

Programming with μ C/OS-II Operating System on the Ide68k Integrated Development Environment

by: [Peter J. Fondse](#)

1. Introduction.

This document describes the IDE68K Integrated Development Environment and the μ C/OS-II development flow. IDE68K gives you the ability to build simple μ C/OS-II applications for the 68000 or 68020 processor quickly and running them on the 68000/68020 Visual Simulator or the Command Line simulator.

It is assumed that the reader has a working knowledge of:

- The basics of editing and compiling files with IDE68K.
- Assembly and C language programming.
- Addressing peripheral devices in the 680x0 architecture.

More information on these subjects can be found in the [Getting started with IDE68K](#) document.


It is assumed that IDE68K version 3.0 is installed on the PC (Windows 8, 7, Vista or XP) and that the non-CPU specific files for μ C/OS-II are obtained from Micrium and copied to the C:\Ide68k\uCOSII directory as described in the [Installing uCOS-II on IDE68K](#) document.

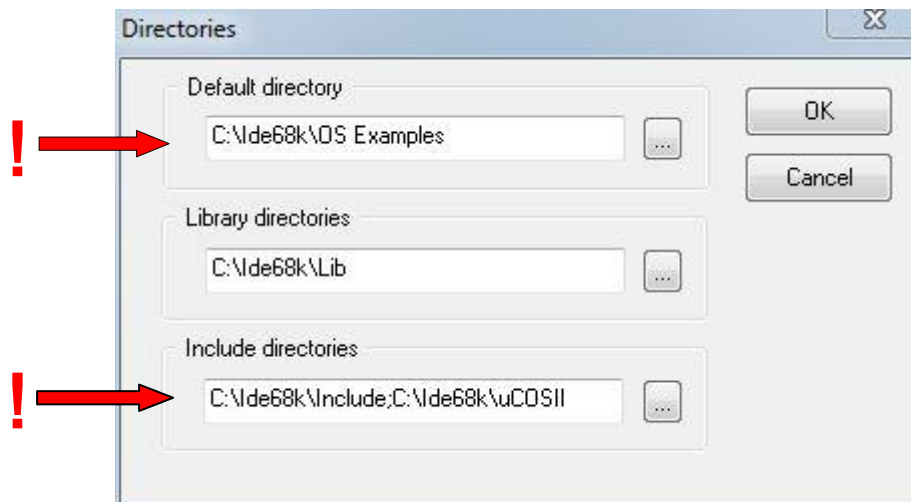
This tutorial provides step-by-step instructions for building and running a simple program based on the 68000 architecture and the μ C/OS-II RTOS.

2. Initialization of the system.

Before IDE68K-functions for editing and compiling can be used, it may be necessary to set certain environment conditions. Most of these are set to default values that apply to standard 68K programs. Some must be changed to run programs under μ C/OS-II. Also, for instance when the program is installed in another directory, the default conditions must be changed to reflect the different environment.

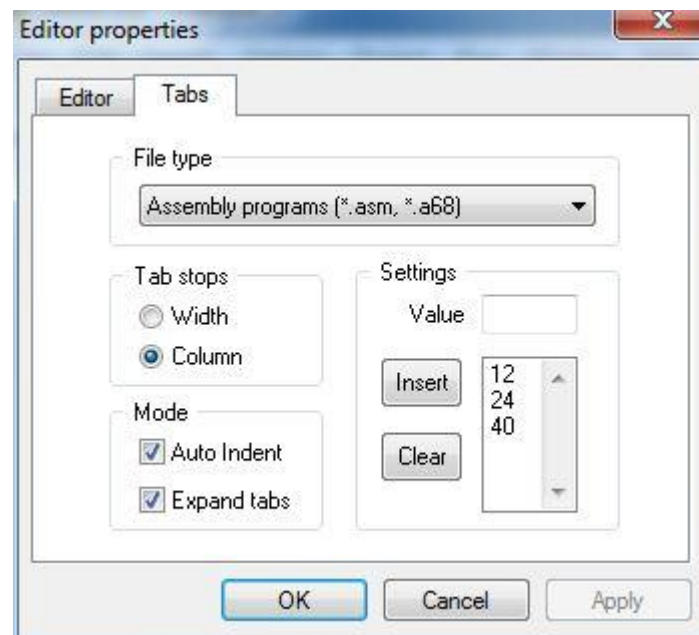
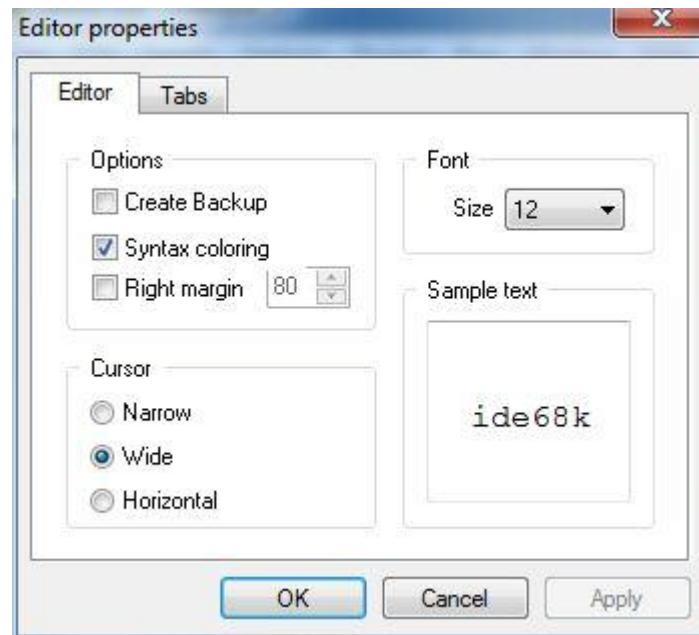
2.1 Directories.

Chose menu [Options->Directories](#) and set the default directory, the library directories and include directories as shown below. Chose your own default directory name that is meaningful to you. By default, this is the directory where the examples for IDE68K are stored. Change this to **C:\Ide68k\OS Examples** if you intend to run the example programs with μ C/OS-II. You only have to do this once, the first time you start IDE68K. The directory you specify is where IDE68K will save your source files and any IDE68K generated files. Library directories are the directories that are searched by the assembler when the INCLUDE directive is found in the source text. Include directories are the directories that are searched by the C compiler when the `#include <...>` preprocessor statement is found in the source text. Change the “Include directories” entry to two directories, **C:\Ide68k\Include** where the standard header files are found and **C:\Ide68k\uCOSII** where μ C/OS-II specific header files are stored, both separated by a semicolon. Use the  button to browse for the desired directories. Directories shown below are the directories used for running μ C/OS-II.




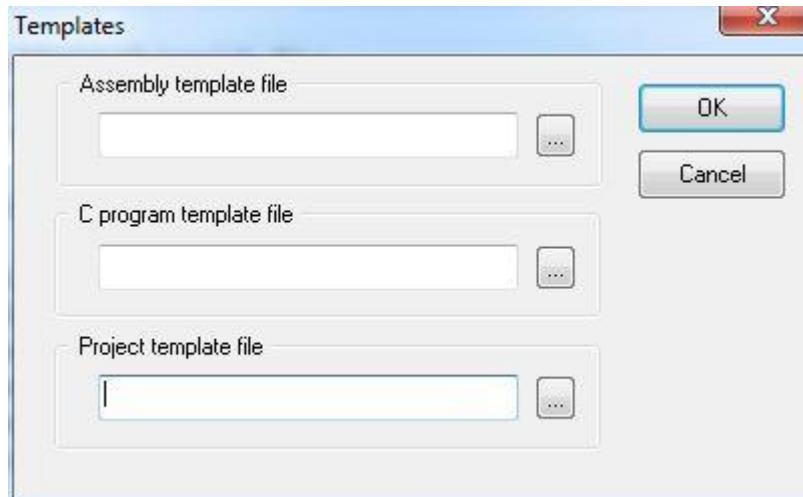
2.2 Editor.

Chose menu [Options->Editor](#) and set the properties for the editor and tabs as shown below. You can change the settings as you like. Note that there are three tab-settings for Assembly programs, C programs and other files. You only have to do this once, the settings are saved and restored when you start IDE68K again. Settings are saved automatically when you quit the program, or chose menu item [Options->Save options](#).



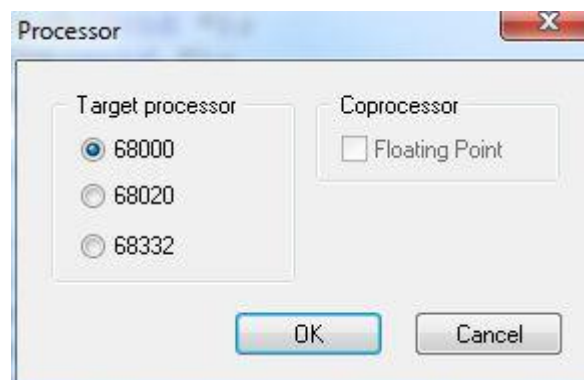
2.3 Templates.

Chose menu [Options->Templates](#) where you can set files that are used as templates. When you start a new file or project, the content of the template file is automatically inserted into the new file or project. No templates are set during the installation of IDE68K. Use the  button to browse for the desired files.




2.4 Processor.

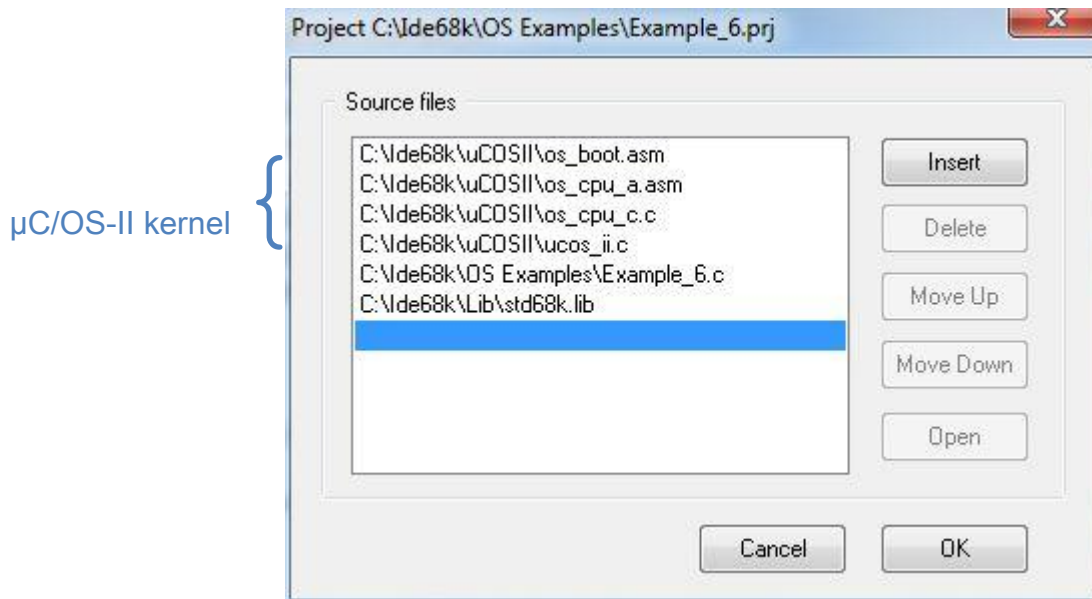
Chose menu [Options->Processor](#) and select the target processor (see below). You only have to do this only once, the selection is saved and restored when you start IDE68K again. Note that you cannot select the floating point coprocessor with the 68000 processor. This confirms with the hardware characteristics of the 68000, this CPU has no coprocessor interface.




3. Loading a project.

Programming for μ C/OS-II starts with loading an existing project or building a new project. A description of how to build a new μ C/OS-II project can be found in paragraph 6. To start with an existing project select menu-item [Project->Open project](#), type the F5 function key or click on the  toolbar button. A standard open file dialog is shown. Select Example_6(.prj) and click on the



“Open” button. The project edit dialog is displayed showing all the files for the project (see below).

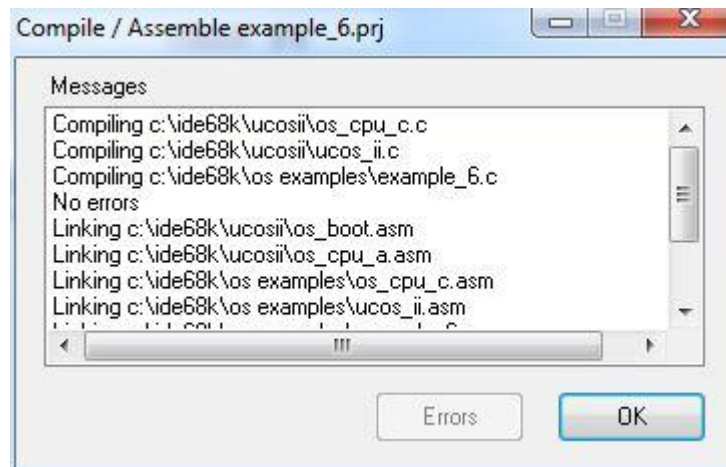


To open a file for editing, select [Example_6.c](#) and click on the “Open” button. Clicking the “OK” button closes the dialog. You can also double-click on [Example_6.c](#). The contents of the file are displayed in an edit window.



Select menu-item [Project->Project edit](#), type the F6 function key or click on the  button to display the project edit dialog again.

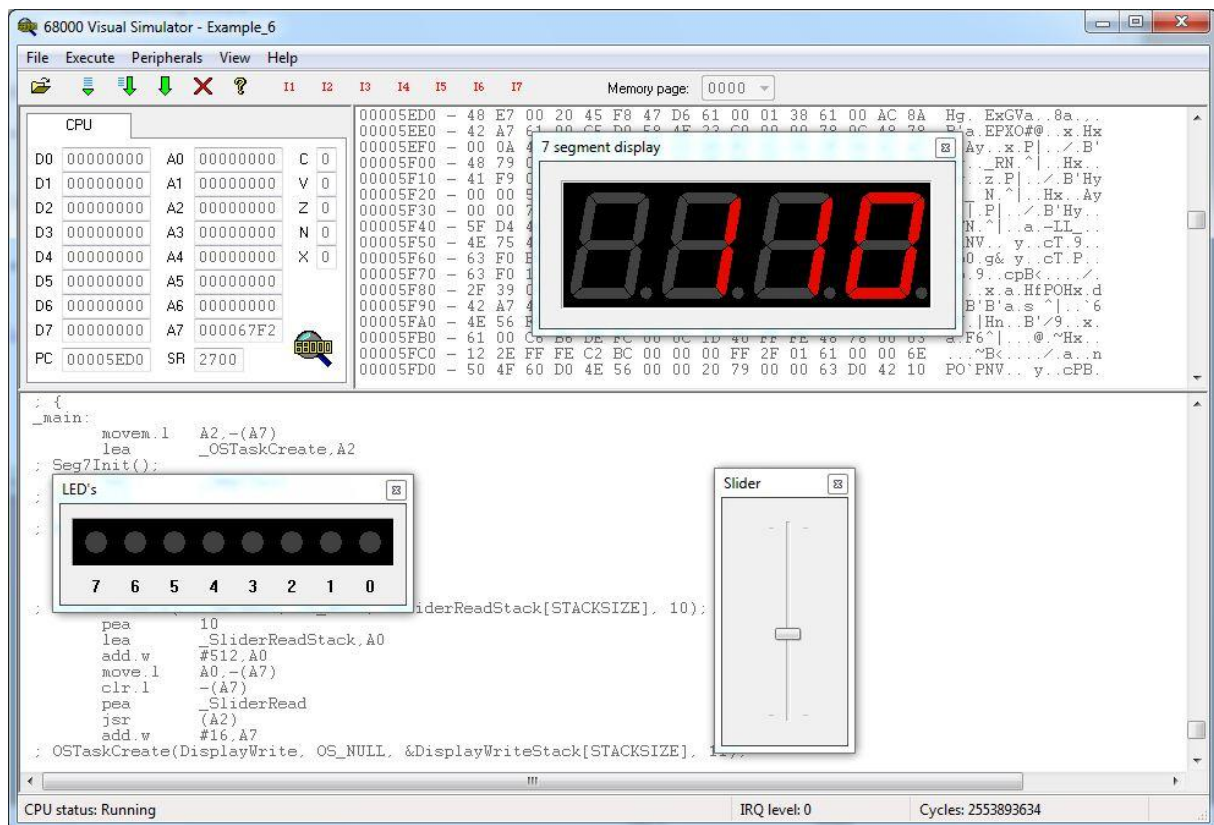
4. Editing and compiling.


You can now edit the file contents. Changes are only made to the edit buffer unless you save the file by selecting menu-item [File->Save](#), typing Ctrl+S or clicking on the  button. Select menu-item [Project->Build all](#), type the F7 key or click on the  button to compile the project. The Compile/Assemble dialog is shown (see below). If warnings or errors are found, the “Warnings” or “Errors” button is enabled. Clicking on these buttons shows a list of warnings and errors found. Double-clicking on a warning or error displays the line where the warning or error is found. Sometimes, when the assembler detects the warning or error, it is not the file itself that is displayed but the combined assembly file for the project. The error however is not here, but in one of the .C or .ASM source files. The combined assembly file cannot be edited, its changes are immediately overwritten by the next compilation.



5. Running the 68000 Visual Simulator.

Once the program has been compiled without errors or warnings, you can run the Visual Simulator. Select menu-item **Run->Visual simulator**, type the F9 function key or click on the  toolbar button. The visual simulator window opens. The visual simulator is a stand-alone program started by the IDE. It has its own menu, toolbar and Help system. To run Example 6, select menu-items **Peripherals->LEDS**, **Peripherals->Slider** and **Peripherals->7 segment display**. Select menu-item **Execute->Run**, type Ctrl+F5 function key or click on the  toolbar button.



Watch the program run, LED 0 will flash once every 2 seconds to indicate that the program is running. Place the mouse cursor on the slider button and move it up and down. The 7-segment display will indicate the slider position, 0 – 255. Menu-item [Execute->Reset](#), typing the Ctrl+Break keys or clicking on the  toolbar button resets the simulator.

6. Creating a new project.

To start with a new project, it may be useful to create a new directory where the files are stored. Use the [Options->Directories](#) menu-item to set this directory as the default directory in the IDE68K environment. Create and edit the .C and .ASM files for the new project. They are saved in the previously created default directory.

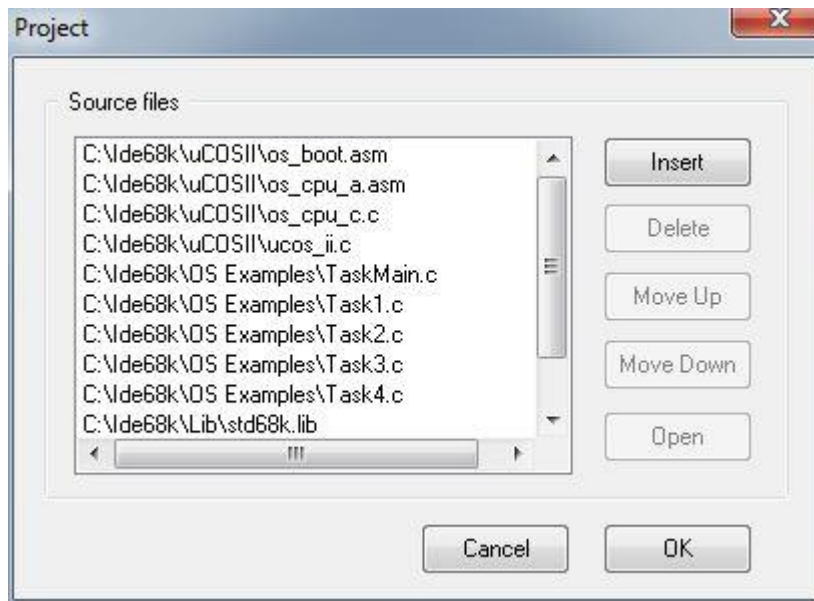
Select menu-item [Project->New project](#) to show the project edit dialog. Click on the “Insert” button to open a standard File Open dialog. Navigate to C:\Ide68k\uCOSII directory, select [os_boot.asm](#) and click on the “Open” button, os_boot.asm is the file which contains the IDE68K specific interrupt vector definitions and a few BIOS functions for the simulator. This file should always be first in the project list. Click on the “Insert” button to insert [os_cpu_a.asm](#) and [os_cpu_c.c](#). These are the files with 68000 specific routines. Insert [ucos_ii.c](#). This is the file with the architecture-independent C code for μ C/OS-II. These four files together comprise the μ C/OS-II kernel. All these files can be found in directory [C:\Ide68k\uCOSII](#).

At this moment it may be helpful to save the project, e.g. under the name os_kernel.prj, and enter this name in the “Project Template file” of the [Options->Templates](#) dialog. Click on the “OK” button to save the project. Immediately choose menu-item [Project->New project](#) to open a new project. From now on, any time you open a new project, these four files are already automatically inserted in the project.

Now insert the application file(s) in the project. This can be one file with the source code for all tasks but, when tasks become larger, the program code can be split in multiple files, one per task.

Finally insert the library file. This file can be omitted when compiling for the 68020 and no library functions like `sprintf()` or `strlen()` are used. The 68000 always needs the library for functions to multiply or divide 32-bit numbers.

Following figure shows a typical project list for μ C/OS-II.



7. Examples.

A number of example projects are in directory C:\Ide68k\OS Examples. These programs have two purposes: (1) To test the RTOS running on IDE68K and (2) to get insight on how to program for multi-tasking.

Example 1.

This is a minimal program to verify multitasking. Two tasks are created, Task #1 prints "This is task 1", task #2 prints "This is task 2". However, simple and small as it is, there is a serious flaw in the program. The device to print on is a shared resource! The error can be observed as sometimes printing of task #2 is interrupted and the higher priority task #1 prints "This is task 1" in the middle of "This is task 2". A mutex or semaphore is required to synchronize both tasks.

Example 2.

This program checks the accuracy of the timer interrupt tick. The program prints the real-time clock time, then waits 1000 ticks and prints again. Printing should be at exactly 10 seconds intervals. The timer tick, at 10 ms intervals (100 interrupts per second) is generated by an internal timer in the Windows operating system.

Example 3.

This program is basically the same as example 2 but uses a timer.

Example 4.

This program uses a 1 second timer to cycle led's 0 and 1. Select the LED display in the Visual Simulator.

Example 5.

This program is basically identical to "switches.asm" or "cswitches.c" but uses tasks that communicate through a mailbox. Task #1 checks the switches at 1 second intervals. When the state of one or more switches has changed, a message is posted to task #2. This task displays the message, that is the state of the switches, on the LED display. Select Switches and LED display in the Visual Simulator.

Example 6.

This program has 3 tasks, task #1 checks the position of the slider control at 100 milliseconds interval. When the state of slider has changed, a message is posted to task #2. This task displays the message, that is the state of the slider (0 – 255) on the 7 segment display. A third task (the "watchdog") flashes led nr. 0 of the LED array at 2 seconds interval to verify that the program is running. Select LED's, Slider and 7-segment display in the Visual Simulator.

Example 7.

This program has 3 tasks, task #1, the LED-display task, sends a query to task #2, the slider-read task, pending on the query mailbox. After reception of the query message, task #2 reads the slider position and posts it back to task #1. Task #1 shifts the active LED one position to the right and then waits for the number of timer ticks derived from the slider position received from task #2. A flag is set every eight shifts to signal this event to task #3. Task #3 waits on this flag and when set, generates a short tone on the PC speaker. Select LED's and Slider in the Visual Simulator.

Example 8.

This is a small program to verify interrupt processing with μ C/OS-II and IDE68K. The program has 3 tasks, a high-priority interrupt task and two normal tasks, task #1, the LED-display task and task #2 the bar-display task. Tasks #1 and #2 are simply to verify that the program is running.

Initially the interrupt task is suspended until resumed by the interrupt service routine for autovector-interrupt level 4. The interrupt service routine must be written in assembly because it requires an operation that cannot be expressed in C. The file, called Example_8.a68, must be added to the project list. (See the project list for Example_8.prj).

When an autovector-interrupt level 4 is received (click on the small button marked I4 at the top of the Visual Simulator window) a .wav file ("bigben.wav") is played.

Select LED- and Bar-display in the Visual Simulator.

Example 9.

This program is basically Example #1 in the book "*MicroC/OS-II, The Real-Time Kernel*" adapted to run on IDE68K instead of an IBM PC with MS-DOS. The program creates 7 identical tasks that print their arguments, passed at creation time, at a random position on the display. The color corresponds to the argument value, 1 is blue (RGB = '001'), 2 is green (RGB = '010'), 3 is cyan (RGB = '011') and so on. The display is not the memory mapped display of the IBM PC but the much slower "drawpad" device of the simulator, in reality a bitmap in Windows. Characters are printed with a size of 20 x 10 (h x w) pixels. With a drawpad size set to 800 x 500 pixels, this corresponds to a text display of 25 lines, 80 characters per line.

Another file, called display.c must be added to the project list. (See the project list for Example_9.prj) This file can be regarded as the display driver for the "drawpad" device. It isolates the drawpad specific I/O operations from the task program.

Example 10.

This program is basically Example #4 in the book "*MicroC/OS-II, The Real Time Kernel*" adapted to run on IDE68K instead of on an IBM PC with MS-DOS. The program creates 10 identical tasks that print the angle in degrees, the sine and the

cosine of that angle. Every second, the angle for every task is incremented by 10 degrees. This program obviously needs floating point arithmetic. Chose menu-item [Options->Processor](#) and select 68020 processor with floating point coprocessor. In order to support floating point operations, μ C/OS-II must be extended to save and restore the floating point registers during context switches. The functions to do this are in [os_fcpu_a.asm](#) and [os_fcpu_c.c](#), these files must be inserted in the project list instead of [os_cpu_a.asm](#) and [os_cpu_c.c](#). (See the project list for Example_10.prj). Also the floating point library, std68kfp.lib replaces the integer library std68k.lib in the project list. The display is the same “drawpad” device of 800 x 500 pixels as in the previous example.