

1. INTRODUCTION

In many instances customers need to migrate their existing DSP projects from the Motorola GNU based tools to the TASKING toolchain. In some cases, it is desirable to create a Motorola COFF object that can be debugged with the Motorola debuggers. This migration guide provides an approach for these processes. Details of the differences between the toolchains and how to solve them are supplied in a separate paragraph. It is assumed readers of this document will be making use of EDE although with minor effort it should be of good use for command line users as well. Paragraphs 1 through 3 contain generalized information and have been added as a setup to paragraph 4 which deals with an actual example available under download *migrate.zip* from the TASKING website (www.tasking.com/support/DSP56xxx/apnotes.html). If these introductory paragraphs give rise to questions they should at some point be answered when working out paragraph 4. To be able to faultlessly work through this document you MUST make sure your system meets the following conditions:

- You have downloaded *migrate.zip* and explicitly installed it in **c:\migrate**
- The TASKING tools reside in **c:\program files\altium\c563\v3.5r1**

- The DSPLOC environmental variable has been properly set, e.g. **c:\progra~1\motorola\dsp**
- The PATH environmental variable MUST include the Motorola GNU based tools

Paragraph 4 differentiates between two toolchains which can both be selected from EDE: **Project | Select toolchain**. These toolchains are important to remember and have been listed below:

- *TASKING C/C++ for DSP563xx v3.5r1*
- *TASKING C/C++ for DSP563xx v3.5r1 with Motorola tools*

The first toolchain is the one you would normally use. It solely consists of TASKING tools. The latter one integrates a mixture of TASKING and Motorola tools. This is also the reason why Motorola tools MUST be included in the PATH environmental variable. Throughout this document MUST and MUST NOT will be capitalized in order to indicate that when ignored to all likeability the concerning topic will not work. It is further assumed that **Project Options** is the shorthand notation of the main project properties plain **Project | Project Options**.

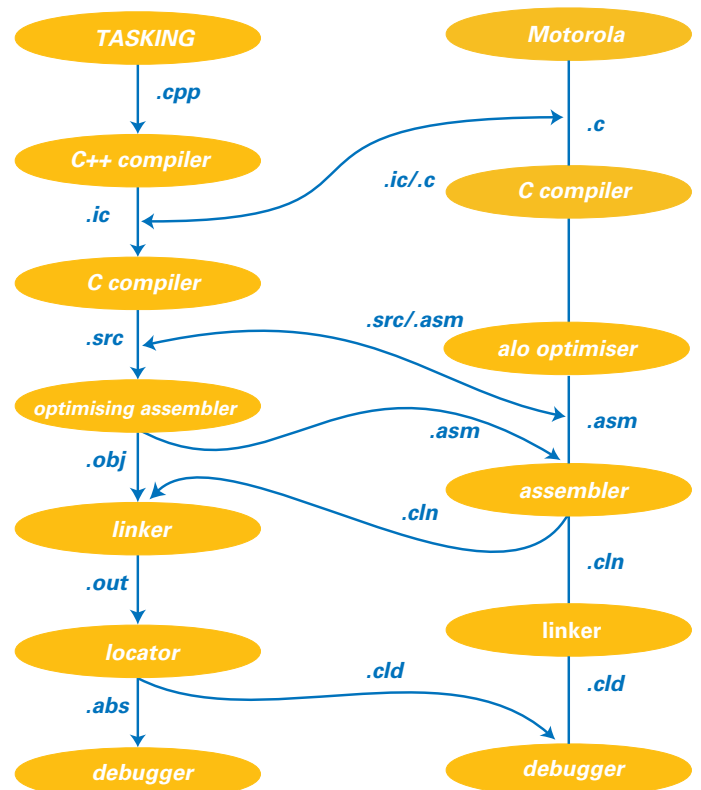
2. INTEROPERABILITY PATHS BETWEEN MOTOROLA AND TASKING TOOLCHAINS

The interconnections between the TASKING and the Motorola tools are depicted in the diagram below. Both toolchains can process C and assembly sources with few restrictions. The TASKING toolchain can read the Motorola libraries and object files. In addition, the compiler can create assembly source files with COFF debug information which can be processed by the Motorola tools. The TASKING assembler can output an optimized assembly source file for the Motorola assembler while taking advantage of the optimizations performed by our assembler.

2.1 Migrating C and assembler sources

The first step towards migration is to be certain that it compiles and operates correctly in the original environment. When this has been accomplished, the project can be moved to the TASKING environment on a file-by-file basis. Of course, it is possible to move all of the source files to the new environment in one go, but there is always a risk that the result will not work and errors will be difficult to track.

Typically the migration begins with a new project based on the *TASKING C/C++ for DSP563xx v3.5r1 with Motorola tools* to which all Motorola generated object files are added. The



project is then built and the resulting executable is checked to see if it still runs correctly. There should be no problems because in this first phase because the Motorola objects are immediately fed to the Motorola linker/locator resulting in the original - Motorola generated - program image.

In the second phase each Motorola generated object file is selectively removed from the EDE project and replaced with the source module from which it was generated. Rebuilding the project will now build this module using the TASKING compiler and assembler using Motorola compatible settings and the resulting optimised assembler is passed onto the Motorola assembler in the normal way. Before proceeding to the next module the new executable must be tested as before. Possible errors are limited to the new object file and are easy to track down. Repeat this process until all C and assembly sources are handled and compiled correctly. The resulting executable will, in general, be much smaller and faster than the original one.

NOTE

There is only one exception to the rule of replacing modules with intermittent tests. This is when floating points are used in combination with standard C library routines.

Since the TASKING tools and the Motorola GNU based tools use a different single precision floating point implementation there is no way a mixed project - a project containing a mixture of TASKING and Motorola objects - will work even if they are linked with both the TASKING and Motorola C-library. Even though such a project would link without errors it would speak two languages where floats are concerned possibly leading to runtime errors such as unexpected exceptions but with certainty all of its calculus will be wrong. For such projects the testing MUST be postponed until all modules have been migrated.

3. DEBUGGING A TASKING APPLICATION WITH THE MOTOROLA DEBUGGER

The *TASKING C/C++ for DSP563xx v3.5r1 with Motorola tools* requires the usage of the Motorola debugger since CrossView Pro cannot read Motorola COFF objects. Existing TASKING projects - build with the *TASKING C/C++ for DSP563xx v3.5r1 toolchain* - can be debugged with the Motorola debugger as well. This requires selecting the Motorola COFF format from **Project Options | Linker/Locator | Output Format**. Additionally the Motorola debugger MUST be entered as third party alternative from **Build | Options | Misc**. The executable will not contain HLL debug information as the locator does not support processing it for Motorola COFF. This requirement can only be met by using the *TASKING C/C++ for DSP563xx v3.5r1 with Motorola tools*. Usually there is no need to debug an existing TASKING project with the Motorola debugger as CrossView Pro can be used for this purpose.

4. EXAMPLE MIGRATION PROJECT

This paragraph explains that exact steps to take for migrating your Motorola GNU based tools project to TASKING. It makes

use of *migrate.zip* which can be downloaded from the DSP56xxx application notes page (www.tasking.com/support/DSP56xxx/apnotes.html). The application in this example computes the FFT of a fixed set of input data and prints the result values to the stdout output. The example has been set up for the DSP563xx, but will also work for the other supported DSPs with the appropriate changes. After unzipping the folder layout will be as follows:

<code>c:\migrate\src\motorola</code>	contains Motorola source modules
<code>c:\migrate\src\tasking</code>	contains TASKING migrated source modules
<code>c:\migrate\out\motorola</code>	used during true Motorola program file generation
<code>c:\migrate\out\mixed</code>	used during migration
<code>c:\migrate\out\tasking</code>	used during true TASKING program file generation

The first step towards migration is to launch a command shell and change directory to:

■ `c:\migrate\out\motorola`

From which the following batch MUST be executed:

■ `rebuild.bat`

Provided the conditions of the introduction are met the Motorola sources should build without errors. After completion the Motorola simulator MUST be invoked as listed below:

■ `sim56300 sim.cmd`

The simulator will output a file called *fft.txt* which MUST be compared to *fft_ref.txt* as stored in the Motorola source folder. For the Motorola GNU based tools this may seem like overkill since *fft_ref.txt* was made with those same tools but even then it may prove useful for detecting inconsistencies. The compare becomes of more interest when TASKING tools, first partially when using the *TASKING C/C++ for DSP563xx v3.5r1 with Motorola tools*, later on fully when using the *TASKING C/C++ for DSP563xx v3.5r1 toolchain*, come to replace the Motorola GNU based tools. At any point in time the results must be the same as with the original. A change is an indication that the most recent change - either a change in the original sources or a project setting - was most likely incorrect.

NOTE

Referring to the concluding note of section 2.1 this is one of those projects where floats are used.

*Normally this would indicate that all testing MUST be postponed until all modules have been migrated and linked against the TASKING C, runtime- and floating point libraries. This example allows a compromise. Floats are only used in *fft_sub.c*. Provided that it is migrated last intermediate tests are still allowed. This example will work along those lines.*

Having created a set of true Motorola objects the next step is to create a new project from EDE using the *TASKING C/C++ for DSP563xx v3.5r1 with Motorola tools*. This project MUST be created in:

- **c:\migrate\out\mixed**

The simulator command file in this folder will attempt to load an object called *mixed.cld* which is the reason why the project itself MUST be called *mixed.pjt*. The assembler objects created after executing *rebuild.bat* MUST be added and *mixed.opt* MUST be imported via **Project | Load Options**. After having done this the most paramount project settings are:

Project Option	Selection
C compiler Code Generation Default data memory:	Y memory
Assembler Miscellaneous Define user macros:	FFTSIZE=16
Motorola Assembler Define user macros:	FFTSIZE 16
Motorola Linker Motorola Linker Options additional libraries:	lib563cy.clb

Before trying your first build from this project you MUST add the library-, include- and execute paths to **Project | Directories** as listed below. Note the abbreviated notation for folder program files. This is a requirement for the Motorola GNU based tools that make part of the *TASKING C/C++ for DSP563xx v3.5r1 with Motorola tools*. The *TASKING C/C++ for DSP563xx v3.5r1* toolchain does not have problems with white spaced folders as all tools used in this toolchain are white space compliant.

- **c:\progra~1\altium\c563v3.5r1\bin**
- **c:\progra~1\altium\c563v3.5r1\include;**
c:\migrate\src\motorola
- **c:\progra~1\altium\c563v3.5r1\lib\563xx;**
c:\progra~1\motorola\dsp\dsp\lib

The build should be faultless and has to be followed with a simulation:

- **sim56300 sim.cmd**

As before the resulting *fft.txt* MUST be compared to *fft_ref.txt* before proceeding to the next step which is to selectively replace all assembler object modules with their corresponding source modules. Not all sources will compile or assemble by default. This depends on the differences between the Motorola GNU based tools and the TASKING tools. After each successfully added module you MUST rebuild, simulate and re-compare files *fft.txt* and *fft_ref.txt*. In this example we start with the assembly file *fft_calc.asm* as stored in:

- **c:\migrate\src\motorola**

To have the TASKING assembler optimise *fft_calc.asm* it MUST first be copied to *fft_calc.src* before adding it to the project (note that adding implicitly assumes removing the original Motorola object file of the same name). The resulting output file from the TASKING assembler will be named

fft_calc.asm and will then finally be assembled by the Motorola GNU assembler. Rebuilding the project will give a warning 'XDEF interpreted as GLOBAL' which may be ignored. Conclude with simulating and comparing the output.

The next module in line is *fft_main.c*. As this module is fully written in ANSI-C the changes are zero. It will compile without error. Simulating *mixed.cld* will result in an *fft.txt* that is not different from *fft_ref.txt*. This leaves us with only *fft_sub.c* and changes to this module are minor because from v2.2r2 the *__asm* intrinsic is also supported by the TASKING 563xx compiler package. In fact it will compile without errors the first time you try it. This suggests no extra edits are required and indeed one could decide to do so. However, we recommend the type of buffers *table* and *data* (see *fft_main.h*) are changed from *int* to *__circ_L long* because *fft_calc.src* tells us they are in fact circular. The Motorola GNU based tools do not feature a type similar to *__circ* and their implementation of *_L* is that of two sequential words in default data space. These two account for the usage of *int* in the original sources. As *__asm* only uses the addresses of both tables there is no harm in that.

For the TASKING 563xx tools the usage of *__circ_L long* will allow the compiler to execute all the required optimizations that can be attributed to these types. After changing the types for *table* and *data* the module will no longer compile. Instead it will report an error for the constraints used for the second parameter of *__asm*. This is only logical since circular pointers and addresses consist of an address and modulo register. The Motorola GNU tools use constraint "A" since all of its pointers are 24 bits only. With the change of storage type the constraint MUST now be changed to "C". The observant reader will remember that before running the final test the C-library MUST be changed from Motorola to TASKING.

As all TASKING C-libraries are for default X space this will take three steps. The first one is to select default X space from **Project Options | C Compiler | Code Generation**. The second one is to clear the Additional libraries edit box from **Project Options | Motorola Linker | Motorola Linker Options** and thirdly the *Link with default TASKING libraries* checkbox MUST be ticked. All this MUST be concluded with an entire rebuild, simulate and compare. You will find two small differences which should be accounted to the difference in floating point implementation between Motorola GNU based tools and TASKING tools explained earlier.

NOTE

At the point of writing this document it should be noted that Motorola's automatic DSIZE calculation fails when not using default Y memory. This means DSIZE must be overloaded by the user program and it is for this reason *__semi_stack__* has been included in the pre-migrated version of *fft_sub.c*. Define *__OVERLOAD_DSIZE__* was automatically set while *mixed.opt* was being imported.

The final step towards migration is to create a true TASKING project. It should be created in:

- `c:\migrate\out\tasking`

Before importing `tasking.opt` the toolchain MUST be switched to `TASKING C/C++ for DSP563xx v3.5r1` thereby enabling a true TASKING project without Motorola tools. At this point you can either add your migrated modules or the pre-migrated ones from:

- `c:\migrate\src\tasking`

Note that the pre-migrated modules use pre-processor controls to allow it to compile and assemble on both the TASKING tools and the Motorola GNU based tools. You can now proceed to build the project and simulate it by either evoking CrossView Pro from the command line:

- `xfw56x tasking.abs -tcfg sim563.cfg -p sim.cmd`

Or by pressing the debug-button from the editor interface. The resulting output, again can be compared with the original `fft_text.txt`.

5. COMPATIBILITY ISSUES

The differences between the TASKING and the Motorola toolchains result in several compatibility issues. Most of these differences are rooted in different base technology within these tools; others have been introduced to improve the code quality of the application. This paragraph describes the differences and methods to resolve them in the migration process. Also note that both the compiler- and assembler manual contain appendixes with supplemental information:

- *Assembler Manual, Appendix F, Migration from Motorola CLAS*
- *Compiler Manual, Appendix B, Motorola Compatibility*

5.1 Compatibility issues on the C source level

From a C point of view there are little differences between the TASKING and Motorola toolchains. The major difference is the better code quality of the TASKING compiler. An important improvement in code density and speed can be obtained by upgrading the algorithm from the integer to the fractional data type. This will remove many shift instructions that are only necessary to express the floating-point calculations in the integer data type. However, backward compatibility with the fixed-point implementation is difficult to retain. Compatibility with the floating-point version of the algorithm is almost automatic because the fractional data type can easily be mapped to a floating-point one.

5.2 Compatibility issues on the C object level

The object files of the TASKING and GNU tools usually cannot be mixed, although the TASKING linker can read the GNU object format. By default there are differences in the user stack pointer, the default memory space, the function calling convention and the floating-point number format. The TASKING C compiler provides compatibility options that enable the generation of GNU compatible objects at the cost of code efficiency. These have been listed below, for

convenience only using the command line switches. Their EDE synonyms can be selected from both **Project Options | C Compiler | Code Generation** and **Project Options | C Compiler | Output**. When using the steering program the user may suffice with using **-S**. It will know this option belongs to the TASKING assembler and as its meaning is to generate Motorola assembly - having used TASKING assembler optimizations - it will launch the compiler to use **-C1** which enables all compatibility switches.

Command line switch	Purpose	Code efficiency effects
-Ca	Motorola assembler compatible output	more efficient ¹
-Cc	Motorola compatible calling convention	less efficient code
-Cg	Generate Motorola COFF debug information	none ²
-Cr	R6 is user stack pointer	none
-Cs	Motorola (old-style TASKING) stack frame	less efficient code
-My	Default Y data space	

1. Provided the TASKING assembler uses **-S** to take full advantage from TASKING optimizer controls generated by the TASKING compiler.
2. This switch controls the type of debug information. Whether it is generated or not still depends on compiler command line option **-g**.

The compiler also implements a function modifier that specifies the GNU calling convention. By specifying the GNU calling convention only where necessary, the implied overhead can be avoided in the rest of the code. In the example below `__motorola` can be resolved from a Motorola library or object and `__tasking` can be linked to a Motorola project that uses it.

```
extern __compatible int __motorola(int,int,int);

int __compatible __tasking(int a,int b,int c)
{
    return a+b+c;
}

void tasking_C(void)
{
    __motorola(1,2,3);
    __tasking(1,2,3);
}
```

The EDE or command line options for the user stack pointer and for the default memory are still required when using this method. Enabling these options will allow generation of assembly sources to be mixed with GNU assembly sources and objects. The floating-point format however remains an unresolved issue i.e. the use of floating point numbers MUST be restricted to one of the two toolchains. C object files may also reference floating-point and run-time library subroutines. As the names and implementations of these routines are

different, they MUST be provided for both toolchains. The TASKING floating-point and run-time libraries have been included in both IEEE and Motorola COFF format.

5.3 Compatibility issues on the assembler source level

The TASKING and Motorola assemblers are to a great extent compatible. The primary differences are in the *opt* directive, the *org* directive and the handling of forward references. Some of the *opt* directives are missing in the TASKING assembler; others are not present in the Motorola assembler. The missing directives represent fairly uncommon options, whereas the additional directives handle new features, like optimization switches.

The *org* directive also has new features, e.g., named sections and section attributes. A different method of creating code overlays exists in the TASKING assembler the details of which can be found in the assembler manual. The TASKING assembler will support forward references in a section, whereas the Motorola assembler only assembles on a strict per-line basis. This gives rise to differences in the handling of expressions involving labels.

Last, but not least, the TASKING assembler contains a new opcode called *gmove* and a new directive called *void* which are both used to improve code optimization. The TASKING assembler also prefers the keyword *extern* to *xref*. By using conditional assembly, it is possible to switch between the syntaxes and retain compatibility with both assemblers. For instance, the TASKING compiler can generate Motorola assembler compatible output which emits the following code at the start of a file:

It is clear that the *opt* directive is emitted in two flavours; the

```
if @def(_AS56)
opt
nops,now109,noop,opjmp,norp,w139,oprep,cache128
else
opt nops,cc,norp
endif

if !@def(_AS56)
define extern 'xref'
define gmove 'move'
define void ';'void'
define SYMB ';'SYMB'
section demo_c
endif
```

TASKING keyword *extern* is mapped to *xref*, the TASKING opcode *gmove* is mapped to *move*, and the directives *void* and *SYMB* are turned into a comment. In addition, a section is created which is not required in the TASKING assembler.

5.4 Compatibility issues on the assembler object level

The TASKING linker can read Motorola COFF object files (with extension *.cln*) as generated by the Motorola assembler. The only restrictions are in the debug information and floating-point symbols. The linker cannot convert debug information from COFF format into TASKING format, therefore omits

them. For proven object code this is not important. The second restriction deals with the TASKING object format, which cannot store floating-point values for symbols, therefore the following construct cannot be stored in the TASKING object format:

```
TWOPI    equ    6.28
global  TWOPI
```

To solve this problem the floating-point symbol can be kept local, imported from an include file or scaled to an integer value. It is also possible to convert a Motorola *.cln* file to the TASKING object format with the linker. Simply feed the *.cln* file to the linker without specifying any libraries to create an *.obj* file. This will remove the debug info as described before.

The Motorola linker cannot read the TASKING object format. To read TASKING output files, the source files must be reprocessed with the Motorola assembler. The TASKING assembler has a command line switch to generate Motorola compatible assembly, with all optimizations in place, instead of the normal *.obj* file. The resulting assembler file (default extension *.asm*) can then be fed to the Motorola assembler, creating a *.cln* file suitable for further processing with the Motorola tools.

5.5 Compatibility issues with libraries

The TASKING linker can access Motorola COFF object libraries (with extension *.clb*) which are generated by the Motorola archiver *dsplib*. The restriction on floating-point symbols, mentioned above, remains valid.

6. CONCLUSION

Migrating a project from the Motorola GNU based tools to the TASKING tools is relatively simple through use of compatibility switches on compiler and assembler level. The file-by-file migration method greatly decreases the failure risk of this process. The added functionality and ease of use of the TASKING tools prove that migration is a desirable step to increase product quality and productivity.

7. REFERENCES

1. Hal Chamberlin, *Musical Applications of Microprocessors*, Prentice-Hall, 1982.
2. *DSP56KCC Users Manual*, Motorola, 1997.

8. GLOSSARY

The term *source file* is used for any handwritten file that translates to object code, whether in C or assembly language. The term *object file* is used for intermediate binary files. The term *executable* is used for the binary file that can be run on the target DSP.

The Motorola C compiler for the DSP56xxx is based on the GNU C compiler framework; therefore the term *GNU tools* is widely used. The Motorola assembler package (assembler, linker and archiver) is needed to process the output of the GNU tools. Motorola has adopted the COFF (Common Object File Format) object file format, but has added some of its own

features. The result is called Motorola COFF or the MCOFF file format. Default filename extensions in this format are .cln (relocatable object file), .cld (executable, absolute object file), and .clb (object library). The Motorola debuggers (text mode and graphical user interface) accept only the .cld file format.

The *TASKING C/C++* toolchain for the *DSP56xxx* uses the IEEE-695 standard for the object file format. Default filename extensions in this format are .obj (relocatable object file), .abs (absolute object file, executable), and .a (object library). The TASKING debugger, called CrossView Pro, only accepts the .abs file format. On the Windows platform, the TASKING tools come with an integrated development environment called EDE (Embedded Development Environment). The EDE has an automatic makefile generation utility that can be switched between all TASKING toolchains.