Lesson #20: Low Power Principles

Lesson #20 Learning Objectives: Upon successfully completing this lesson and the associated homework, students will be able to:

1. Calculate the received power and link margin for a wireless link.
2. Design a wireless link to meet the required link margin.
3. Calculate the received power using the Friis model.
4. Calculate and plot the effects of multipath interference on signal transmission

20-1. So Speed matters?

As a reminder, our goal is to build a small computing device that is powered completely from power absorbed by an antenna. We have talked about backscatter and how we can use reflected energy to transmit data, although we haven't seen it in practice yet. It gives us communication on the cheap. We know that power is at a premium, and we are therefore shooting for the lowest power possible. Speed kills.

In the last lesson, we created a couple of programs that made a pulsating beep sound. In the end we had 2 programs that did the exact same task – annoy us with a beep. However, what did you note from the power draw? There should have been a massive difference.

20-2. Yes

There is an inherent relationship between speed and heat. From a physics standpoint, rub things together faster, and it heats up. A lá, caveman fire. Or if a car drives faster, its tire gets hotter.

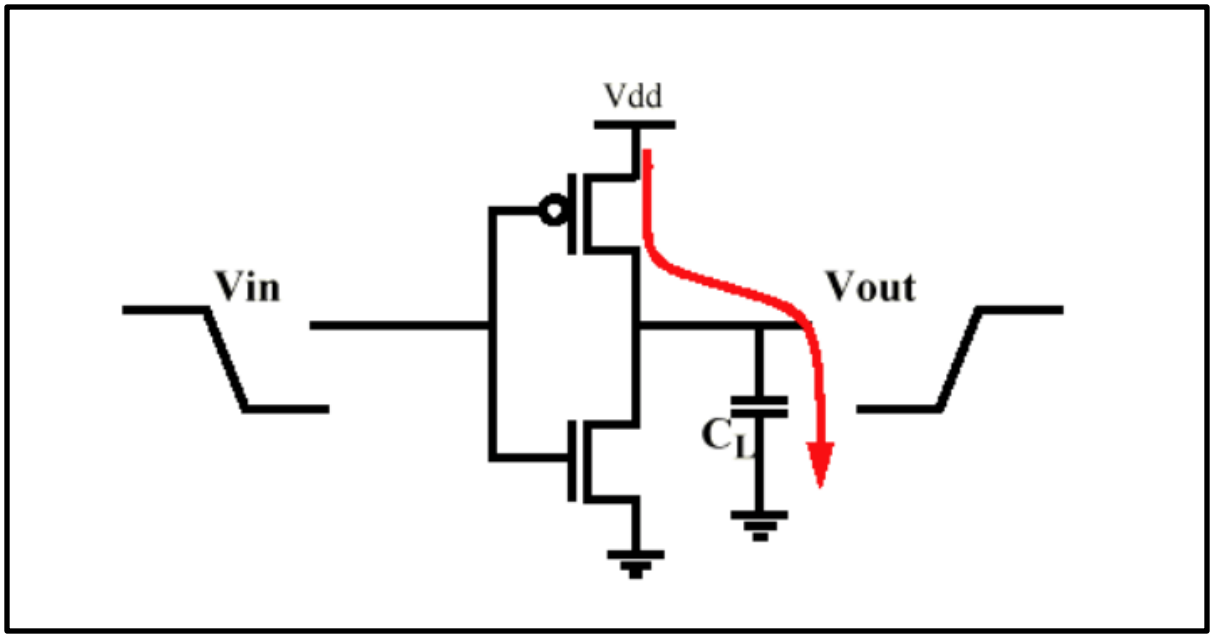
Figure 1Mr. Transistor

The same thing applies to computers. You want to overclock your CPU and GPU so you can run Big Rigs in 4K with the graphics enhancements and the Realism mod? Then better get some fancy heatsinks.

So the faster a computer goes, the heat it generates as a consequence of more power draw. Alternatively, lowering the speed (underclocking?) will lead to a reduction in power. But why? Let's ask our good friend, Mr. Transistor.

20-1.2 Yes, yes it does

Most. Ok, basically ALL CPU chips we use today are made in CMOS. Complementary Metal Oxide Semiconductor. The ‘C’ part is key. Complementary. Look at the schematic for a transistor below. (you have to draw it)



For this gate, assuming that the input is in a HIGH state, the top transistor is ON/OFF and the bottom transistor is ON/OFF. This means that for ideal transistors in steady state, there is no power leaving VDD.

When the input is LOW, then the top transistor is ON/OFF and the bottom transistor is ON/OFF. Again, at steady state, there is no power leaving VDD. The capacitor CL gets charged, and that is it.

However, the magic, or rather, the unnervingly horrible leakage of power, all happens in the transition. Because a transistor is not idea, there is some finite time when it is switching from a high state to a low state (or vice-versa). During these times, the channel is neither open or closed, but in a semi in-between state. Look at the graph. That means there is a direct connection from VDD to ground. SsSSsshsshshshhshhhhshshshshshshsshpkunk. Did you hear that? That was the sound of millions of electron-hole pairs screaming out and then being silenced as they recombine.

Peak power consumption for a CMOS gate occurs midway between a binary transition. The basic mechanism is finite time between switching leads to an effective capacitance that is charged and discharged every transition. This is known as the dynamic power consumption. The steady-state power consumption would describe the losses when everything is at steady-state. Going to back to Physics I: The energy stored on a capacitor is:

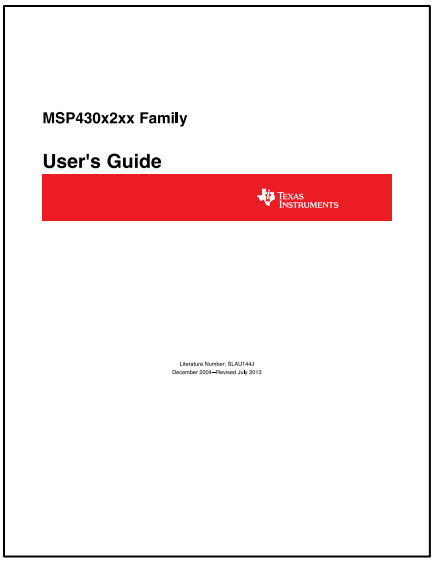
And a Watt is the flow 1 Joule per second. So if we divide the energy stored on the capacitor by time (seconds), we'll end up with:

Ploss = ½ C v^2 f.

The 'f' is the switching frequency of the particular element. If switch faster, there will be more charge/discharge cycles and we expect power dissipation to increase. If we slow down he frequency, power dissipation decreases. So careful clock management is the key.

20-3. Clocks in the MSP430

This is a very confusing and tedious subject. The MSP430 has a very complicated clocking system. This means that there is a lot that it can do, but it also means, there's a lot that it can do.



Remember the good ole Family User's guide? Ours is for the MSP430x2xx Family. This will cover both chips we are using. Chapter 5 covers the clocking system. There are a couple things we want to do with the clock in our program. We want to set up where the different clocks are being sourced from, turn off the ones not being used, and set the clock to the lowest speed possible. All these things can be found in the User's guide. Found here: http://www.ti.com/lit/ds/symlink/msp430f2011.pdf

The gist is: There are three clocks that drive the MSP430. ACLK (auxiliary clock), MCLK (Main System Clock) and SMCLK (Sub System Clock). Each of these system clocks can be assigned a different source. That source can be selected from a variety of oscillators that generate square waves. This square wave can be divided down to be slower, and will finally be chosen as the *official* ACLK or MCLK or SMCLK. That's the main thing diagram 5-2 is telling us.

The MCLK is what drives the CPU to process instructions. The SMCLK is what would normally be used to drive peripherals. The ACLK is something we could use to drive peripherals as well. Each has their own uniqueness about them.

As section 5.2.1 states, we are going to enter into a paradox.

5.2.1 Basic Clock Module+ Features for Low-Power Applications

Conflicting requirements typically exist in battery-powered applications:

- Low clock frequency for energy conservation and time keeping
- High clock frequency for fast reaction to events and fast burst processing capability
- Clock stability over operating temperature and supply voltage

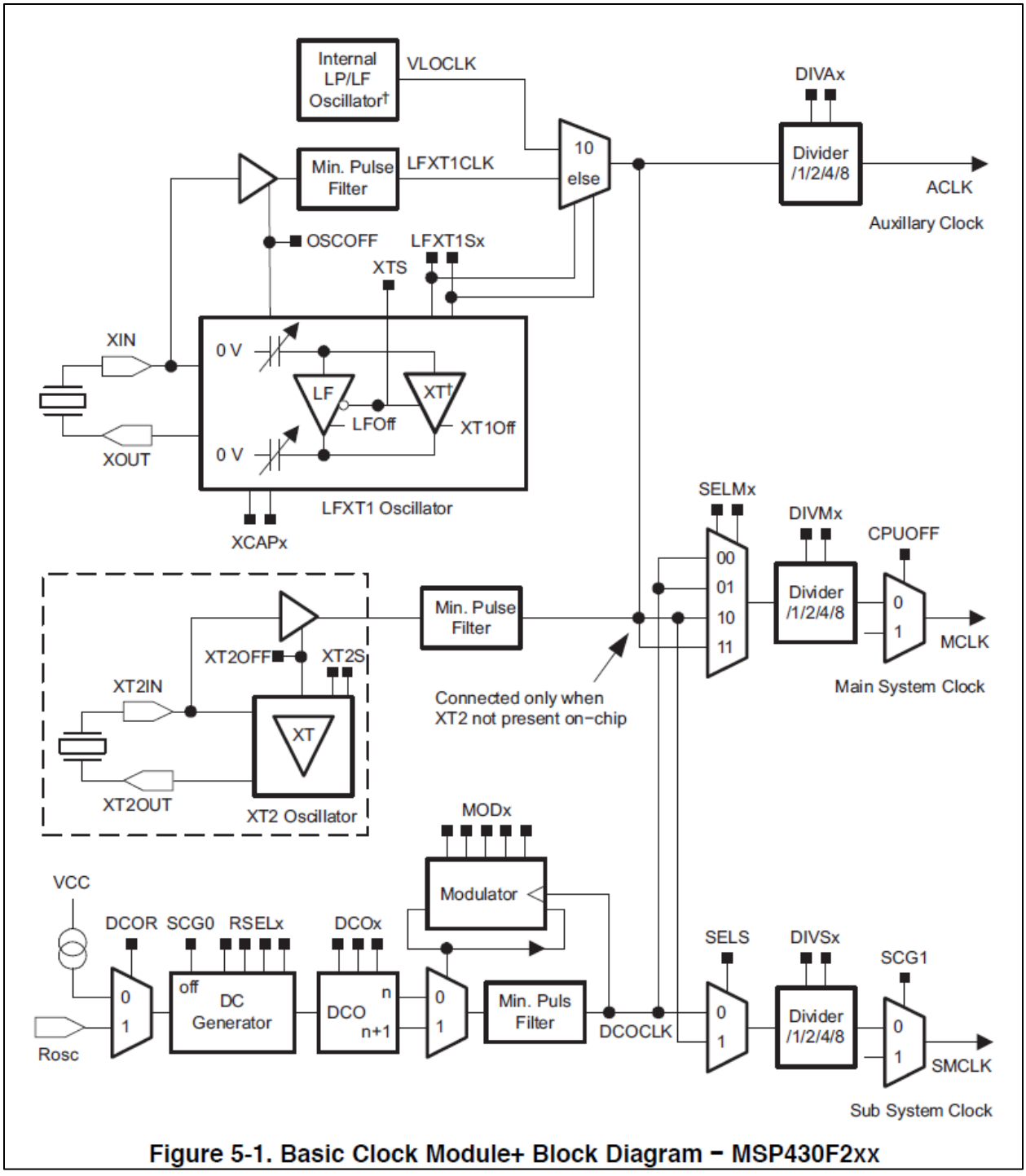


Figure 5-1. Basic Clock Module+ Block Diagram – MSP430F2xx

So what's a developer to do? For the best performance, we can run ACLK from a 32 kHz watch crystal, and leave it power peripherals while we wait for something to occur. Like a button press. (The two pads next to the microcontroller are there for a watch crystal, but we have just left it unpopulated for now.) MCLK can be operated from the on-chip oscillator and

only be activated when driven by an interrupt. Perform a short burst of processing, and then go back down for a rest. SMCLK can then be specifically selected for the task.

Alternatively, a very-low power, low-frequency oscillator (VLO) can be used to source ACLK and provide a time base while we are waiting for an in interrupt.

Take note in the figure above: the CPUOFF bit will turn off the MCLK, thus shutting down all processing. Each clock could be sourced from the VLOCLK, or a crystal (the LFXT1CLK or XT2), or from the DCOCLK. Each clock source can then be divided down slower (/1, /2, /4, /8).

The DCOCLK can be configured for a variety of frequencies. It's out of the scope to go into exactly how this is happening (It's wonky), but figure 5-6 gives some typical values. The important thing to note is that the DCO clock is ONLY guaranteed to output certain frequencies at a few factory calibrated values. These are the CAL_DCO_1MHZ and so forth that you have seen in ECE 322. All other frequencies are more of a guess and can change between individual microcontrollers due to fabrication process variation.

20-4. Time to Fall Back – Or actually remembering to take the time to go to all the clocks that you have and remember to set them so you don't show up get into a frenzy to rush and show up an hour early, err, save power that is

In DCOCTL, we can select the frequency of the DCO and therefore MCLK. In BCSCTL1, we can select the crystal oscillator, the ACLK divider and the DCO range select. In BCSCTL2, we can select the MCLK source, the MCLK divider, the SMCLK source, and the SMCLK divider. In BCSCTL3, we can mainly adjust settings for the crystal oscillator.

Download the code from Blackboard and create a program with low power beeping.