Technical Note TN268

Converting Projects to CodeWarrior ColdFire V7.2

Project Conversion

Converting a project created in CodeWarrior Development Studio for ColdFire Architectures V7.1 to V7.2 is a simple procedure. One major change in V7.2 may impact projects when moving from V7.1 to V7.2. The Main Standard Libraries (MSL) has been replaced by the Embedded Warrior Libraries (EWL). See Libraries section for more information on EWL.

This document provides a general overview of the changes in V7.2 and a quick guide to successfully migrate projects from V7.1 to V7.2.

This document contains these topics:

Terms and Abbreviations

The following terms and abbreviations are used in this document:

- MSL - Main Standard Libraries
- EWL - Embedded Warrior Libraries
- V7.1 - Refers to CodeWarrior Development Studio for ColdFire Architectures, Version 7.1.
- V7.2 - Refers to CodeWarrior Development Studio for ColdFire Architectures, Version 7.2.

Libraries

Embedded Warrior Libraries (EWL) introduces a new library set aiming at reducing the memory footprint taken by IO operations and introduces a simpler memory allocator. The IO operations are divided in three categories: printing, scanning and file operations.

The printing and scanning formatters for EWL are grouped in an effort to provide only the support required for the application:

```
int - integer and string processing
int_FP - integer, string and floating point
int_LL - integer (including long long) and string
int_FP_LL - all but wide chars
c9x - all including wide char
```

The buffered IO can be replaced by raw IO, this works solely when printf and scanf are used to perform IO, all buffering is bypassed and writing direct to the device is used. EWL libraries contain prebuilt versions for all formatters and IO modes. Selecting a model combination enables correct compiling and linking. The EWL layout for ColdFire is built per core architecture. It is composed of:

`libm.a` - math support (c9x or not)

`libc.a` - non c9x std C libs

`libc99.a` - c9x libs

`librt.a` - runtime libraries

`libc++.a` - non-c9x matching c++ libraries

`libstdc++.a` - c9x/c++ compliant libs

`fp_coldfire.a` - FPU emulation libraries

Selecting an EWL model for the libraries frees the user from adding libraries to the project, the linker will determine from the settings the correct library set, these settings are: processor, pid/pic, hard/soft FPU. The process of selecting a model is explained in the Librarian section below. Although the library names are known to the toolset their location is not.

Access Include Paths

The "System Access Paths" point to code in MSL. These have to be changed to point to EWL. The new system access paths are (compiler relative & recursive)…

```
{Compiler}\ColdFire_Support\ewl\EWL_C
{Compiler}\ColdFire_Support\ewl\EWL_C++
{Compiler}\ColdFire_Support\ewl\EWL_Runtime
{Compiler}\ColdFire_Support\ewl\lib
```

NOTE These access path changes are done automatically during Project conversion by the IDE, when opening a project that was built using V7.1.

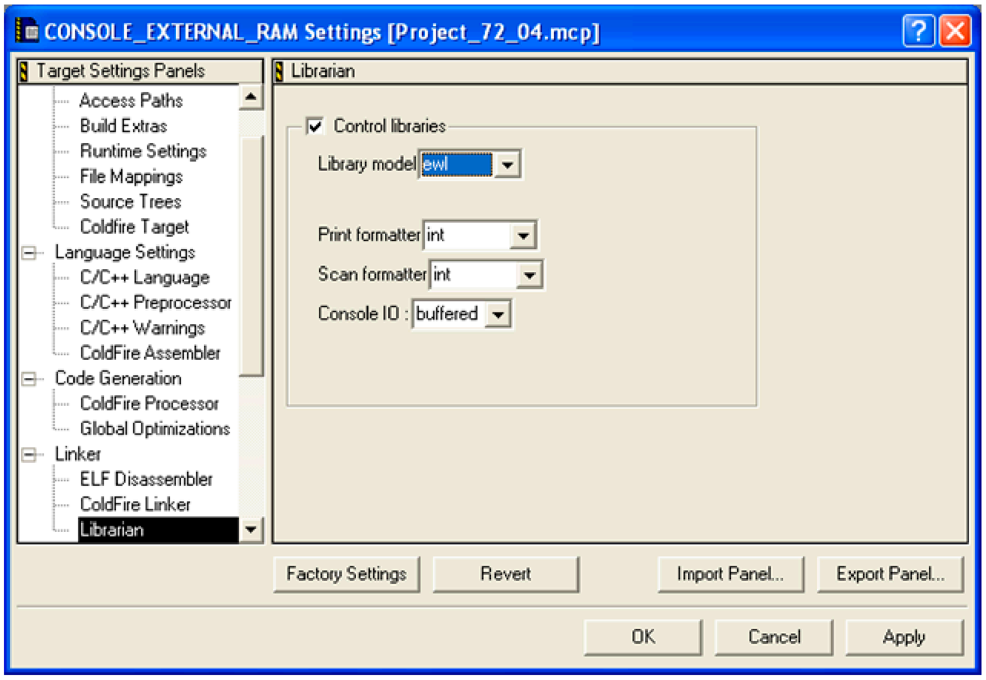
Librarian

The Librarian panel allows the user to select a "Library model" from a pre-defined list of available models. The lists of available models are:

```
"ewl"
"ewl_c++"
"c9x"
"c9x_c++"
```

The "ewl" and "ewl_c++" have a smaller memory footprint, while the "c9x" and "c9x_c++" models are fully C99 compliant. The "ewl' and "ewl_c++" models have relevant sub-models that allow the user to pick the desired print and scan formatters and the desired IO scheme. The "c9x" and "c9x_c++" models do not have sub-models.

Figure 1.1 Console External RAM Settings - Librarian


CONSOLE_EXTERNAL_RAM Settings [Project_72_04.mcp]

Librarian

☑ Control libraries

Library model: ewl

Print formatter: int

Scan formatter: int

Console IO : buffered

For the print and scan sub-models the available choices and the functionality that they cover are listed in the "Libraries" section above.

When choosing the "c9x" or the "c9x_c++" models, the sub-model drop-downs are disabled, as they do not apply to these models.

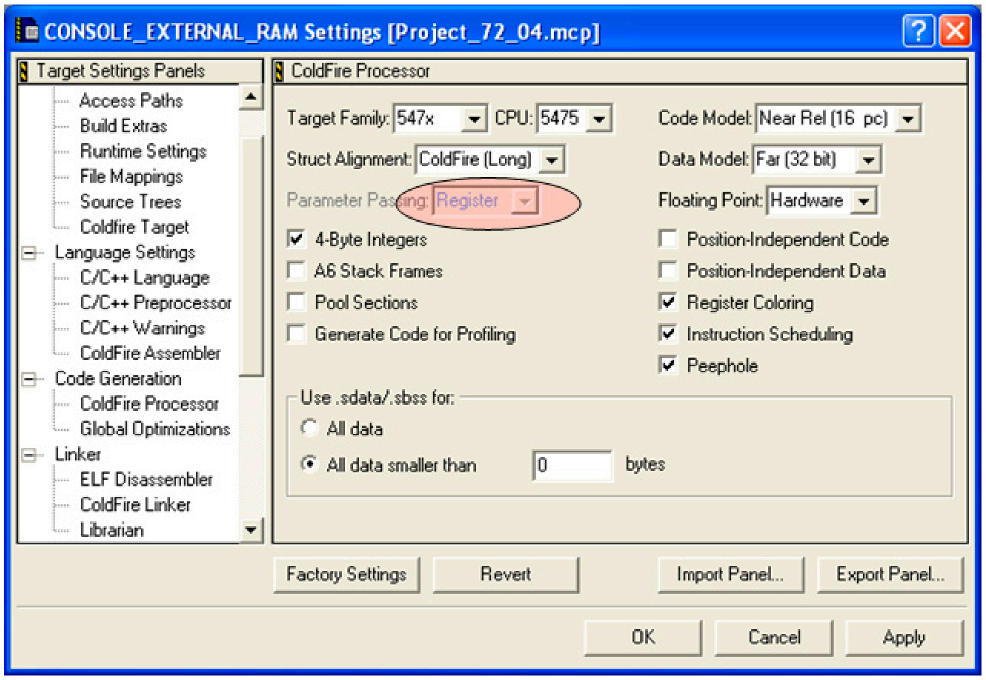
The "Control libraries" check-box, determines whether or not the EWL mechanism of library selection will be used by the build tools (compiler and linker). When this check-box is not selected, the user has to manually add the required library files to the project. The user can select the relevant library files by choosing the correct architecture (v2, v3 or v4) and whether or not FPU and PIC/PID is used.

NOTE When opening a V7.1 project using V7.2 the converter, chooses the "ewl" model with the print and scan formatter set to "int_FP" and "Console IO" set to "buffered".

Parameter Calling Convention

The parameter passing affects space and time performance. The best performance for both occurs when selecting the register passing ABI. The default parameter passing convention is Register, and it can not be changed from the "ColdFire Processor" panel.

Figure 1.2 Console External RAM Settings - ColdFire Processor



Target Family: 547x CPU: 5475 Code Model: Near Rel (16 pc)
Struct Alignment: ColdFire (Long) Data Model: Far (32 bit)
Parameter Passing: Register Floating Point: Hardware

- [✓] 4-Byte Integers
- [ ] A6 Stack Frames
- [ ] Pool Sections
- [ ] Generate Code for Profiling
- [ ] Position-Independent Code
- [ ] Position-Independent Data
- [✓] Register Coloring
- [✓] Instruction Scheduling
- [✓] Peephole

Use .sdata/.sbss for:
- ○ All data
- ● All data smaller than 0 bytes

NOTE When using the V7.2 product the "Parameter Passing" drop down, in the "ColdFire Processor" panel is disabled and fixed to "Register".

Other calling conventions are available through one of the following methods:

1. Use declspec for function prototypes, also described in the “Declaration Specifications” section of the *ColdFire Architectures Edition Build Tools Reference v7.x*:

   ```
   asm void __declspec(compact_abi) check_CC(unsigned long)
   {
       ....
   }
   ```

2. Use `pragma` to specify the calling convention for function defined from:

   ```
   #pragma compact_abi
   asm void check_CC(unsigned long)
   {
      ....
   }
   ```

Assembly Function Declarations and Definitions

For all functions in the application that are pure assembly functions, the function definition and declaration(s) should contain a "declspec" qualifier that defines the parameter passing convention. For example,

```
asm void __declspec(register_abi) TrapHandler_printf(void)
```

Without this declspec, there is a warning

WARNING! “possible abi conflict, use `__declspec`(register_abi):” generated by the compiler for all such assembly only functions.

Please note while converting V7.1 projects to V7.2 the user has to modify their code such that the assembly functions contain the "`__declspec`" qualifier. Also if the function contains code that assumes a different calling convention, and is called from a "C" function, for example,

```
asm void mcf5xxx_wr_vbr(unsigned long) { /* Set VBR */
move.14(SP),D0
  movec d0,VBR
nop
rts
}
```

The code should be modified to use the "Register" parameter passing convention. In this example, in addition to adding the "`declspec`" mentioned above, the line `move.14(SP),D0` must be removed.

EWL Memory Allocation Scheme

EWL supports an improved memory allocation scheme. The memory allocation scheme in EWL requires the following symbols to be defined in the LCF file: `___mem_limit` and `___stack_safety`. `__mem_limit` signifies an address beyond which memory cannot be allocated. `__stack_safety` is the size of the cushion between the stack and the heap.

In the example below, it is set to 16 bytes, which is typical.

```
___mem_limit = ___HEAP_END;
___stack_safety = 16;
```

These symbols can be added to the LCF file right after the definition of `___HEAP_END`

Additional Information

See the CodeWarrior Development Studio for ColdFire Architectures V7.2 Release Notes and documentation for more information.

Visit `http://www.freescale.com/support` for additional assistance.

Document Number: TN268 Rev. C

14 May 2010