

PIC14000

copyright, Peter H Anderson, Baltimore, MD, Oct 2, '03

Introduction.

Routines for the PIC14000 were developed in June, '02 in a small cabin on a lake in northern Vermont. Sounds nice. My development tools consisted of a laptop, a PIC Start Plus, three PIC14000 EEPROMs, a uV eraser and an HP Logic Dart. I don't know if the EEPROMs I had were exceptions, but it took up to two hours to erase a device.

The result was a set of routines, all of which were tested at the time but unlike other PICs, I simply never had the massive amount of time to go back and clean up each routine. Thus, I procrastinated and as the PIC14000 is very different from other PICs, I forgot just about everything I knew about it. In addition, there just are not that many people who are all that eager to use the PIC14000 and thus no great incentive to resurrect these routines.

However, after some 14 months, I have decided to distribute these routines. Aside from the program description at the head of each program, no changes have been made to the code.

To my knowledge, there is no emulator for the PIC14000 and thus the design process was pretty primitive. Carefully write the code with many printf statements for debugging, burn the PIC, observe, and then erase the PIC. Even with three PICs, the two hour delay in erasing the PICs greatly slowed the effort.

Files.

All special function registers and the bits within those registers are defined in "defs_14.h". As with all previous efforts, the names of each register is all upper case and the bits within the registers are lower case.

General timing and bit banded serial routines are contained in "delay.c" and "ser_14.c". The associated header files are "delay.h" and "ser_14.h". Note that I was using a 9600 baud serial LCD and thus "ser_14.c" includes various routines to position a cursor which would not normally be used when communicating with a terminal.

Constants are stored in each PIC14000 which would of course be lost after I erased the device. Therefore, program "rd_cal1.c" was run on each of the three PICs I had. This routine displays each constant as four hex bytes and in floating point format. I then recorded each of these values and developed program "rd_cal2.c" which enabled me to #define which CERDIP I was using and fetch the appropriate constants. Note that the CCS compiler provides a function which reads a calibration byte; read_calibration(). This was one of those rare times when I opted to use the CCS function rather than develop my own.

Measurement capabilities are illustrated in "admeas_1.c" and "lo_meas.c". Use of the pushbutton feature is illustrated in "push_btn.c". Use of the voltage reference is illustrated in "v_ref_1.c" and "v_ref_2.c".

I spent a good deal of time trying to use the PIC14000 as an I2C slave. I used the www.basicx.com BX24 as the I2C master using routines "i2c_3.bas" and "i2c_4.bas". The low level I2C routines for the BX24 are in file "i2c_bx24.bas".

Routine "i2c_slv3.c" was used to test the use of the PIC14000 as a slave to flash an LED a specified number of times. Routine "i2c_slv4.c" was used to either flash an LED a specified number of times or to perform an A/D measurement and return the result to the BX24. As I recall, both routines worked but I was troubled that any I2C slave is a rather complex state machine and one must always have a technique for returning to a home state.

