Program 10.1 shows the steps 0 to 6 to activate the LM4F123/TM4C123 Launchpad with a 16 MHz main oscillator to run at 80 MHz. 0) Use RCC2 because it provides for more options. 1) The first step is to set BYPASS2 (bit 11). At this point the PLL is bypassed and there is no system clock divider. 2) The second step is to specify the crystal frequency in the four XTAL bits using the code in Table 10.1. The OSCSRC2 bits are cleared to select the main oscillator as the oscillator clock source. 3) The third step is to clear PWRDN2 (bit 13) to activate the PLL. 4) The fourth step is to configure and enable the clock divider using the 7-bit SYSDIV2 field. If the 7-bit SYSDIV2 is **n**, then the clock will be divided by **n**+1. To get the desired 80 MHz from the 400 MHz PLL, we need to divide by 5. So, we place a 4 into the SYSDIV2 field. 5) The fifth step is to wait for the PLL to stabilize by waiting for PLLRIS (bit 6) in the **`SYSCTL_RIS_R`** to become high. 6) The last step is to connect the PLL by clearing the BYPASS2 bit. To modify this program to operate on other microcontrollers, you will need to change the crystal frequency and the system clock divider.

CHECKPOINT 10.1

How would you change Program 10.1 if your microcontroller had an 8 MHz crystal and you wish to run at 50 MHz?

Hide Answer

Change the specification from 16 MHz to 8 MHz. Change the line
SYSCTL_RCC_R += 0x00000540; // 10101, configure for 16 MHz crystal
to
SYSCTL_RCC_R += 0x00000380; // 01110, configure for 8 MHz crystal
Change the specification from divide by 5 to divide by 8. Change the line
SYSCTL_RCC2_R += (4<<22); // configure for 80 MHz clock
to
SYSCTL_RCC2_R += (7<<22); // configure for 50 MHz clock

```
void PLL_Init(void){
  // 0) Use RCC2
  SYSCTL_RCC2_R |=  0x80000000;  // USERCC2
  // 1) bypass PLL while initializing
  SYSCTL_RCC2_R |=  0x00000800;  // BYPASS2, PLL bypass
  // 2) select the crystal value and oscillator source
  SYSCTL_RCC_R = (SYSCTL_RCC_R &~0x000007C0)   // clear XTAL field, bits 10-6
                 + 0x00000540;   // 10101, configure for 16 MHz crystal
  SYSCTL_RCC2_R &= ~0x00000070;  // configure for main oscillator source
  // 3) activate PLL by clearing PWRDN
  SYSCTL_RCC2_R &= ~0x00002000;
  // 4) set the desired system divider
  SYSCTL_RCC2_R |= 0x40000000;   // use 400 MHz PLL
  SYSCTL_RCC2_R = (SYSCTL_RCC2_R&~ 0x1FC00000)  // clear system clock divider
                  + (4<<22);      // configure for 80 MHz clock
  // 5) wait for the PLL to lock by polling PLLLRIS
  while((SYSCTL_RIS_R&0x00000040)==0){};  // wait for PLLRIS bit
  // 6) enable use of PLL by clearing BYPASS
  SYSCTL_RCC2_R &= ~0x00000800;
```

Help

}

Program 10.1. Activate the LM4F/TM4C with a 16 MHz crystal to run at 80 MHz (C10_PLL.zip).

We can make a first order estimate of the relationship between work done in the software and electrical power required to run the system. There are two factors involved in the performance of software: efficiency of the software and the number of instructions being executed per second

Software Work = algorithm * instructions/sec

In other words, if we want to improve software performance we can write better software or increase the rate at which we execute instructions. Recall that the compiler converts our C software into Cortex M machine code, so the efficiency of the compiler will also affect this relationship. Furthermore, most compilers have optimization settings that allow you to make your software run faster at the expense of using more memory. On the Cortex M, most instructions execute in 1 or 2 bus cycles. See section 3.3 in **CortexM4_TRM_r0p1.pdf** for more details. In CMOS logic, most of the electrical power required to run the system occurs in making signals change. It takes power to make a digital signal rise from 0 to 1, or fall from 1 to 0. It takes some power to run independent of frequency. Simplifying things greatly, we see a simple and linear relationship between bus frequency and electrical power. Let *m* be the slope of this linear relationship

Power = $m * f_{Bus}$

Some of the factors that affect the slope m are operating voltage and fundamental behavior of how the CMOS transistors are designed. If we approximate the Cortex M processor as being able to execute one instruction every two bus cycles, we can combine the above two equations to see the speed-power tradeoff.

Software Work = algorithm * ½ f_{Bus} = algorithm * ½ Power/*m*

Observation: To save power, we slow down the bus frequency removing as much of the wasted bus cycles while still performing all of the required tasks.

For battery-powered systems the consumed power is a critical factor. For these systems we need to measure power. Since there are so many factors that determine power, the data sheets for the devices will only be approximate. Since power equals voltage times current, and we know the voltage (in our case 3.3V), we need to measure supply current. Figure 10.2 shows a current sense amplifier that can measure current to a Target system. We made a special printed circuit board placing a fixed resistor (R1=1 ohm in this circuit) between the 3.3V supply and the target system. The voltage across R1 is a linear function of the current to the target. The current sense amplifier (LT1187) amplifies this signal and the output of the amplifier is measured by an analog to digital converter.

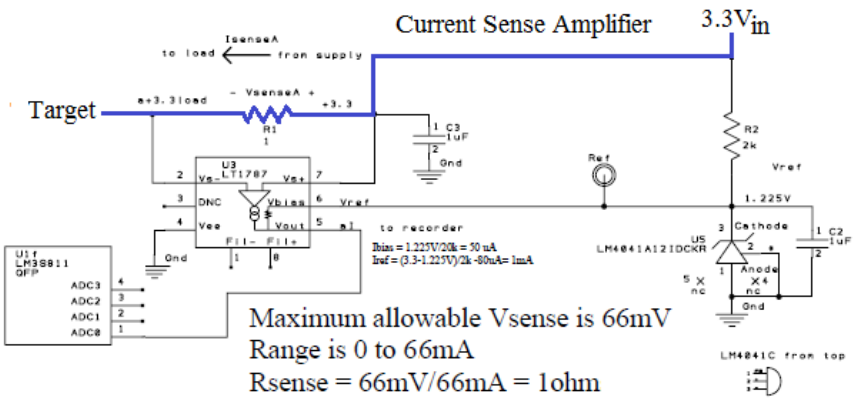

Figure 10.2. A current sense amplifier can be used to measure instantaneous current to the target. The blue trace shows the current path from supply to target.

Figure 10.3 shows a current measurement for a battery-powered medical device based on a TI MSP430. In sleep mode all the clocks are turned off and the current drops to 0.5 µA. Just like the TM4C123 the MSP430 has control over which I/O devices are active. You can see from the data, the I/O devices are turned on one at a time (sample from ADC, transmit using RF, and receive using RF). The total energy needed to collect one measurement on this system can be found by multiplying current*voltage*time, where current and time are measured in Figure 10.3. In this calculation, assume we take on measurement every hour (sleeps for 1 hour, wakes up samples, transmits, receives, and then goes back to sleep). Each hour it needs 122 mA-sec from the battery (the actual energy is 122mA-s*3.3V which equals 0.4J).

(13.8mA*0.4s+30.6mA*0.680s+20.3mA*4.63s+0.0005mA*3600s) = 122 mA-s

A battery with 130mA-hr has 468000 mA-sec of storage. This battery will run the system for 3832 hours or 5 months. In this system 94% of the power is consumed by the RF communication.

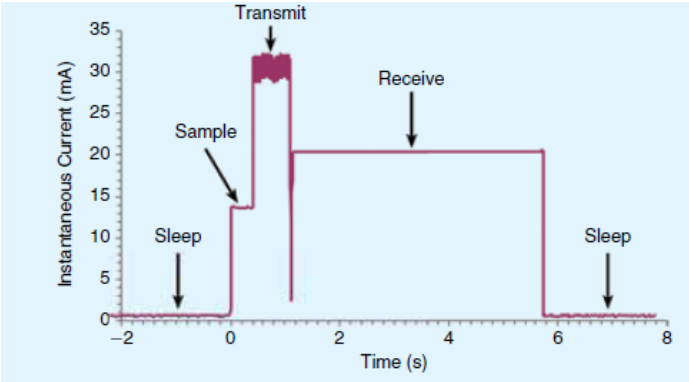

Figure 10.3. Instantaneous current measured on a battery powered system.

Reference Kathryn Loeffler, John E. Porterfield, Ph.D., Erik R. Larson, Ph.D., Daniel Escobedo, G. Patricia Escobar, D.V.M., John A. Pearce, Ph.D., Marc D. Feldman, M.D., and Jonathan W. Valvano, Ph.D., Embedded Medical Devices: Pressure Volume Loops in Rodents, IEEE Potentials, November/December 2012.

Observation: Being able to dynamically control bus frequency and I/O devices is important for low-power design.