Simple Power Management Techniques Using the Hamstack CPU Board

J. S. Best
June 1, 2012

This note is the result of discussions with Shel, KF0UR, as he is developing an application to display decoded CW and PSK data from a KX3. These techniques allow for full performance Hamstack applications using 10 – 20 mA of power supply current.

The Microchip PIC processors include many options for minimizing power consumption for battery powered applications. In this note, we apply some very simple techniques which do not require a lot of special code to dynamically adjust clock rates. The power savings are still quite significant. The current required for a simple application that has serial ports enabled, keeps track of time, and displays the result on a character LCD is only about 10mA.

This note is based on an example developed using the Hamstack C library. The same techniques can easily be applied to applications developed using Swordfish Basic.

Experiments

The example here is based on the user_code_gpsclock.c example application supplied with the Hamstack C library. This application displays date and time on a 2 line by 16 character LCD display, updating the time every second. The time is maintained by the millisecond interval timer interrupt provided by the Hamstack C library. It is initialized by either setting the time using a simple command sent via the serial port, by a Garmin GPS18 attached to the second serial port, or by a Dallas/Maxim DS1307 real time clock.

The CPU board was initially attached to a Hamstack Project board, which was used primarily for the convenience of the purely passive DB9 serial port connector and LCD connector. The project board also includes a switching regulator to supply the 5V for operation of the components.

Powering the board with the 78L05 regulator on the CPU board enabled resulted in a current draw of 54.5 mA without the GPS attached. The GPS adds an additional 60 mA to this. Experiments showed that the switching regulator was adding 9 mA to the current draw. The numbers below are the values that are achieved with the LCD attached directly to the CPU board without the extra 9 mA from the swithing regulator.

The LCD backlight LCD was disabled by clipping one of the leads on the ¼ watt current limit resistor on the LCD interface board. The power LCED was disable by clipping one of the LED leads.

Derived values (measured with attachments through project board, with 9mA from switching regulator subtracted):

Clock	CPU Pwr LED	LCD backlight	PicKit 3	IDLE sleep and peripherals off	Current
64 MHz	On	On		No	52 mA
64 MHz	On	On	Attached	No	60 mA
64 MHz	On	On		Yes	45 mA
64 MHz	On	On	Attached	Yes	53 mA
64 MHz	Off	Off		No	21.5 mA
64 MHz	Off	Off		Yes	15.0 mA
16 MHz	On	On		Yes	41 mA
16 MHz	On	Off		Yes	24 mA
16 MHz	Off	Off		Yes	11 mA
16 Mhz	Off	Off		No	12 mA

Here are the savings or adders for the various elements:

64 vs. 16 MHz w/o IDLE sleep & peripherals off	9 mA
64 vs. 16 MHz w IDLE sleep & peripherals off	4 mA
IDLE sleep at 64 MHz	6.5 mA
IDLE sleep at 16 MHz	1.5 mA
LCD backlight	17 mA
CPU power LED backlight	13 mA
LCD without backlight	2 mA
64 MHz processor alone w/o management	19.5 mA
16 MHz processor alone w/o management	10.1 mA

At 16 Mhz, the gain from the IDLE sleep is smaller because the timer interrupt takes up a significantly larger fraction of the total available CPU bandwidth.

The IDLE sleep and periperals disable provide similar gains. The gains are not additive. Roughly 2/3 of the gain shown above is achieved with either alone.

Gains from IDLE sleep are smaller with tone interrupts turned on, as is expected

Conclusion

The big power hogs are LED's. The biggest benefit comes from turning off the CPU power LED and the LCD backlight LED.

The second largest contributor is processor clock speed. Dropping for 64MHz to 16MHz cuts the power by a factor of two, from roughly 20 to roughly 10 mA.

The simple IDLE sleep and peripheral disable does provide a significant benefit of a 30% reduction (down by 6.5 mA) when using a 64 MHz clock.

Code

The following lines were added to user_init() to disable unused peripherals:

```
// Shut down unused stuff for power management
PMD0 = 0x3C;  // disable Timers 3..6
//PMD0 = 0x3E;  // disable Timers 2..6
PMD1 = 0x9C;  // disable CCP 3,4,5, i2c2
PMD2 = 0x0F;  // disable CTMU, comparators, A/D
// PMD2 = 0x0E  // disable CTMU, comparators
```

The following lines were added to user_code() to put the processor into IDLE mode until an interrupt comes along:

```
// Go to idle mode to save power.  Loop will run around once
// each time that timer or uart interrupt comes in
OSCCONbits.IDLEN = 1;
Sleep();
```

To change to a 16MHz clock, the 4X PLL most be turned off:

In *config.c*, change

```
#pragma config PLLCFG=ON
```

In *user_options.h*, removed the comments from the clock frequency line, and change to

```
#define CLOCKFREQ 16000000L
```

If the clock crystal is not 16 MHZ, the CLOCKFREQ constant should be changed to match.