Programming for Embedded Systems Lecture 6: Port Interrupts and Low Power Modes

Bernhard Firner

February 23, 2014

RUTGERS

Finishing Port Inputs

- There are a few more details of the I/O pins
- Interrupts and power consumption
  - See section 8.2.7-8 of the family user guide
- This is also a good time to talk about low power modes

What we Know

- We've been using these registers
  - PxDIR - Sets the direction
    - 0 is input, 1 is output
  - PxOUT - Write this register to set an output value
    - 0 is high (on), 1 is low (off)
  - PxIN - Read this register to get an input value
    - 0 is high (on), 1 is low (off)
  - PxREN (Pullup/Pulldown enable)
    - Enables pullup or pulldown resistor

Other Registers

- PxSEL - Turn on any special functions of the pin
  - 0: regular I/O
  - 1: turn on special functions
  - Things like capacitive sensing, special timers, etc
- Interrupt control registers:
  - PxIFG: Interrupt flag (source of interrupt)
  - PxIE: Interrupt enable
  - PxIES: Interrupt edge select (low->high or high->low)

Leaving Pins Alone

- When you don't use a pin set it to:
  - PxSEL = 0
  - PxDIR = 1
  - PxOUT = 0
- or
  - PxSEL = 0
  - PxDIR = 0
  - PxREN = 1
  - PxOUT = 0
- This reduces power consumption!

Port Interrupts

- We can set interrupts for each I/O pin
- All pins on a port share the same interrupt

```
//Set up an interrupt for Port1
#pragma vector=PORT1_VECTOR
__interrupt void portOneInterrupt(void) {
    //Interrupt code goes here...
 }
```

I/O Port Interrupt Variables

- Interrupt flags are set when the interrupt occurs
- The programmer is responsible for clearing the flags
  - If the flags aren't cleared the interrupts can't happen again
  - If you set a flags manually you can trigger an interrupt

Port1 Setup

- P1IFG has the interrupt flags (set when triggered)
- P1IE enables interrupts
- P1IES sets rising (0) or falling (1) edge
- These are maskable interrupts, so we must also enable those:

```
__interrupt_enable();
```

RUTGERS

Example Setup

```
//Set up the unused pins
P1SEL = 0;
P1DIR = 0xFF;
P1OUT = 0;

//Set up the red LED to be controlled by the button
P1DIR |= BIT0;
P1OUT |= BIT0;

//Set up the pushbutton at input
P1DIR &= ~BIT3;
//Turn on the pull up resistor
P1REN |= BIT3;
P1OUT |= BIT3;

//Make sure that interrupts are enabled for P1.3
//Set the interrupt to occur on the falling edge
P1IES &= ~BIT3;
P1IFG &= ~BIT3;
P1IE |= BIT3;
__enable_interrupt();
```

Example Interrupt

```
#pragma vector=PORT1_VECTOR
__interrupt void p1(void) {
    //Check if BIT3 triggered the interrupt
    if (P1IFG & BIT3) {
        //Toggle the LED
        P1OUT ^= BIT0;
        //Clear the BIT3 interrupt flag
        P1IFG &= ~BIT3;
    }
}
```

Toggling the Interrupt Edge

- The port interrupt means we don't need to loop and watch inputs
- We can also toggle the interrupt edge to catch rising and falling events
- Anecdotally, this also seems to reduce button bouncing problems

Toggling the Direction

```
#pragma vector=PORT1_VECTOR
__interrupt void p1(void) {
    //Check if BIT3 triggered the interrupt
    if (P1IFG & BIT3) {
        //Toggle the LED
        P1OUT ^= BIT0;
        //Toggle the interrupt edge
        P1IES ^= BIT3;
        //Clear the BIT3 interrupt flag
        P1IFG &= ~BIT3;
    }
}
```

Interrupt Advantages

- Interrupts make it easier to deal with asynchronous events
- Interrupts also make it possible to vastly reduce power consumption
- Basically, the MCU can sleep until interrupts are triggered
- After dealing with the interrupt, the MCU goes back to sleep

Power Consumption

- Power consumption is a major concern in embedded systems
    - Think cell phones, music players, wireless car keys, etc
- It is possible for the MSP430 to have very lower power consumption
- How we use it affects power consumption though

Turning off Inputs

- From the data sheet
  - Turn I/O pins to output, disconnect to prevent floating pins

    or
  - Use pullup/pulldown to prevent floating input

Clocks and Power Consumption

- Every clock tick consumes energy
  - PC advances by one
  - All analog circuits change state
  - etc.
- So: slow down the clock, reduce power consumption

Current Consumption and Clock Frequency

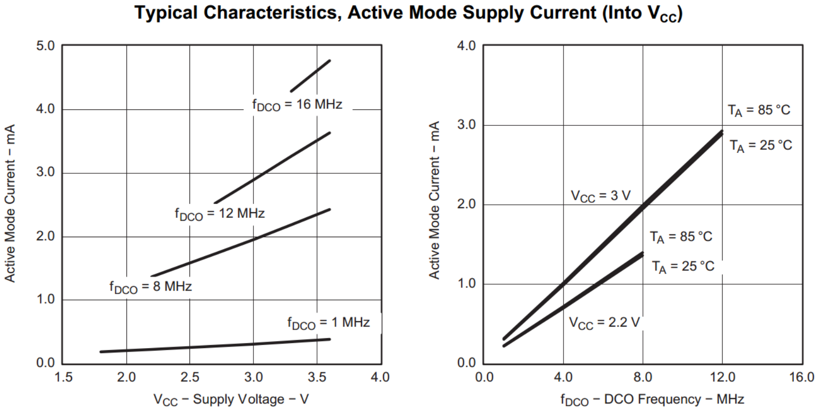

Figure 2. Active Mode Current vs V_{CC}, T_A = 25°C

Figure 3. Active Mode Current vs DCO Frequency

From MSP430G2x53, SLAS735G

Configuring Clocks to Save Power

- There are 5 low power modes, LPM0-4
    - Each mode turns off different combinations of 4 clocks
- The commands LPM0, LPM1, ..., LPM4 enter these modes
- The commands LPM0_EXIT, LPM1_EXIT, ... LPM4_EXIT leave

Clock Subsystems

- SCG1 - System clock generator 1
  - Turns off SMCLK and peripherals
- SCG0 - System clock generator 0
  - Turns off DCO
- OSCOFF
  - Turns off the crystal oscillator on LFXT1
- CPUOFF
  - Turns off the CPU

LPM0

- Turns off the CPU
- Go from 330μAmps at 1MHz to 56μAmps

LPM1

- Turns off CPU and DCO
- Not useful – could just go into LPM3

LPM2

- Turns off CPU and SMCLK
- Go from 330μAmps at 1MHz to 22μAmps

LPM3

- Turns CPU, SMCLK, and DCO
- Using 32KHz clock: 0.7μAmps
- Using VLO 0.5μAmps
    - VLO is the very low oscillator (low frequency)

LMP4

- Turn off all clocks! Down to 0.1μAmps

How Much Lifetime is This?

- about 2000mA hours in a coin cell battery
- about 220mA hours in a coin cell battery
- Active mode: 220mA hours / 330μAmps = 666.6 hours

How About LPMs?

- LPM0: 220mA hours / 56μAmps = 3,929 hours = 164 days
- LPM2: 220mA hours / 22μAmps = 10,000 hours = 417 days
- LPM3 (crystal): 220mA hours / 0.7μAmps = 314,285 hours = 35.9 years
- LPM3 (VLO): 220mA hours / 0.5μAmps = 440,000 hours = 50 years
- LPM4: 220mA hours / 0.1μAmps = 2,200,000 hours = 251 years
    - Obviously batteries don't actually last this long

What Can Happen While We Sleep?

- Interrupts
- Certain peripherals
  - e.g. Timer A and B can generate PWM signals

RUTGERS

Waking Up

- Need to use interrupts to wake up
- Non-clock interrupt to wake up from LPM4 though
- In general LPM3 is the simplest to use
  - Use crystal for exact sleep timing
  - Otherwise use internal VLO

Sleeping Through Button Presses

- Take the code from before and add this line:

```
LPM4;
```

- Once that line happens the code stops!
- If an interrupt occurs the CPU wakes up briefly to handle the interrupt
- Unless we call LPM4_EXIT the code never advances outside of the interrupt