CPE 323:

MSP430 Resets, Interrupts

Aleksandar Milenkovic

Electrical and Computer Engineering
The University of Alabama in Huntsville

milenka@ece.uah.edu

http://www.ece.uah.edu/~milenka

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

Outline

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

- Resets
- Special Purpose Registers
- Software Initialization
- Interrupts
- MSP430 Interrupts
- Operating Modes
- An Example

MSP430 Resets

- Reset: a sequence of operations that put device into a well-defined state

  - From which the user's program may start

- Performed when

  - Power is first applied and

  - Device detects serious fault in hardware or software from which the user's program cannot be expected to recover

- MSP430 supports two types of resets (HW and SW controlled)

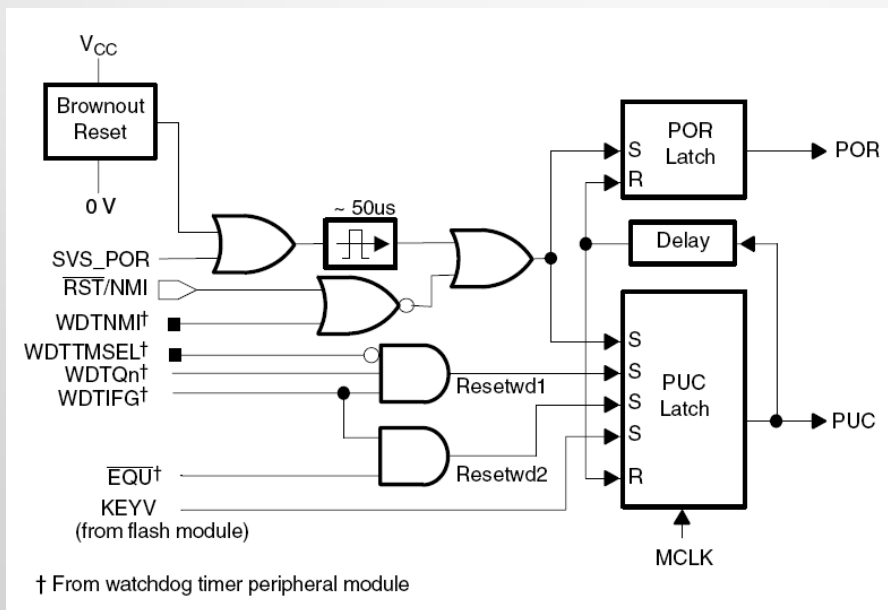 - Power-on Reset (POR)

  - Power-up Clear (PUC)

MSP430 Resets

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

- 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 (PUC)

  - A POR signal

  - Watchdog timer expiration when in watchdog mode only

  - Watchdog timer security key violation

 - A Flash memory security key violation

4

Power-on Reset

- 1. Device is powered-up

  - POR us raised if the supply voltage drops to so low a value that the device may not work correctly (Include brownout detector)

- 2. A low external signal on the #RST/NMI pin

  - If the pin is configured for the reset function rather than the nonmaskable interrupt

- 3. Supply voltage supervisor (SVS) is low

  - Circuitry that monitors power supply. It sets the SVSFG flag if the voltage falls below the programmed level and can optionally reset the device

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

Brownout Reset

- **Supply Voltage Supervisor** (SVS) circuits are used to monitor the supply voltage to embedded and other micro-controller systems for under voltage conditions. If an under voltage condition is detected then the supervisory circuit will reset the controller and keep it in that state as long as the under voltage condition persists. This type of reset is called brown out reset.

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE
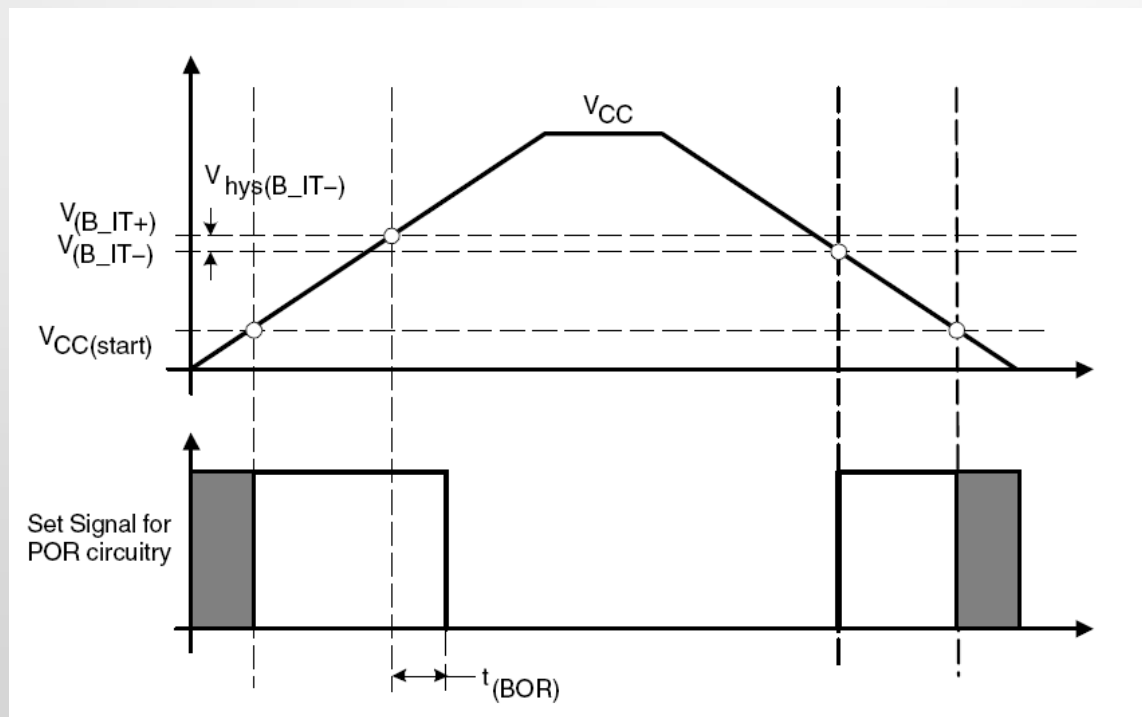
Power-up Clear

- Always follows the POR. Generated when software appears to be out of control in the following ways

- 1. Watchdog time overflows in watchdog mode

- 2. Write into the watchdog control register (WDTCTL) with incorrect password in the upper byte. Can be triggered even if the WDT is disabled or operates in the interval mode

  - Correct password is 0x5A available as symbol WDTPW

- 3. Write an incorrect password into the flash memory controller registers (FCTLn). FWKEY=0xA5

  - Protects the stored program from a runaway software

- 4. In newer devices, a PUC is triggered when we try to fetch an instruction from the range of addresses reserved for peripheral I/O or for unimplemented memory

Conditions after RESET

- Initial conditions for all registers and peripherals after POR and PUC are specified in the family's user guides; some common effects

- #RST/NMI pin is configured for reset

- Most I/O pins are configured as digital inputs

- For registers see the manual. Notation is as follows

    - rw-0: means that a bit can be read and written and is initialized to 0 after a PUC

    - rw-(0): means that a bit can be read and written and is initialized to 0 after a POR and retains its value after a PUC

- Status register is cleared (R2=0): active mode

- WDT starts in watchdog mode

- PC is loaded with the reset vector which is @0x0FFFE

Special Purpose Registers

- Interrupt Flag Registers: IFG1, IFG2

  - Track pending interrupt requests

- Interrupt Enable Registers: IE1, IE2

  - Enable selective masking of interrupts (enable/disable)

- Module Enable Registers: ME1, ME2

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

Interrupt Flag Registers

Cleared on POR, retains its value on PUC

Cleared on PUC

interrupt flag register 1 and 2

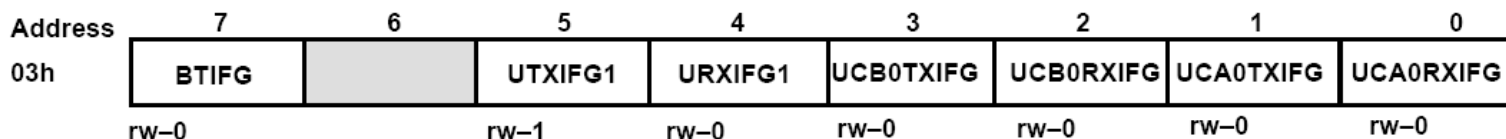| Address	7	6	5	4	3	2	1	0
02h				NMIIFG			OFIFG	WDTIFG
				rw–0			rw–1	rw–(0)
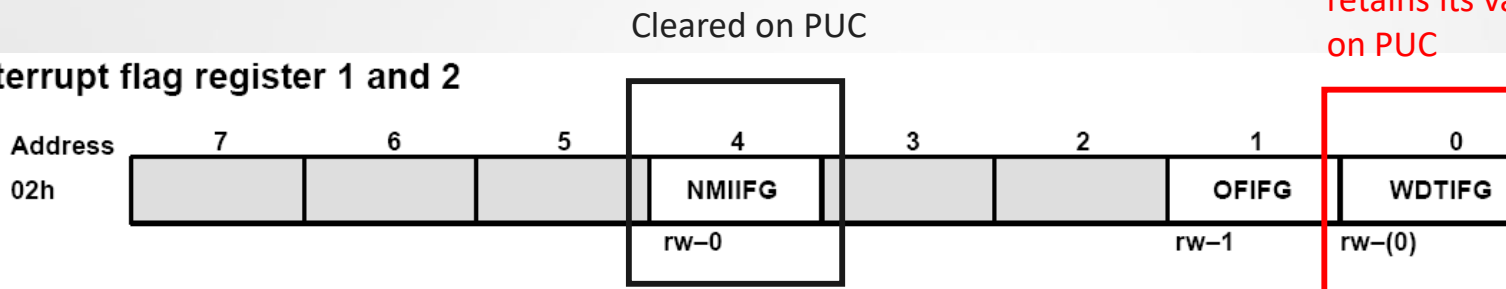
WDTIFG: Set on watchdog timer overflow (in watchdog mode) or security key violation
          Reset on V$_{CC}$ power-on or a reset condition at the $\overline{RST}$/NMI pin in reset mode

OFIFG: Flag set on oscillator fault

NMIIFG: Set via $\overline{RST}$/NMI pin

Address	7	6	5	4	3	2	1	0
03h	BTIFG		UTXIFG1	URXIFG1	UCB0TXIFG	UCB0RXIFG	UCA0TXIFG	UCA0RXIFG
	rw–0		rw–1	rw–0	rw–0	rw–0	rw–0	rw–0

UCA0RXIFG   USCI_A0 receive-interrupt flag
UCA0TXIFG   USCI_A0 transmit-interrupt flag
UCB0RXIFG   USCI_B0 receive-interrupt flag
UCB0TXIFG   USCI_B0 transmit-interrupt flag
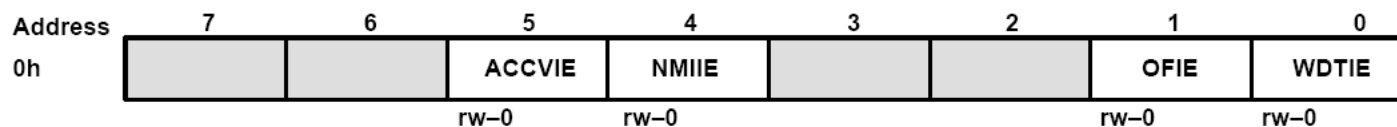URXIFG0:   USART1: UART and SPI receive flag
UTXIFG0:   USART1: UART and SPI transmit flag
BTIFG: Basic timer flag

LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Interrupt Enable Registers

- IE1, IE2

interrupt enable 1 and 2

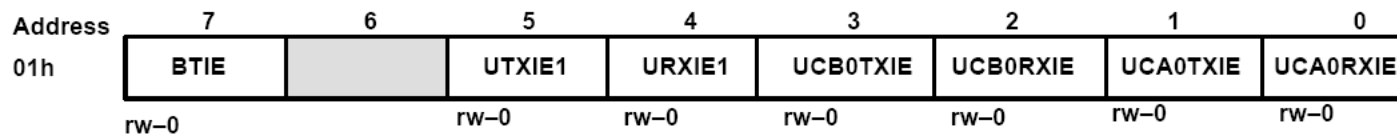| Address	7	6	5	4	3	2	1	0
0h			ACCVIE	NMIIE			OFIE	WDTIE
			rw–0	rw–0			rw–0	rw–0

WDTIE	Watchdog-timer interrupt enable. Inactive if watchdog mode is selected. Active if watchdog timer is configured as a general-purpose timer.
OFIE	Oscillator-fault-interrupt enable
NMIIE	Nonmaskable-interrupt enable
ACCVIE	Flash access violation interrupt enable

Address	7	6	5	4	3	2	1	0
01h	BTIE		UTXIE1	URXIE1	UCB0TXIE	UCB0RXIE	UCA0TXIE	UCA0RXIE
	rw–0		rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

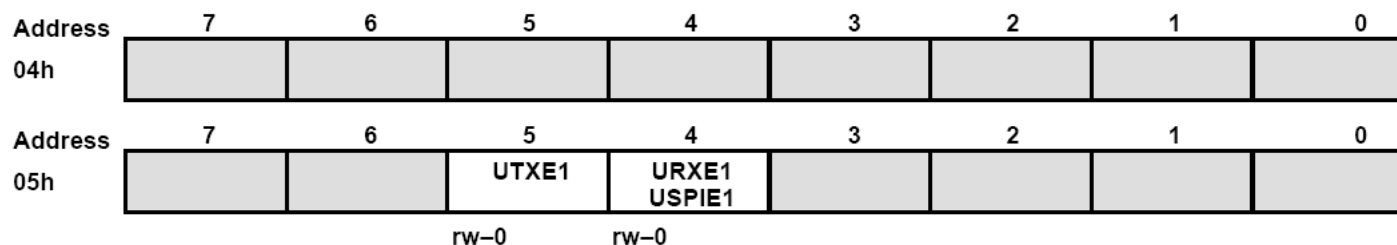| UCA0RXIE	USCI_A0 receive-interrupt enable
UCA0TXIE	USCI_A0 transmit-interrupt enable
UCB0RXIE	USCI_B0 receive-interrupt enable
UCB0TXIE	USCI_B0 transmit-interrupt enable
URXIE1	USART1 UART and SPI receive-interrupt enable
UTXIE1	USART1 UART and SPI transmit-interrupt enable
BTIE	Basic timer interrupt enable

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

Module Enable Registers

- Enable specific modules

module enable registers 1 and 2

Address	7	6	5	4	3	2	1	0
04h								

Address	7	6	5	4	3	2	1	0
05h			UTXE1	URXE1 USPIE1				

rw−0   rw−0

URXE1:   USART1: UART mode receive enable
UTXE1:   USART1: UART mode transmit enable
USPIE1:   USART1: SPI mode transmit and receive enable

Legend   rw: Bit can be read and written.
         rw-0,1:   Bit can be read and written. It is Reset or Set by PUC.
         rw-(0,1):   Bit can be read and written. It is Reset or Set by POR.
                      SFR bit is not present in device

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

Reset Related Flags

- How to identify a source of the reset when debugging

- IFG – Interrupt Flag Register (IFG1, IFG2)

  - WDTIFG: shows that the WDT timed out or its security key is violated

  - OFIFG: indicates an oscillator fault (causes a nonmaskable interrupt, not reset)

  - RSTIFG: indicates a reset caused by a signal on the #RST/NMI pin

  - PORIFG: is set on power-on reset

  - NMIIFG: flags a non-maskable interrupt caused by a signal on #RST/NMI

- These bits are not cleared by a PUC, so they can be tested to identify the source of the PUC

Software System Initialization

- Initialize the SP, typically to the top of RAM

- Configure the watchdog to the requirements of the application

- Setup the clock (clocks)

- Configure all ports (unused pins should never be left as floating inputs)

- Configure peripheral modules to the requirements of the application (e.g. TimerA, ADC12, …)

- Finally enable interrupts if needed

- Note: Additionally, the watchdog timer, oscillator fault, and flash memory flags can be evaluated to determine the source of the reset

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

Exceptions: What are they?

- Exceptions or interrupts (we use these as synonyms)

- Events caused by either hardware or software that require an urgent response

  - E.g., a packet of data has been received, a new sample from ADC is ready, etc.

- Can be asynchronous to program execution

  - Can occur at any time during an instruction execution

  - Multiple interrupts can be raised concurrently

Exceptions: What do we do?

- What do we do when an interrupt request is raised?

- Processor needs to respond, stops its current task

- Processor stores enough information to be able to resume the task later (PC and SR)

- Processor executes an Interrupt Service Routine (ISR) or Interrupt Handler

Interrupt Service Routines

- ISRs look like subroutines, but are written to handle a specific request

- Triggered by events in the CPU, peripherals, buses (hardware) or software (interrupt instruction)

- Unique characteristics

  - It can be invoked at hardware at UNPREDICTABLE times

  - ISR is carried out in such a way to allow the main code to resume without any error (like it has never occurred)

- No CALL instructions to invoke them

- No input parameters, no output parameters

Exceptions Questions

- Why do we need them?

- What are possible sources of interrupt requests (HW internal/external, SW)?

- What do we do in presence of multiple requests?

- How do we decide which request to accept (priorities, masking, selective masking)?

- Where do we find the starting address of an ISR => Interrupt Vector Table (IVT)?

- Where is IVT located?

- How do we initialize IVT?

- Answers define how we do EXCEPTION PROCESSING

Use of Exceptions

- Crucial in embedded systems –
  tasks are often triggered by external events

- Handle urgent tasks that need to be executed at higher priority than the main code

  - E.g., a received data packet should be read from a communication devices before it gets overwritten by another one

- Handle infrequent tasks

  - E.g., reading slow input devices (key pressed). An alternative is I/O polling – CPU waits for an event – would be extremely inefficient

- Wake the CPU from sleep

  - CPU is in a low-power mode to conserve energy

- Calls to an operating system (SW interrupt)

Tracking Interrupts

- Interrupt flags are use to track pending interrupts

- Each interrupt has a flag

  - E.g., TAIFG – Timer A Interrupt Flag

- It is set when a the condition for interrupt occur

  - Can be read from SW at any time (polling)

- Cleared when an interrupt is accepted

  - by HW automatically in case of single-sourced interrupts

  - by SW in the ISR in case of multiple-sourced interrupts

Masking Interrupts

- Maskable – can be disabled/enabled globally or selectively

- Nonmaskable – can be disabled

- Handling selective masking

  - Each interrupt has a corresponding enable bit

    - E.g, Timer_A has TAIE – Timer A Interrupt Enable and TAIF – Timer A Interrupt Flag

  - Allows us to selectively enable/disable interrupts (can be set or cleared in software)

- GIE – General Interrupt Enable in SR

  - When cleared all MASKABLE interrupt requests are ignored (not lost though); non-maskable are not affected (though they may have their own mask bits)

Interrupt Vector Table

- Address of an ISR (vector) is stored in a table – interrupt vector table

- Starts at defined address in memory

- Vector is determined based on the source of the interrupt

  - Single sourced interrupts:
    each vector is associated with a unique interrupt request and ISR

  - Multi-sourced interrupts: multiple interrupt requests may share a single ISR

- Priority: each vector has a distinct priority

  - Defines which ISR is selected to be processed first
    if more than one request exists

MSP430 Interrupts

- 3 types
  - System reset
  - (Non)-maskable NMI
  - Maskable
- Interrupt priorities are fixed and defined by the arrangement of modules (the nearer a module is to the CPU/NMIRS, the higher the priority)
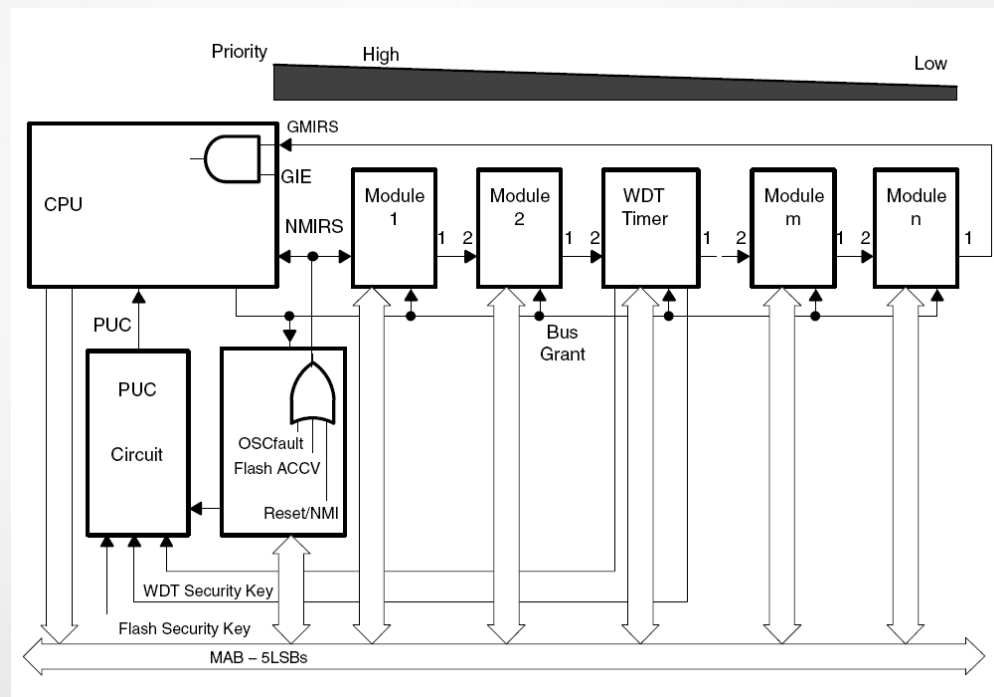
MSP430 Exception Processing

- 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 ISR

- 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*

MSP430 Family IVT

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

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

Non-Maskable Interrupts

- 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)

Non-maskable ISR (Vector 14)

- Note 3 sources

- Each handled separately, but share a single ISR

- Note that flags are cleared explicitly in SW (not cleared during exception processing)

Figure 2-6. NMI Interrupt Handler

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE
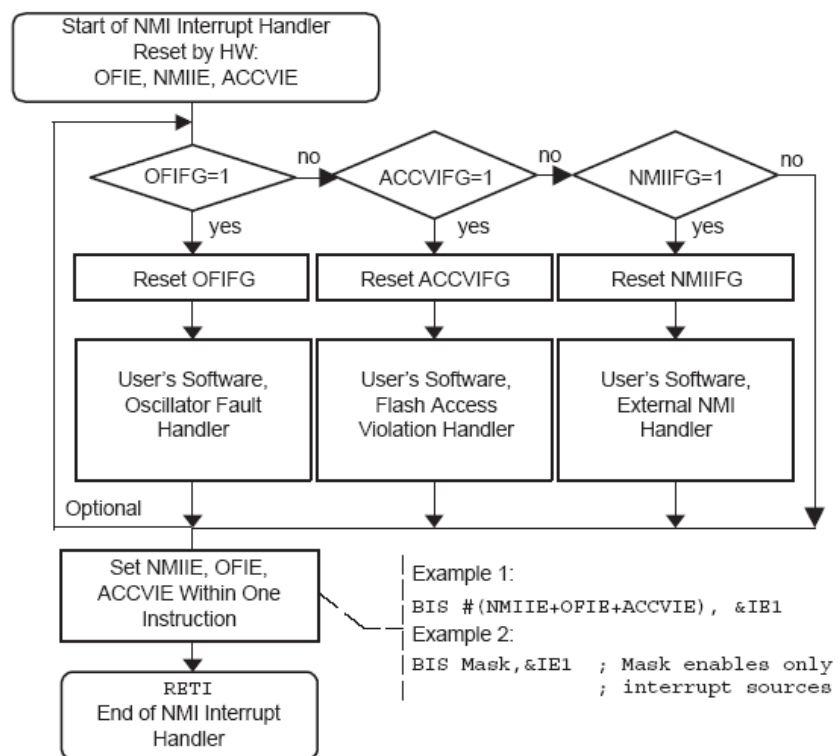
Interrupt Vector Table for F4618

```
/***********************************************************
* Interrupt Vectors (offset from 0xFFC0)
***********************************************************/
#define DAC12_VECTOR        (14 * 2u) /* 0xFFDC DAC 12 */
#define DMA_VECTOR          (15 * 2u) /* 0xFFDE DMA */
#define BASICTIMER_VECTOR   (16 * 2u) /* 0xFFE0 Basic Timer / RTC */
#define PORT2_VECTOR        (17 * 2u) /* 0xFFE2 Port 2 */
#define USART1TX_VECTOR     (18 * 2u) /* 0xFFE4 USART 1 Transmit */
#define USART1RX_VECTOR     (19 * 2u) /* 0xFFE6 USART 1 Receive */
#define PORT1_VECTOR        (20 * 2u) /* 0xFFE8 Port 1 */
#define TIMERA1_VECTOR      (21 * 2u) /* 0xFFEA Timer A CC1-2, TA */
#define TIMERA0_VECTOR      (22 * 2u) /* 0xFFEC Timer A CC0 */
#define ADC12_VECTOR        (23 * 2u) /* 0xFFEE ADC */
#define USCIAB0TX_VECTOR    (24 * 2u) /* 0xFFF0 USCI A0/B0 Transmit */
#define USCIAB0RX_VECTOR    (25 * 2u) /* 0xFFF2 USCI A0/B0 Receive */
#define WDT_VECTOR          (26 * 2u) /* 0xFFF4 Watchdog Timer */
#define COMPARATORA_VECTOR  (27 * 2u) /* 0xFFF6 Comparator A */
#define TIMERB1_VECTOR      (28 * 2u) /* 0xFFF8 Timer B CC1-2, TB */
#define TIMERB0_VECTOR      (29 * 2u) /* 0xFFFA Timer B CC0 */
#define NMI_VECTOR          (30 * 2u) /* 0xFFFC Non-maskable */
#define RESET_VECTOR        (31 * 2u) /* 0xFFFE Reset [Highest Priority] */
```

Resets Special Purpose Registers SW Initialization Interrupts MSP430 Interrupts Oper. Modes Example

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

Interrupt Service Routines

```
ASM EXAMPLE

Reset: MOV.W #SFE(CSTACK),SP  ; main program

....

     JMP $          ; loop forever

TA0_ISR: ...        ; ISR

     RETI           ; return from interrupt

;------------------

COMMON INTVEC       ; segment for vectors

ORG TIMERA_VECTOR

DW TA0_ISR          ; ISR for TimerA

ORG RESET_VECTOR    ; address to start ex.

DW Reset

END
```
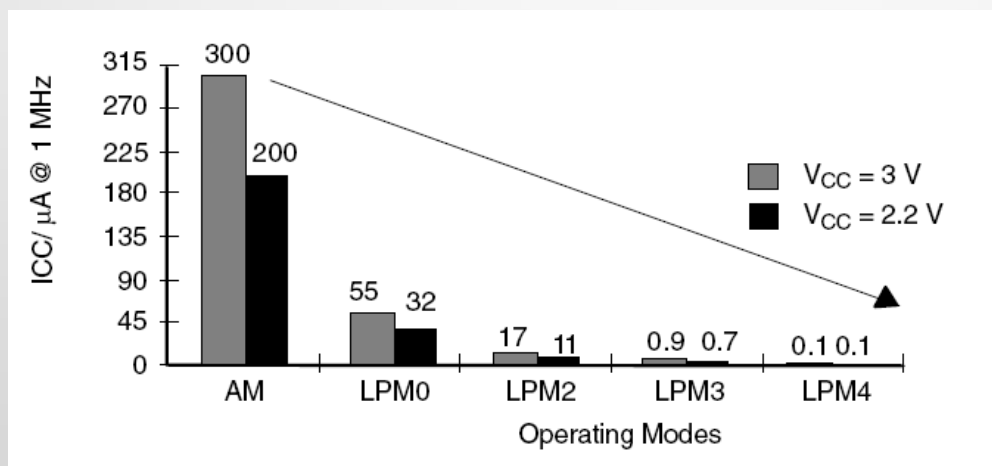
```
C EXAMPLE

#pragma vector = TIMERA0_VECTOR

__interrupt void TA0_ISR (void){

  //  ISR code

}
```

Operating Modes

- MSP430 supports several operating modes

  - Active – all clocks are up and running

  - LPM0-LPM4 – low power modes (some clocks are turned-off)

- Save energy – device is typically in a LPM and only ISRs

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

Operating Mode States

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE
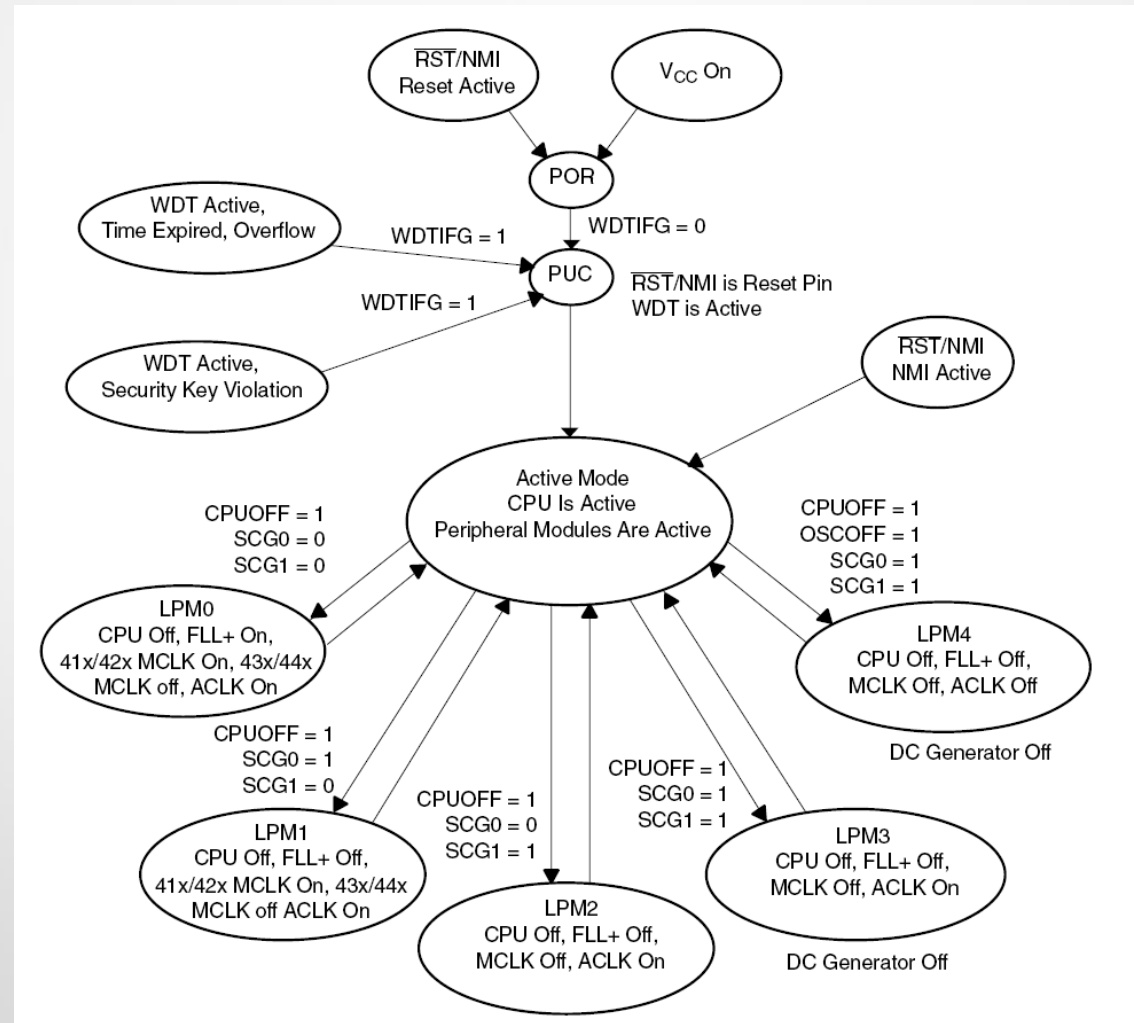
Operating Mode Bits in SR

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 (41x/42x peripheral MCLK remains on) SMCLK , ACLK are active
0	1	0	1	LPM1	CPU, MCLK, DCO oscillator are disabled (41x/42x peripheral MCLK remains on) 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 oscillator are disabled DC generator remains enabled ACLK is active
1	1	0	1	LPM3	CPU, MCLK, SMCLK, DCO oscillator are disabled DC generator disabled ACLK is active
1	1	1	1	LPM4	CPU and all clocks disabled

Blink a LED Using WDT ISR (Interval Mode)

```
/*************************************************************************
;    MSP430FG461x/F20xx Experimenter Board
;
;    Description: Toggles LED1 and LED2 (green + yellow) using WDT ISR (interval mode).
;                 LED1 (P2.2) should be ON for 6 seconds and off for 2 seconds;
;                 LED2 (P2.1) should be ON for 5 seconds and off for 3 seconds;
;                 The toggle period is 8 seconds.
;    ACLK = 32.768kHz, MCLK = SMCLK = default DCO
;
;                   MSP430FG461x
;           -----------------
;         /|\|                |
;          | |                |
;          --|RST             |
;            |                |
;            |            P2.1|--> LED2 (1 - on, 0 - off)
;            |            P2.2|--> LED1 (1 - on, 0 - off)
;
;    Alex Milenkovich, milenkovic@computer.org
;
;
***************************************************************************/
```

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

Blink a LED Using WDT ISR (Interval Mode)

```c
#include  <msp430xG46x.h>

void main(void)
{

  P2DIR |= BIT2 + BIT1;  // LED 1, LED2 are outputs
  /* configure WDT in interval mode, src clock of ACLK, 2^15 clock ticks */
  WDTCTL=WDT_ADLY_1000;  // WDT, ACLK, ~1000 ms ~ 1 s
  P2OUT += BIT2 + BIT1;  // set output bits to 1
  IE1 |= WDTIE;          // Enable watchdog interrupt
  _BIS_SR(LPM0_bits+GIE);// Goto LPM0 and set GIE
}



#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer(void){
  static unsigned int cnt = 0;          // Count the number of ISR visits
  cnt++;
  if (cnt%5 == 0)   //
    P2OUT &= ~BIT1; // turn off LED2
  else if (cnt%6 == 0) //
    P2OUT &= ~BIT2; // turn off LED1
  else if (cnt%8 == 0) {
    P2OUT |= BIT1 + BIT2;
    cnt = 0;
  }
}
```