Memory Paging in the Axiom CML-9S12DP256 AXM – 0285 Revision F, using the ImageCraft ICC12 V6 Professional Version Compiler and the P&E Microsystems USB HCS08/HCS12 Multilink and Prog12Z software

Dave Werner and T.J. Colgan
Senior Design, Spring Semester 2005
University of Wyoming

To start with, the memory map for the CML-9S12DP256 produced by Axiom Manufacturing is shown in Figure 1. As you can see, the paged external Random Access Memory (RAM) lies in the range of $8000-BFFF as well as the paged Read Only Memory (ROM) resides in this address space. The next few paragraphs will go step by step into how we used the ICC12 compiler to access both the paged ROM and paged RAM available on this board.

ADDRESS	TYPE MEMORY	MEMORY APPLICATION
$C000 - $FFFF	FLASH	MON12, NOICE, and Utility firmware located in internal flash, Page $3F.
$8000 - $BFFF	External Ram	User Paged Program Memory space, pages $20 - $2E. Note: Pages $30 - $3F reside in the internal flash.
$4000 - $7FFF	External Ram	User Program Memory, emulate fixed page $3E.
$3F8C - $3FFD	Internal Ram	Ram Interrupt Vector Table
$3E00 - $3F8B	Internal Ram	Monitor reserved ram memory. Stacks and variables.
$1000 - $3DFF	Internal Ram	User Internal Ram memory
$0400 - $0FEB	Internal EEprom	User EEprom memory, Monitor reserves $FEC - $FEF for Autostart, user should avoid $FF0 - $FFF memory use.
$0000 - $03FF	HCS12 Registers	Monitor or user access to control registers.

Figure 1: Axiom CML-9S12DP256 Memory Map

To start with we had to install several jumpers on the Axiom board. The MEM_EN jumper and the ECS jumper both had to be installed to activate the paged memory functions of the microprocessor. Also, the MODC jumper needed to be open or idle for correct communication from with the processor. The version of MON12 that came with our board needed to be updated to allow memory paging as well. This update (CML12update2p7f.zip) is freely available from Axiom, and is specifically designed to enable paged memory on the CML12 board rev F.

Next, we used the ImageCraft ICC12 compiler to take our C code and convert it to the proper Motorola .s19 file to be downloaded into the microprocessor. To set up the compiler correctly for memory paging we had to change some of the compiler options that can be found under the Project pull down menu, then Options. This will bring up the window labeled "Compiler Options," and under the Target tab we had to make the following settings. For the

case of RAM we set the Device Configuration option to be Custom and we configured the rest of the settings in the following way. The Program Memory was set to 0x1000, which placed our 'main' routine in the internal RAM of the microprocessor. Next, we set the Data Memory field to be blank so the compiler would place the data memory directly after the program memory in internal RAM. The Stack Pointer field was set to 0x3DFF, because this was before the RAM interrupt vector table of the MON12 monitor supplied with the microprocessor. We were able to cram our 'main' routine and our data all before the stack in internal RAM because the majority of our code was broken up into functions. However, we could have used the space 0x4000-0x7FFF for storing some of this information as well.

Under the Expanded Memory subsection of the Target tab, we also had to select the Enable, and Make Paged Functions Default boxes. Then in the Addr box we had to put 0x80000.0xBFFFF. The 0x80000 is calculated by taking the size of each page (16K or 16384 bits) and multiply that by the first page of external memory (shown as page 0x20 in Figure 1). The first page of ROM in this processor is at page 0x30 (shown in Figure1), so the start of ROM would lie at 16384 multiplied by 48 (0x30 in hexadecimal) which is 0xC0000. Consequently, paged RAM will last up to an address of 0xBFFFF which corresponds to the last number in the Addr field. The last settings we made on the Target tab of the Complier Options window, was we selected both the Linear and Map Vector Page under the S2 Record Type subsection. A screen shot of all these settings can be seen in Figure 2.

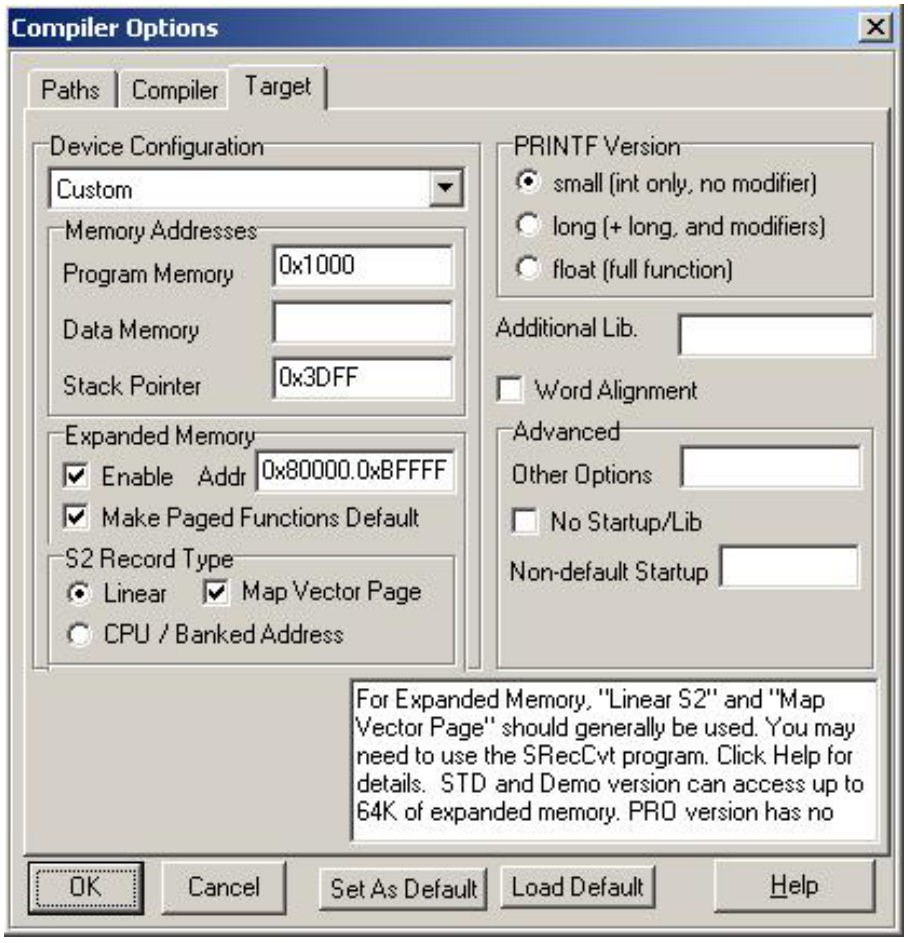

Figure 2: Compiler Options- Target Tab

Too avoid the “File Too Large to Fit in Code Page Window” we had to break up our program into several smaller files. To do this we generated a header file that contained all of our function prototypes and global variable definitions. We included that file in our “main” routine and broke our function definitions into three smaller files called bigstuff in the example below. Then with the ICC12 compiler we just had to place the files containing the function definitions into the ‘Files’ folder in our project. Since we never directly included any of these files in the project, any function or global variables used in these file had to be declared external through the use of the extern keyword. For example one of our function prototypes in our header file was void SCI_INT(void);, and when we wanted to use or define this function in one of the definition files, we had to declare at the top of the file extern void SCI_INIT(void);. Figure 3 demonstrates what the project setup should look like to break up one file into smaller files.

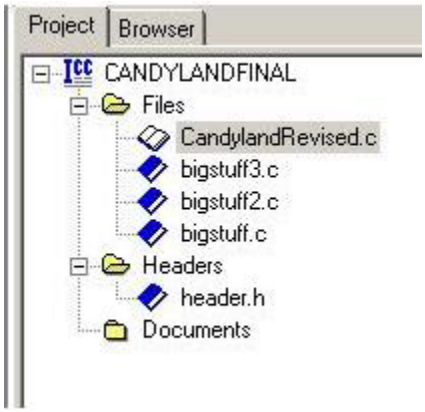

Figure 3: Project Setup in ICC12

On the Compiler tab of the Compiler Options window we also had to add the following setting. In the box labeled Execute Command After Successful Build we had to insert the following line “c:\icc12pro\bin\SRecCvt -m 00000 BFFFF 32 -lp -o %p.s2 %p.s19.” This setting is demonstrated in Figure 4. The path name would change depending on where you installed the copy of ICC12 but this command is designed to execute Motorola’s SRecCvt. This software is used to reformat the S record output of ICC12 to the correct format for memory paging in the M68HC12 family. According to the SRecCvt documentation the 0x00000 and 0xBFFFF values are supposed to correspond to the start and stop addresses of paged memory space and the 32 is supposed to correspond to the number of pages, but we selected these parameters to keep from getting the S-Record Data Not In Specified Memory Map error. Finally, once the S record conversion is complete, all that needs to be done is the .s2 file needs to be downloaded into the processor like any other file.

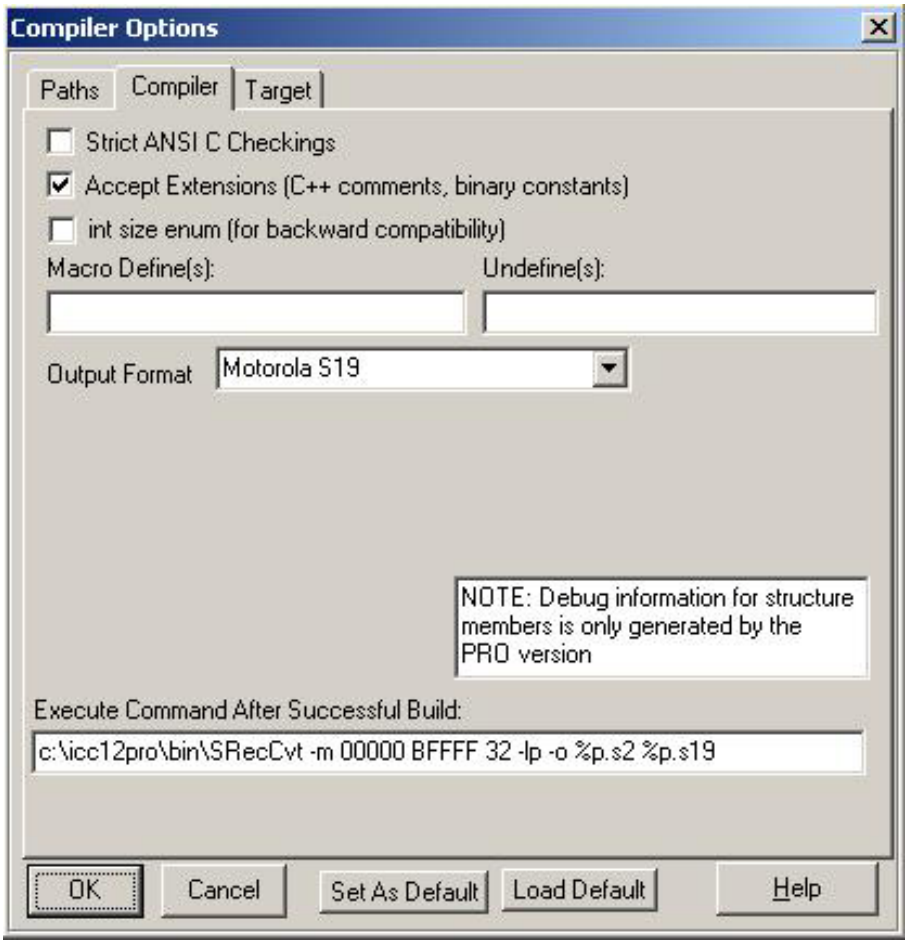

Figure 4: Compiler Options- Compiler Tab

Next, we also used the ImageCraft ICC12 compiler in conjunction with the P&E Microcomputer Systems' USB HCS08/HCS12 Multilink and PROG12Z FLASH/EEPROM Programmer software to program the paged ROM available on this processor. To start out with, we had to change all of the settings in the ICC12 compiler to correctly implement paged ROM. The first change we changed was the Program Memory on the Compiler Options menu (Target tab) was set to a value of 0xC0000, the Data Memory to 0x1000, as well as the Stack Pointer to a value of 0x3DFF. This time the selection of the program memory had to do with interrupt vectors, which will be discussed in detail a little later. The Data Memory needs to be in RAM along with the Stack Pointer, because these are variables or structures that are going to be constantly changing while your code is running, so ROM is not a suitable choice for dynamic data. When downloading into ROM we actually did not perform the same conversion using the SRecCvt as we did with RAM and we didn't even use the converter that comes with the P&E software. Instead we directly used the .s19 file in conjunction with the PROG12Z FLASH/EEPROM Programmer software. The settings we used in ICC12 for downloading into ROM can be found in Figure 5.

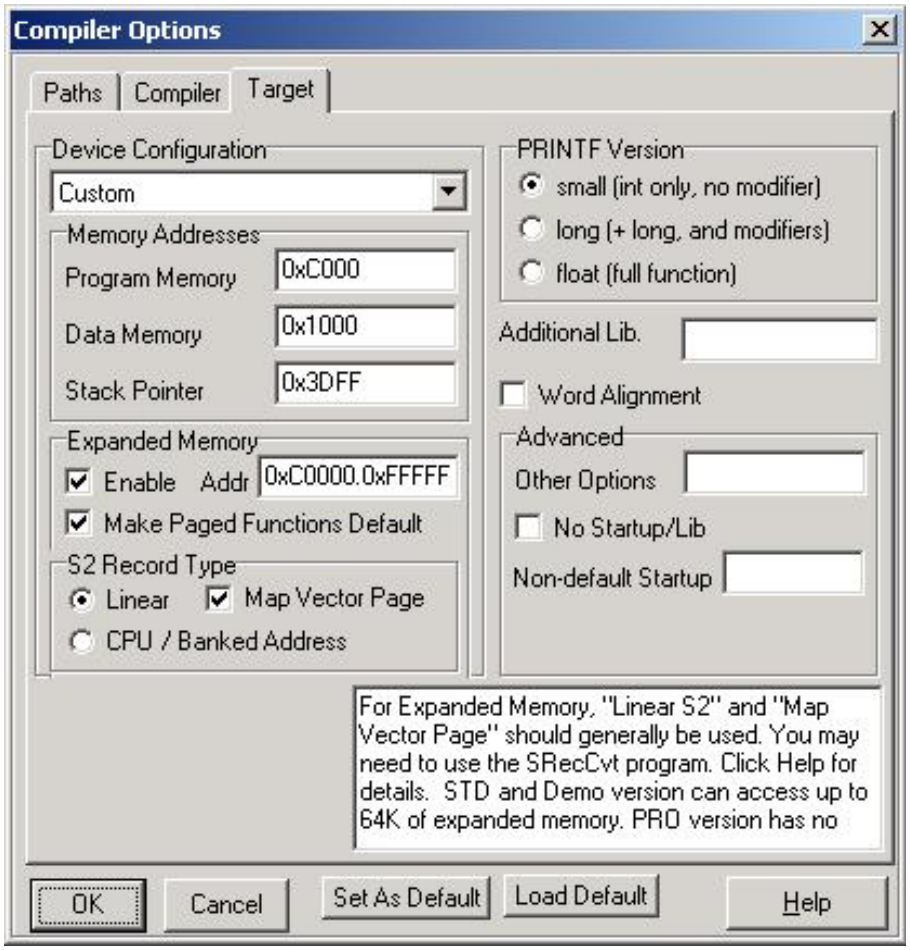

Figure 5: Compiler Options- Target tab (ROM)

To correctly implement our RAM code into ROM correctly, we had to add one line of code and change the declarations of our interrupts. The line of code we had to add was:

```
asm(    ".area vector(abs)\n"
        ".org 0xFFE8\n"
        ".word _toggle3_isr\n"
        ".word _toggle2_isr\n"
        ".word _toggle1_isr\n"
        ".word _toggle0_isr\n"
        ".org 0xFFF8\n"
        ".word 0xC000, 0xC000, 0xC000, 0xC000\n"
        ".text" );
```

The asm( ) command of ICC12 is used to insert inline assembly into the C code. The first .org statement and 4 subsequent .word statements are used to place the interrupts service routines in the ROM interrupt vector map. Our interrupts were for the first four timer channels and the address of these interrupts started at 0xFFE8, we had to delete any “pragma abs_address:” commands that were used to set the RAM interrupt vector table. We still had to keep any “pragma interrupt_handler” commands that were used to define interrupt service routines in ICC12. The next .org statement and the following .word statement is used to place the starting address of our code into the Reset interrupt vector address as well as the Unimplemented Instruction Trap interrupt vector address. The effect of this statement is to ensure that whenever the processor first starts up, when the reset button is pressed, or if an unimplemented instruction trap occurs the processor will move to the start of our code at address 0xC000.

The starting address of 0xC000 was chosen because according to the P&E Microcomputer Systems' frequently asked questions section, using interrupts on top of paged memory makes things even more complicated. Since you never know when an interrupt is going to occur, you will not know what value is in the PPAGE register (key register in doing paged memory) when an interrupt is called. Therefore, if your interrupt service routine is located in paged memory, and considering the fact that you never know what page the microprocessor is currently looking at when the interrupt occurs, you probably won't be on the right page for accessing your interrupt service routine. Consequently, interrupt service routines are usually stored in non-paged memory, like 0xC000 in our case that holds both our interrupt service routines and our 'main' routine.

To use the P&E Microcomputer Systems' USB HCS08/HCS12 Multilink and PROG12Z FLASH/EEPROM Programmer software we first had to pick a module file for this software package. We picked the 9S12DP256_256K.12P file because it closely resembled the paged memory system of the Axiom board. The base address of this file is 0x0C0000. After we let the auto-detect option of the software acquire the microprocessor we then selected the module file mentioned above. Next we had to specify the s-record for the software to use, and once again we used the .s19 file directly from the output of ICC12, no converter algorithms applied to it. Then we simply erased and programmed the module and our code was successfully downloaded. Here is a picture demonstrating the setup we used in the PROG12Z software.

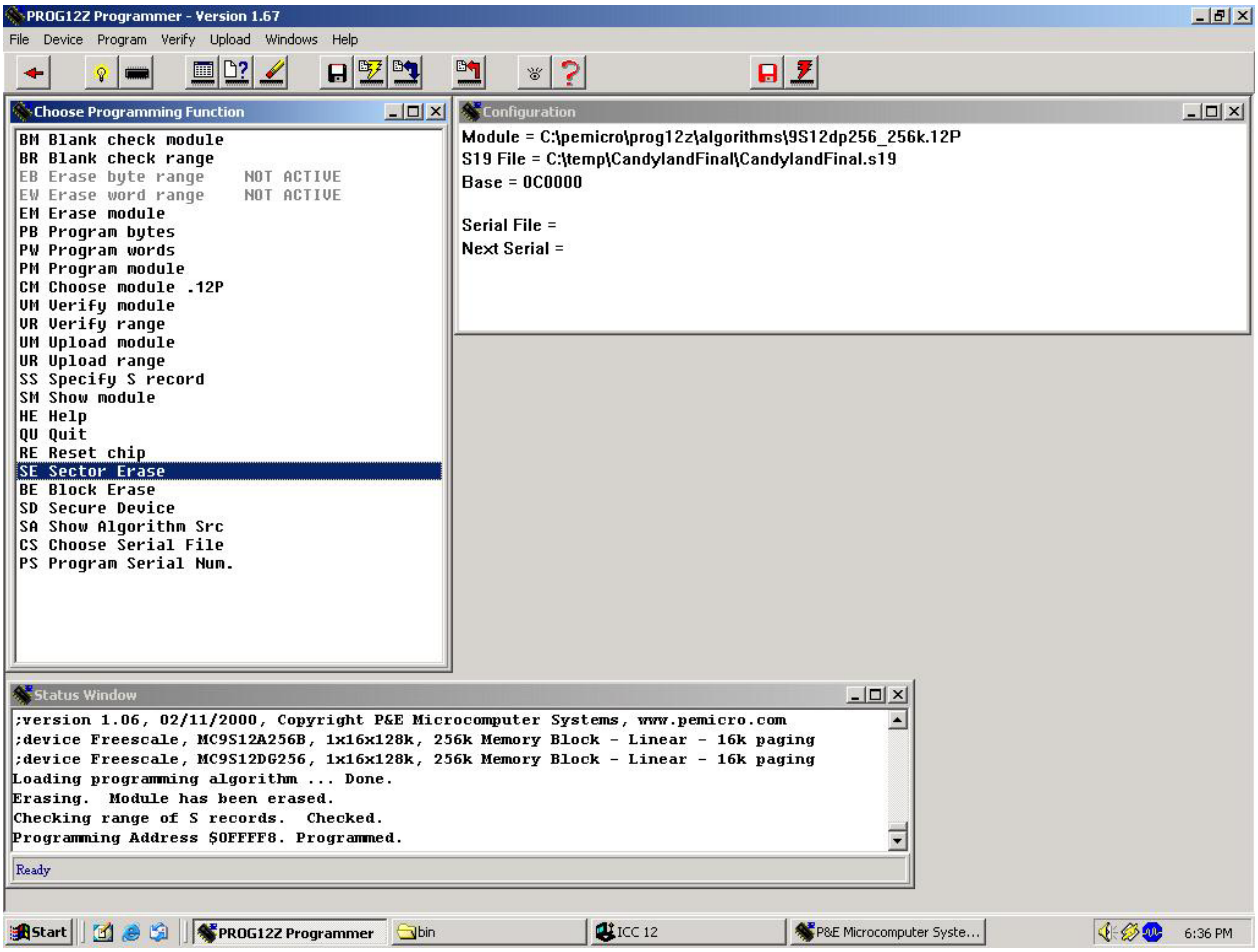

Figure 6: PROG12Z software setup