ELEC327: Lecture 3

Clocks, Interrupts

Today

- Soldering
- MSP430 Clocks
- A-to-D conversion
- Low power/Interrupts

Soldering

- You’re only soldering one thing this week, so don’t fret, but get it done!

Resources
- http://www.youtube.com/watch?v=J5Sb21qbpEQ
- http://www.sparkfun.com/tutorials/category/2

Surface Mount Device Soldering
Soldering Iron

- The main soldering tool
- Used to melt solder

Solder

- Metal “glue” used to join together metal parts
- Contains flux
- Made of Tin and Copper, usually Lead

Tweezers, Clippers, & Pliers

- Tools used to handle components and PCBs

Sponge

- Used to clean off soldering iron tip
- Can be made of various materials
- Should be used every time before soldering a joint
Flux

- Makes solder melt easier
- Always use extra flux for rework

Solder Wick

- Use to remove solder
- Careful, it gets really hot!

PCB

- Printed Circuit Board
- Comprised mainly of fiberglass and copper foil

Leads and Pads

- Component and board parts that you will solder together
What is oxidization?

- Oxidization is what happens when oxygen breaks down matter
- It impedes heat and electricity transfer
- As a result you must clean your iron tip constantly... really!

Cleaning your soldering iron

- Cleaning your soldering iron is one of the most important aspects of soldering. With SMD we suggest doing it after every joint
- If you can’t get the oxidization off using a sponge try tip tinner

Perfect solder joint, step 1 of 4

- Hold the iron touching one of the pads for a count of one
- Make sure to touch the side of the iron tip to the pad, not the actual tip of the iron
Perfect solder joint, step 2 of 4

- Feed a tiny bit of solder onto the pad while continuing to hold the iron to the pad
- Feed enough solder onto the pad to cover the pad, but no more

Perfect solder joint, step 3 of 4

- Continue to hold the soldering iron on the pad and pick up your component using the tweezers with your other hand
- Make sure the tweezers are making contact only with the sides of the component that do not have leads, otherwise you may get your tweezers in the solder

Perfect solder joint, step 4 of 4

- Continue to hold the soldering iron on the pad so the solder stays liquid
- Slide the component into the solder from the side while keeping the bottom of the component flat on the PCB
- Make sure you slide the component far enough into the pad so that the pad on the other side of the component is at least half exposed

Perfect solder joint, step 4 of 4

- Continue to hold the component on the pad with your tweezers and remove the soldering iron, wait for the solder to solidify and then remove your tweezers
Oops.

- If you don’t slide the component in from the side you may wind up with a component that does not touch pads on both sides.
- If this happens just reflow the solder on the first join and replace the component.

Here’s How to Use Solder Wick

- Apply flux to the solder you want to remove.
- Place the solder wick on top of the solder you want to remove.
- Sandwich the solder wick between the solder and your iron until the heat sucks the solder into the wick.

Here’s How to Use Solder Wick

- The solder will flow into your wick and the wick will turn silver with solder.
- This may take a while, be patient and apply constant contact with the iron.
- Try rolling the soldering iron along the wick if you’re really having trouble.
- You might also want to try cleaning your iron if it isn’t working.

Architecture
Architecture

Auxiliary Clock
Master Clock
Sub-Main clock

System State at Reset

- At power-up (PUC), the brownout circuitry holds device in reset until Vcc is above hysteresis point
- RST/NMI pin is configured as reset
- I/O pins are configured as inputs
- Clocks are configured
- Peripheral modules and registers are initialized (see user guide for specifics)
- Status register (SR) is reset
- Watchdog timer powers up active in watchdog mode
- Program counter (PC) is loaded with address contained at reset vector location (0FFFFh). If the reset vector content is 0FFFh, the device will be disabled for minimum power consumption

Software Initialization

After a system reset the software must:

- Initialize the stack pointer (SP), usually to the top of RAM
- Reconfigure clocks (if desired)
- Initialize the watchdog timer to the requirements of the application, usually OFF for debugging
- Configure peripheral modules

Clock System

- Very Low Power/Low Frequency Oscillator (VLO)*
  - 4 – 20kHz (typical 12kHz)
  - 500nA standby
  - 0.5%/°C and 4%/Volt drift
  - Not in ‘21x1 devices
- Crystal oscillator (LFXT1)
  - Programmable capacitors
  - Failsafe OSC_Fault
  - Minimum pulse filter
- Digitally Controlled Oscillator (DCO)
  - 0-to-16MHz
  - ± 3% tolerance
  - Factory calibration in Flash

On PUC, MCLK and SMCLK are sourced from DCOCLK at ~1.1 MHz. ACLK is sourced from LFXT1CLK in LF mode with an internal load capacitance of 6pF. If LFXT1 fails, ACLK defaults to VLO.
**DCO Power**

If we turn the CPU off, we’re at ~ 50 µA.

**G2xxx - No Crystal Required DCO**

<table>
<thead>
<tr>
<th>DCO Calibration Data (provided from factory in flash info memory segment A)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DCO Frequency</strong></td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td>1 MHz</td>
</tr>
<tr>
<td>1 MHz</td>
</tr>
<tr>
<td>8 MHz</td>
</tr>
<tr>
<td>8 MHz</td>
</tr>
<tr>
<td>12 MHz</td>
</tr>
<tr>
<td>12 MHz</td>
</tr>
<tr>
<td>16 MHz</td>
</tr>
<tr>
<td>16 MHz</td>
</tr>
</tbody>
</table>

// Setting the DCO to 1MHz
if (CALBC1_1MHZ == 0xFF || CALDCO_1MHZ == 0xFF)
while(1); // Erased calibration data? Trap!
BCSCTL1 = CALBC1_1MHZ; // Set range
DCOCTL = CALDCO_1MHZ; // Set DCO step + modulation

- G2x1 devices have 1MHz DCO constants only. Higher frequencies must be manually calibrated.
- G2xx2 & G2xx3 (like the G2553) have all 4 constants + calibration values for the ADC & temperature sensor.

---

**Run Time Calibration of the VLO**

- Calibrate the VLO during runtime
- Clock Timer_A runs on calibrated 1MHz DCO
- Capture with rising edge of ACLK/8 from VLO
- \( f_{VLO} = 8\text{MHz}/\text{Counts} \)
- Code library on the web (SLAA340)

**System MCLK & Vcc**

- Match needed clock speed with required \( V_{cc} \) to achieve the lowest power
- External LDO regulator required
- Unreliable execution results if \( V_{cc} < \) the minimum required for the selected frequency
- All G2xxx device operate up to 16MHz
Watchdog Timer Failsafe Operation

- If ACLK / SMCLK fail, clock source = MCLK (WDT+ fail safe feature)
- If MCLK is sourced from a crystal, and the crystal fails, MCLK = DCO (XTAL fail safe feature)

Lab 2

- Due Thursday 1/17 !!!
- MSP430 GPIO module

Watchdog Timer Clock Source

- Active clock source cannot be disabled (WDT mode)
- May affect LPMx behavior & current consumption
- WDT(+) always powers up active

Next labs

- Using the on chip ADC and Timer modules
- Interrupts!
- Low power
- Serial communications
Analog-to-Digital Converters

- Basic linear concept
- Nyquist!
- Quantization noise
- Types
  - Flash
  - Successive Approximation
  - Sigma-Delta

Sample Timing

- Reference must settle for ≤30μS
- Selectable hold time
- 13 clock conversion process
- Selectable clock source
  - ADC10OSC (~5MHz)
  - ACLK
  - MCLK
  - SMCLK

Autoscan + DTC Performance Boost

// Software
Res[pRes++] = ADC10MEM;
ADC10CTL0 &= ~ENC;
if (pRes < NR_CONV) {
  CurrINCH++;
  if (CurrINCH == 3)
    CurrINCH = 0;
  ADC10CTL1 &= ~INCH_3;
  ADC10CTL1 |= CurrINCH;
  ADC10CTL0 |= ENC+ADC10SC;
}

Fast Flexible ADC10

- 10-bit 8 channel SAR ADC
  - 6 external channels
  - Vcc and internal temperature
- 200 ksp+s
- Selectable conversion clock
- Autoscan
  - Single
  - Sequence
  - Repeat-single
  - Repeat-sequence
- Internal or External reference
- Timer-A triggers
- Interrupt capable
- Data Transfer Controller (DTC)
- Auto power-down

Sample Timing...
Timer_A2 and A3 Features

- Asynchronous 16-bit timer/counter
- Continuous, up-down, up count modes
- 2 or 3 capture/compare registers
- PWM outputs
- Two interrupt vectors for fast decoding

Interrupts and Stack

- 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
- 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

MSP430G2553 Vector Table

<table>
<thead>
<tr>
<th>Interrupt Source</th>
<th>Interrupt Flag</th>
<th>System Interrupt</th>
<th>Word Address</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power-up</td>
<td>PORIFG</td>
<td>Reset</td>
<td>0FFFEh</td>
<td>31</td>
</tr>
<tr>
<td>External Reset</td>
<td>RSTIFG</td>
<td></td>
<td>0xFFFFh</td>
<td></td>
</tr>
<tr>
<td>PC out-of-range</td>
<td></td>
<td></td>
<td>0xFFFFh</td>
<td></td>
</tr>
<tr>
<td>Oscillator fault</td>
<td></td>
<td></td>
<td>0xFFFFh</td>
<td></td>
</tr>
<tr>
<td>Flash memory access violation</td>
<td></td>
<td></td>
<td>0xFFFFh</td>
<td></td>
</tr>
<tr>
<td>Watchdog Timer+</td>
<td></td>
<td></td>
<td>0xFFFFh</td>
<td></td>
</tr>
<tr>
<td>Comparator_A+</td>
<td></td>
<td></td>
<td>0xFFFFh</td>
<td></td>
</tr>
<tr>
<td>Watchdog Timer+</td>
<td></td>
<td></td>
<td>0xFFFFh</td>
<td></td>
</tr>
<tr>
<td>Timer1_A3</td>
<td>TA1CCR0 CCIFG</td>
<td>maskable</td>
<td>0xFFFAh</td>
<td>29</td>
</tr>
<tr>
<td>Timer1_A3</td>
<td>TA1CCR2 TA1CCR1 CCIFG, TAIFG</td>
<td>maskable</td>
<td>0xFFFAh</td>
<td>28</td>
</tr>
<tr>
<td>Comparator_A+</td>
<td>CAFG</td>
<td>maskable</td>
<td>0xFFFAh</td>
<td>27</td>
</tr>
<tr>
<td>USCI_A0/USCI_B0 receive</td>
<td>UCA0RXIFG</td>
<td>maskable</td>
<td>0xFFE6h</td>
<td>23</td>
</tr>
<tr>
<td>USCI_A0/USCI_B0 transmit</td>
<td>UCA0TXIFG</td>
<td>maskable</td>
<td>0xFFE6h</td>
<td>22</td>
</tr>
<tr>
<td>ADC10</td>
<td>ADC10FG</td>
<td>maskable</td>
<td>0xFFE6h</td>
<td>21</td>
</tr>
<tr>
<td>IO Port P2 (up to 8)</td>
<td>P2FGE to P2FG</td>
<td>maskable</td>
<td>0xFFE6h</td>
<td>19</td>
</tr>
<tr>
<td>IO Port P1 (up to 8)</td>
<td>P1FGE to P1FG</td>
<td>maskable</td>
<td>0xFFE6h</td>
<td>18</td>
</tr>
<tr>
<td>Boot Strap Loader Security</td>
<td></td>
<td></td>
<td>0xFFE6h</td>
<td>16</td>
</tr>
<tr>
<td>Unused</td>
<td></td>
<td></td>
<td>0xFFE6h to 0xFFFFh</td>
<td>14</td>
</tr>
</tbody>
</table>

ISR Coding

```c
#pragma vector=WDT_VECTOR
__interrupt void WDT_ISR(void)
{
    IE1 &= ~WDTIE;       // disable interrupt
    IFG1 &= ~WDTIFG;     // clear interrupt flag
    WDTCTL = WDTPW + WDTHOLD; // put WDT back in hold state
    BUTTON_IE |= BUTTON; // Debouncing complete
}
```

```c
#pragma vector - the following function is an ISR for the listed vector
interrupt void - identifies ISR name
No special return required
```