Getting a Start on the hid Project in Code Warrior 10

May 26, 2015

Introduction:

In past years, for EE342, we have been using Freescale's Code Warrior for Microcontrollers 6.3 (CW6) running under Windows XP. As a part of laboratory computer replacement, we need to transition to Windows 7 and, consequently, Code Warrior for Microcontrollers 10 (CW10). It's not an easy transition. There are particularly problems when the JM series microcontrollers, in particular the MCF51JM128 "Coldfire" processor is used. The purpose of this paper is to serve as a guide to porting the CW6 Human Interfaces Device (hid) demonstration to CW10. This particular demonstration is important because it is used as a departure point for student projects in EE342.

About Code Warrior 10:

There are many differences between Code warrior 6.x and Code Warrior 10. Many of these have to do with navigation, build techniques, and other details which are more or less incidental. However, there are a number of important changes that need to be mentioned before getting into the details of the hid demonstration proper. The CW features of note are:

1. The project files that direct the linker are different. In CW6, the files "Project_flash.lcf" and "Project_ram.lcf" define particular parts of memory and direct the loading of code, the stack etc. to certain addresses. For CW10, there is one file, "Project.lcf" which serves both purposes for the CW6 files. However, a couple of symbols were left out, and need to be added to get USB to work.

2. The interrupt files and interrupt service routine (isr) designation is different. In CW6, to add an isr, one need only use the key word "interrupt" and add the interrupt number before giving the isr function. In CW10, the isr looks just like any other function, but is preceded by "#pragma TRAP_PROC" which designates it as an isr. Furthermore, the file "exceptions.c" must be modified to designate the isr as an external function, then one must add the isr to the interrupt table in place of the default handler "asm_exception_handler". This isn't so difficult; it's just more complicated.

3. CW10 and its debugger include a very useful feature: a "printf" capability that allows the 'JM's second serial port (which is connected to the debug port) to print messages to a "console" within the debugger. This is independent of anything being done with the "terminal" (using USB or the first serial port). This debug facility can also support input from the console. To enable this capability, choose to designate console support when creating the project. That includes the file "console_io_cf.c" in the project. It is also necessary to flip a bit near the beginning of "exceptions.c" to read "#define NO_PRINTF 0" (rather than 1). In any file that you want to print from, typically main.c, include the header file "<stdio.h>". Now, all that is needed to print out a message is "printf("control string",variables). Very handy indeed. The slow speed of this port is such that prints need to be used carefully. However, if you are using printf, the microcontroller will only work properly under the debugger. Once code is debugged, you need to comment out all of the printf statements and change back to "#define NO_PRINTF 1" in exceptions.c.

4. In CW6, the project structure seen in the project view of the Integrated Development Environment (IDE) was independent of where the files actually were. This caused confusion. CW10 is more straightforward. Code files (.c or .cpp or .asm) that are in the "Sources" folder are in the project, without a separate step to "add" them. Header files (.h) in the "Project_Headers" folder are likewise included in the project. (Be sure you don't put header files in the Sources folder; they won't be found.)

Porting hid_demo:

Unfortunately, CW10 will not open (and update) a CW6 project automatically. There are enough differences that this step-by-step guide was thought necessary.

1. Copy all of the files that are needed for the project from the various folders within the "CMXUSB_LITE_V1" folder (usually installed on C:) that are needed for the project into the Project Headers or Sources folder as appropriate. These are: "hcc_types.h", "hid_generic.h", hid_kbd.h", "hid_mouse.h", "hid_usb_config.h", "hid.h", "mcf51xx_reg.h" (which contains definitions needed for usb), "target.h", "usb_config.h", ""usb.h", "hid_generic.c", "hid_kbd.c", "hid_mouse.c", "hid_usb_config.c", "hid.c", "mcf5xxx_lo.s", "target.c", and "usb.c". (I normally like to consolidate all files needed for a given project in the project folders proper anyway, even in CW6. To do otherwise risks modifying a file that may be in use by a different project. Think of it as simplified configuration control.)

2. In all of the files, references to header files need to be "flattened". That is, where in the original directory structure for CW6 the files referred to headers in other folders, now all of these will be local. So, modify all references in "#include" statements to be simply of the form "file_name.h".

3. In CW6 reference is made to the standard MCF51xx registers using an include of "mcf51jm128.h". For CW10 all of these must be changed to <mcf51jm128.h>. For some files, notably "usb.c" and perhaps others, "#include "mcf51JM_reg.h" is also needed.

4. The main program for the CW hid_demo is "hid_main.c". You can copy its code into the new "main.c" that is created when the new hid project is created. Header files need to be included are "mcf51xx_reg.h", "target.h", "usb.h", "hid_mouse.h", "hid_kbd.h", and "hid_generic.h".

5. Modify the new "Project.lcf" file by adding needed references (for usb.c) to the symbols "_IPSBAR" and "_BDT_BASE" as follows:

a. after "___FLASH_SIZE = 0x00020000;" add:

```
__IPSBAR = 0xFFFF8000;
```

b. after "___stack_safety = 16; / . = ALIGN (0x4);" add:

```
. = ALIGN (512);
__BDT_BASE = .;
. = . + 512;
__BDT_END = .;
. = ALIGN(512);
```

6. In some cases (but not others) I experienced a linker error or warning and error or just a warning: “All segments to userram must begin at address aligned to 512. Please add ‘. = ALIGN(512);’.” This was a tough one to track down. In the linker file “Project.lcf”, by adding “ALIGN(512);” at the END of the .data and .bss segment declarations the error / warning goes away. Out of the box these segments end with “.=ALIGN(0x4);”. Change to 0x4 to 512. For a new project this wasn’t a problem, but it popped up eventually.

7. There is an unresolved reference to the function void stacksize(void) in hid_mouse.c. That needs to be fixed by adding an #include “target.h” at the beginning under the other includes.

8. Fix the bug in “target.h” which incorrectly identifies switch 2:

“#define SW2_ACTIVE ((PTGD_PTGD1)==0)”

(Note that there is another bug somewhere that causes the LED order to be wrong when running the hid_generic demo. The buttons for LED’s 1 and 2 actually control 3 and 4, and vice versa.)

9. In the header file “usb_config.h” you may be getting errors due to things not being defined. Simply comment out everything except “#include “hid_usb_config.h””.

10. There are a number of places where CW generates warnings that no prototype is defined for a function. This will be particularly true for isr’s. The warnings can be easily suppressed by adding declaration statements near the beginning of the various files.

11. The linker handles stack initialization under CW10. The default is a generous 0x400 bytes. You should not need any stack initialization in main() or hw_init() copied over from CW6. I recommend getting rid of the “stack_init(0x88);” in main(). Stack overrun is an often encountered problem when local variables of significant size are added to main or other code that stays resident. One form of obscure error is making usb not work for messages longer than some arbitrary limit. (If you move a variable array from local to global and the problem gooes away, you have a stack size problem.)

12. The Coldfire processor had a more complex interrupt priority system. A function that sets the priority needs to be declared. Target.c is the file where this needs to be defined. Add the declaration somewhere early in the file:
hcc_imask asm_set_ipl (hcc_imask);

Observations:

After getting the hid_demo to work in CW10, I think CW10 will be a superior environment thanks primarily to the availability of a “printf” debug capability, as well as a somewhat better debugger. It’s easier to follow the machine language and toggle back and forth between the C view and the debugger view. Variables are a bit easier to track, although chars are shown only in character mode. On the other hand, things overall are definitely more complicated, particularly exceptions, so the “entry cost” to use the tools and spin up is a bit higher. Needing to port projects from CW6 adds a hassle factor to development for EE342. Perhaps already ported versions of projects can be developed prior to the next class offering.