt enable bit

Interrupt Processing on an MSP430

- After Interrupt acceptance
  - Any currently executing instruction is completed.
  - The PC, which points to the next instruction, is pushed onto the stack.
  - The SR is pushed onto the stack.
  - The interrupt with the highest priority is selected if multiple interrupts occurred during the last instruction and are pending for service.
  - The interrupt request flag resets automatically on single-source flags. Multiple source flags remain set for servicing by software.
  - The SR is cleared. This terminates any low-power mode. Because the GIE bit is cleared, further interrupts are disabled.
  - The content of the interrupt vector is loaded into the PC: the program continues with the interrupt service routine at that address.
- Execution of the interrupt service routine
  - After executing the interrupt service routine a return from interrupt instruction is encountered
- Return from Interrupt
  - The SR with all previous settings pops from the stack and are restored.
  - The PC pops from the stack and begins execution at the point where it was interrupted.

Timing and Nesting on an MSP430

- Timing
  - The latency between acceptance of an interrupt and the start of execution of the first instruction of the interrupt-service routine is 6 cycles
  - The time required for the interrupt service routine
  - Return from interrupt takes 5 cycles to continue at the original program
- Interrupt Nesting
  - Interrupt nesting is enabled if the GIE bit is set inside an interrupt service routine
    - If GIE is set inside, any interrupt occurring during an interrupt service routine will interrupt the routine, regardless of the interrupt priorities.

Interrupt Vectors on an MSP430

- The C compiler places the 16 bit address of the appropriate ISR routine into the appropriate interrupt vector through your use of the #pragma directive

Interrupt Source	Interrupt Flag	Interrupt Type	Address	Priority
Power-up, external reset, watchdog, flash password, illegal instruction fetch	PORIFG RSTIFG WDTIFG KEYV	Reset	0FFFEh	31 (highest)
NMI, oscillator fault, flash memory access violation	NMIIFG OFIFG ACCVIFG	NMI	0FFFCh	30
device-specific		Maskable	0FFFAh	29
device-specific		Maskable	0FFF8h	28
device-specific		Maskable	0FFF6h	27
Watchdog timer	WDTIFG	Maskable	0FFF4h	26
device-specific		Maskable	0FFF2h	25
...		...	...	...
device-specific		Maskable	0FFC6h	3
device-specific		Maskable	0FFC4h	2
device-specific		Maskable	0FFC2h	1
device-specific		Maskable	0FFC0h	0 (lowest)

Setting ISR address in the interrupt vector table

- Use “pragma to set the vector number
  - Syntax: #pragma vector = Vector number
  - Vector number is enumerated as:
    - 0 at FFE0h
    - 1 at FFE2h and so on.
  - Example:

```
// Timer_A2 Interrupt Vector (TAIV) handler
#pragma vector=TIMERA1_VECTOR              // Select to react to the timer A interrupt
__interrupt void Timer_A(void)
{
  switch( TAIV )                           //Test the Timer A return value, TAIV
  {
    case  TAIV_TACCR1:  break;             // CCR1 case not handled
    case  TAIV_TAIFG:  P1OUT ^= 0x01;      // Timer A overflow case
     break;
  }
}
```