ERIKA Enterprise Manual for the Microchip PIC24, dsPIC30 (R) DSC and dsPIC33 (R) DSC targets

the RTOS for PIC devices

version: 1.1.9
September 22, 2011

About Evidence S.r.l.

Evidence is a spin-off company of the ReTiS Lab of the Scuola Superiore S. Anna, Pisa, Italy. We are experts in the domain of embedded and real-time systems with a deep knowledge of the design and specification of embedded SW. We keep providing significant advances in the state of the art of real-time analysis and multiprocessor scheduling. Our methodologies and tools aim at bringing innovative solutions for next-generation embedded systems architectures and designs, such as multiprocessor-on-a-chip, reconfigurable hardware, dynamic scheduling and much more!

Contact Info

Address:
Evidence Srl,
Via Carducci 56
Località Ghezzano
56010 S.Giuliano Terme
Pisa - Italy
Tel: +39 050 991 1122, +39 050 991 1224
Fax: +39 050 991 0812, +39 050 991 0855

For more information on Evidence Products, please send an e-mail to the following address: info@evidence.eu.com. Other informations about the Evidence product line can be found at the Evidence web site at: http://www.evidence.eu.com.

Contents

1 Introduction

Embedded microcontroller units are spreading in thousands of applications, ranging from single to distributed systems, control applications, multimedia, communication, medical applications and many others. Modern microcontrollers, which are growing in computational power, speed and interfacing capabilities, are more and more feeling the need of tools to make the development of complex scalable applications easier.

The dsPIC (R) DSC family represents one of the latest products of Microchip Technology Inc., the world leading company in the field of microcontroller units. With a speed of up to 40 MHz, the dsPIC (R) DSC family seamlessly integrates a DSP core for high performance computation with a full range of interfaces to several buses like CAN, I2C, SPI, serial lines, and so on.

1.1 Erika Enterprise **and** RT-Druid **for** dsPIC (R) DSC

Embedded applications often require tight control on the temporal behavior of each single activity in the system. The research in the field of real-time systems brought the team of Evidence Srl to design a small, efficient, modular real-time kernel that can be used to easily guarantee real-time constraints in every embedded applications.

Erika Enterprise and RT-Druid represent the answer of Evidence Srl for the development of scalable real-time applications for the dsPIC (R) DSC family.

Erika Enterprise provides dsPIC (R) DSC developers the following features:

Traditional RTOS features

- Support for four conformance classes to match different application requirements;
- Support for preemptive and non-preemptive multitasking;
- Support for fixed priority scheduling;
- Support for stack sharing techniques, and one-shot task model to reduce the overall stack usage;
- Support for shared resources;
- Support for periodic activations using Alarms;
- Support for centralized Error Handling;
- Support for hook functions before and after each context switch.

RT-Druid **development environment**

- Development environment based on the Eclipse IDE;
- Support for the OIL language for the specification of the RTOS configuration;
- Graphical configuration plugin to easily generate the OIL configuration file and to easily configure the RTOS parameters;
- Full integration with the Cygwin development environment to provide a Unix-style scripting environment;
- Apache ANT scripting support for code generation;

dsPIC (R) DSC **integration features**

- Installation setup which integrates Microchip software together with fully configured Evidence Erika Enterprise + RT-Druid;
- Full support for the Microchip devices libraries;
- Full support for the Microchip C30 compiler;
- Full support of the MPLAB IDE debugging environment;
- Full support for the Microchip ICD2 debugger;
- Full support for dsPIC (R) DSC series 30 and 33, and for PIC24;
- Support for the 802.15.4 (ZigBee) wireless communication protocol (coming soon);
- Support for I/O to Multimedia Card (MMC) / Secure Digital with FAT filesystem (coming soon);
- Development of many specific hardware drivers for dsPIC (R) DSC, like multiple servomotor driving, bus EIB support (domotic), and many other (coming soon);
- Support for the FLEX development board and for some other dsPIC (R) DSC evaluation boards;

1.2 Integration with Microchip Inc. products

Erika Enterprise and RT-Druid aims to the best integration with the existing tools for development available from Microchip Inc.

RT-Druid will be used to quickly configure the application, setting temporal parameters of real-time tasks, memory requirements, stack allocation and many other parameters. RT-Druid generates the application template, and leaves the developer the task to implement the logic of each single task.

While programming the application, the developer can exploit the power and flexibility offered by the primitives of the Erika Enterprise real-time kernel.

The application can be imported into MPLAB IDE to be written into the dsPIC (R) DSC EPROM flash memory. Moreover, the application can be debugged from within the MPLAB IDE.

1.3 Content of this document

The purpose of this document is to describe all the information needed to create, develop and modify an Erika Enterprise application under the Microchip dsPIC (R) DSC family of microcontrollers. In particular, the document describes:

- The design flow which should be used to generate an Erika Enterprise application;
- The configuration of the development environment;
- The options which are available to configure the system.

Note: If you are looking for a step-by-step / quick guide tutorial on how to use Erika Enterprise and RT-Druid with dsPIC (R) DSC, please read the "Erika Enterprise Tutorial for the microchip dsPIC (R) DSC Platform", available for download on the Evidence Web site.

2 Erika Enterprise for PIC devices

2.1 The RT-Druid and Erika Enterprise design flow

The typical design flow of a Microchip application based on Microchip tools is done inside the MPLAB IDE, a development environment for Microsoft Windows which integrates a source code editor, an instruction set simulator and a debugger.

In addition to the traditional development flow based on MPLAB IDE, Evidence Srl provides a design and configuration environment named RT-Druid, based on Eclipse [1]. Eclipse is an open framework initially developed by IBM, which allows the possibility of integrating various development tools in a common environment.

For that reason, when developing an application for Erika Enterprise, the user is supposed to write the source code inside the RT-Druid IDE (see Figure 2.1).

Application compilation is also done inside the Eclipse Framework. In fact, the RT-Druid code generator is able to generate the Erika Enterprise configuration files together with a set of configuration files (typically, a `makefile` plus a set of `.c` files) which are then used to compile the source code.

After that, compilation is started automatically by pressing on the "Build Project" menu item inside the "Project" menu, which automatically calls the underlying `make` application provided by the Cygwin environment. As an alternative, the "Build Project" command is also available by right clicking on the project name.

The choice of the Cygwin environment has been done to simplify the building process of an application: in fact, Cygwin provides a set of traditional Unix tools like `make`, `awk`, `sed`, which are really useful to implement a command line application building framework. Moreover, these tools are typically available for free on Linux platform, easing in this way the porting of the application to a free development environment such as Linux.

2.1.1 Building an application from command line

The RT-Druid plugins provide three ways to develop an application:

1. A graphical interface to simplify the development of an application, based on Eclipse.

2. A scripting interface based on Apache ANT [2], which is the default scripting environment used in the Eclipse Framework.

3. A standalone code generator, that does not use Eclipse.

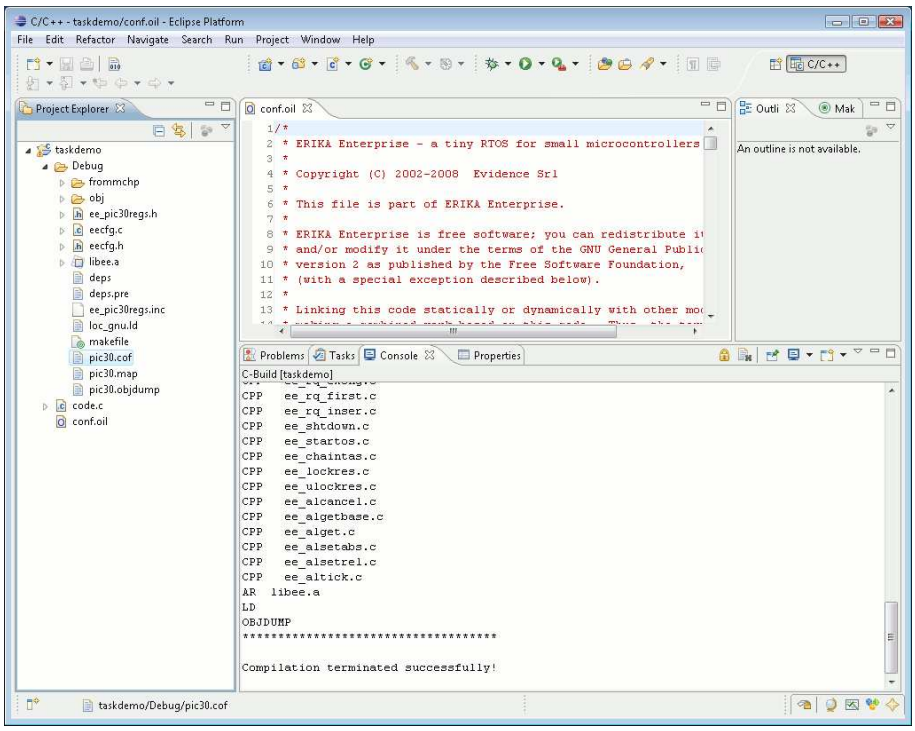

Figure 2.1: The Eclipse workspace and the RT-Druid plugins for dsPIC (R) DSC.

Using ANT or the standalone version, the developer can automatically generate from scripts the configuration data and the makefiles which are then used to compile the application. This removes the need of opening the graphical environment to compile an application, providing a way to implement automatic compilation scripts and regression tests.

Please refer to the RT-Druid reference manual for information about ANT scripting.

2.2 Setting up the compiling environment for dsPIC (R) DSC

Erika Enterprise has been designed to be compiled using the GNU gcc toolchain. The dsPIC (R) DSC porting of Erika Enterprise in particular can be compiled using the GNU tools for dsPIC (R) DSC provided by Microchip. The porting provides both the `binutils` package and the `gcc` package, plus a set of proprietary libraries from Microchip which can be used to control the various peripherals provided by the dsPIC (R) DSC microcontrollers.

The following list describes the various packages which contains the various parts of the compilation toolchain:

The GNU assembler and binutils. This package is distributed inside the MPLAB IDE

from Microchip.

The GNU GCC from Microchip. The compiler is packaged in a separate product, called *The Microchip C30 Compiler*, which is available as a product under the Microchip website. A free version is also available for students and universities. The source code of the compiler is also available under the GPL license on the Microchip web site.

The GNU GCC recompiled from Microchip sources. In addition to the Microchip C30 Compiler, Evidence also offer a free version of the GCC toolchain for dsPIC (R) DSC compiled from the sources made available under the GPL license on the Microchip web site. This version can be used to compile Erika Enterprise without additional packages other than MPLAB IDE.

> **Warning:** Please note that the free version of the gcc compiler recompiled from the Microchip sources is not a *full* replacement for the Microchip C30 compiler. The Microchip C30 compiler includes many features like include files and libraries to control the dsPIC (R) DSC peripherals which are not available as open-source.

C Libraries. A set of libraries which can be used to control the peripherals implemented on the particular Microchip chip in use. These libraries are packaged together with the Microchip C30 Compiler.

To compile an Erika Enterprise application, the development environment needs to be configured to correctly recognize the Microchip C30 compiler and the MPLAB ASM30 assembler programs. For doing so, please go to the "Preference" menu, as shown in Figure 2.2, and find the "RT-Druid/Oil/PIC30 Configurator" form as depicted in Figure 2.3. The first textbox, labeled `Gcc path`, refers to the installation directory of the Microchip C30 compiler. The second textbox, labeled `Asm path`, refers to the installation directory of the ASM30 assembler provided with the MPLAB IDE.

Moreover, Figure 2.3 contains the following checkboxes:

Use EE gcc to resolve dependencies When checked, the C30 compiler recompiled by Evidence from the Microchip sources will be used instead of the installed C30 compiler to compute the dependencies of the .C and .S files, and to perform the C preprocessing of the .S files. This feature is useful to avoid compilation problems when the system has a Student Edition of the C30 compiler with an expired license. In that case, the compiler puts a message on the standrd output, corrupting the dependencies and preprocessing outputs.

Use EE gcc to compile When checked, the C30 compiler recompiled by Evidence from the Microchip sources will be used instead of the C30 compiler to compile the .C files. Please note that the original Microchip Libraries will be linked also in this case.

Warning: The install directories specified in the two textboxes of Figure 2.3 does *not* include the `bin` directory!
That is, `c:\Programmi\Microchip\MPLAB C30` is correct, wheras `c:\Programmi\Microchip\MPLAB C30\bin` is not.

Warning: The install directory of the assembler refers to the assembler provided with MPLAB IDE and *not* the assembler provided with the C30 compiler. The reason is that the directory is used to call the assembler and *also* to copy the `crt0.s` file, which has a different position in the two assemblers distributions made by Microchip.

Warning: If you are using a Student Editon of the Microchip C30 compiler which has an **expired license**, please check the "Use EE gcc to resolve dependencies" checkbox in Figure 2.3.

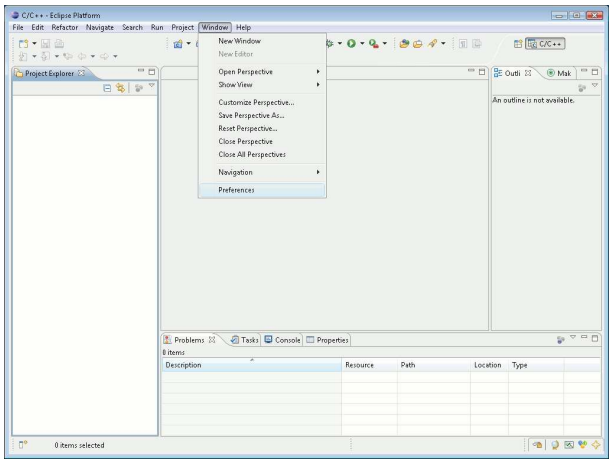

Figure 2.2: Go to the "Preference" menu.

2.3 Writing software for dsPIC (R) DSC using Erika Enterprise

Note: Writing an application for dsPIC (R) DSC using Erika Enterprise is very simple. Please refer to the Erika EnterpriseTutorial for the dsPIC (R) DSC architecture for a step-by-step guide with screenshots on how to create, compile and debug a dsPIC (R) DSC application written with Erika Enterprise.

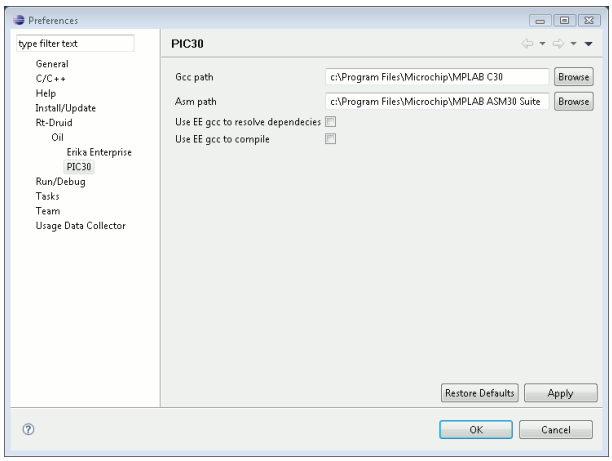

Figure 2.3: Select paths for compiler and assembler.

This section describes the details about the various configuration options which are available to create and compile an Erika Enterprise application for a dsPIC (R) DSC microcontroller.

> **Note:** For a complete description of all the OIL parameters, please refer to the RT-Druid reference manual.

2.3.1 Avoid the generation of dependency files

The typical compilation process of an Erika Enterprise application involves the computation of a dependency file which is used to understand which are the files which needs to be compiled or updated.

To avoid the computation of these dependencies (useful when you are sure you basically have to compile everything), you can put the following line in the OIL file:

```
CPU mySystem {
  OS myOs {
    EE_OPT = "NODEPS";
    ...
  };
  ...
};
```

2.3.2 Avoid the generation of .src files from C files

The typical compilation process of an Erika Enterprise application produces various files which can be used to better analyze the code generated by the C30 compiler. In partic-

ular, from each .C file, a .SRC file is produced containing the corresponding assembler listing, which is then compiled by the MPLAB ASM30 compiler to produce the .o.

It is possible to avoid the intermediate step which leads to the production of the .SRC file. In that case, the compiler will be responsible of producing the .O file directly from the .C file. This in general also speeds up the compilation process a little bit.

To obtain that feature, you can put the following lines in the OIL file.

```
CPU mySystem {
  OS myOs {
    EE_OPT = "NOSRC";
    ...
  };
  ...
};
```

2.3.3 Printing the commands executed (verbose mode)

The default compilation process typically prints only a compact output for each compilation step. That is in general not useful whenever a file is not compiled properly and the user wants to know the exact command which is executed in the compilation process.

To obtain a printing of the complete list of commands issued by the Erika Enterprise makefile, you can add the following line to the OIL file:

```
CPU mySystem {
  OS myOs {
    EE_OPT = "VERBOSE";
    ...
  };
  ...
};
```

2.3.4 Source files composing an application

The source files which can be put in an RT-Druid project are composed by C-language files (with extension `.c`) and Assembler files (with extension `.S`). Assembler files are always preprocessed by the C preprocessor. All the application files which has to be included in the final application needs to be listed inside the OIL file, as in the following OIL example:

```
  ...
  CPU_DATA = PIC30 {
    ...
    APP_SRC = "file_1.c";
    APP_SRC = "file_2.c";
  };
  ...
```

2.3.5 Stack handling

Erika Enterprise can be configured as monostack or multistack.

In a monostack configuration, only a single stack exists in the system. No blocking primitives are supported, and all the tasks and interrupts execute on the same stack. In this case, the one and only stack starts from the top of the application allocated memory, growing towards higher addresses. The monostack configuration can *not* be used if the application needs to call RTOS primitives such as `WaitSem` and `WaitEvent`. Moreover, it cannot be used when Erika Enterprise conformance classes ECC1 and ECC2 are used.

To configure a monostack kernel in the OIL file, the user has to write the following lines:

```
...
CPU_DATA = PIC30 {
  ...
  MULTI_STACK = FALSE;
};
...
```

In a multistack configuration, the kernel support the existence of different stacks in the same application. Having different stacks allow the application tasks to use blocking primitives like `WaitSem` and `WaitEvent`, which basically may block the execution of the running task. In that case, the calling task must have a *private* stack which is changed upon blocking. The stack will be selected again when the task will be rescheduled. There are different stacks available in a multistack configuration:

- A shared stack (used by all the tasks which have a shared stack);
- An IRQ stack (used by all the ISR Type 2 routines);
- A set of private stacks (one for each task which has selected a private stack).

In the dsPIC (R) DSC architecture, the shared stack works as in the monostack configuration, that is it is allocated at the end of the application data section, growing towards higher addresses. The IRQ stack and the private stacks, instead, are allocated in the application data space as arrays.

The following example shows an OIL configuration which configures a multistack kernel without a separate IRQ stack (in this case, IRQ handlers execute on the stack of the interrupted task):

```
...
CPU_DATA = PIC30 {
  ...
  MULTI_STACK = TRUE {
    IRQ_STACK = FALSE;
  };
};
...
```

The following example shows an OIL configuration which configures a multistack kernel with a separate IRQ stack (in this case, some registers are saved on the stack of the interrupted task, but the IRQ handler C function is executed on a separate IRQ stack).

```
...
CPU_DATA = PIC30 {
  ...
  MULTI_STACK = TRUE {
    IRQ_STACK = TRUE {
      SYS_SIZE=64;
    };
  };
};
...
```

2.3.6 Runtime stack checking exceptions

When multistack configurations are used, it is very useful to be informed when a particular stack becomes full. For this reason, the dsPIC (R) DSC core allows the user to specify a stack limit over which an exception should be raised. The stack limit is contained inside the internal register `SPLIM`.

Erika Enterprise is able to automatically handle the `SPLIM` register, setting it at runtime to the top of the current stack when a multistack configuration.

The `SPLIM` feature is by default disabled, because it adds some (little) overhead at each stack change. To enable it, you must include the following lines inside the OIL configuration file:

```
...
CPU_DATA = PIC30 {
  ...
  ENABLE_SPLIM = TRUE;
};
...
```

Please note that Erika Enterprise does not provide a default handler for the stack overflow exception generated by the `SPLIM` register. The exception should be specified by the developer as the behavior to implement is often application dependent.

2.3.7 Interrupt handling

Erika Enterprise for dsPIC (R) DSC provide support for fast Interrupt Service Routines (ISR) which do not require any RTOS primitive to be called, as well as regular ISRs, which can call RTOS primitives (e.g., a timer interrupt can call `ActivateTask` to activate a periodic task). The first kind of ISRs are called *ISR Type 1*, and always have hardware interrupt priority greater than the second kind of ISRs which are called *ISR Type 2*.

At the implementation level, Erika Enterprise uses the dsPIC (R) DSC `DISI` assembler instruction to implement interrupt disabling. The `DISI` instruction only disables the first 6 priority levels out of the 7 available in the 16-bit dsPIC (R) DSC core.

For this reason, ISR Type 2 must always have an interrupt priority between 1 and 6. ISR Type 1 must always have a priority greater or equal than ISR Type 2. As a matter of fact, interrupt priority 7 is reserved for ISR Type 1 only.

ISR names follow the Microchip convention. Basically the ISR names are listed inside the linker script for the particular target which are provided together with the ASM30 assembler packaged together with Microchip MPLAB IDE under the directory `<MPLAB_install_directory>/MPLAB ASM30 Suite/Support/gld`.

To define an ISR Type 1 the developer has to write an interrupt handler as it is written in typical dsPIC (R) DSC applications which does not use Erika Enterprise. Here is an example of the definition of an ISR Type 1 for the timer 3 Interrupt of a `pic30F2010` device:

```
void __attribute__((__interrupt__)) _T3Interrupt(void)
{
  ...
}
```

Writing an ISR 1 in this way implies that:

- The C function will be attached to the interrupt of the peripheral (in the example, timer 3). Every time an interrupt for the peripheral arrives, then the C function will be executed.
- The compiler will generate a proper function prologue and epilogue which saves the register used before starting executing the statements inside the function. The registers will be saved inside the stack of the interrupted task. For that reason, when using a multistack configuration, the user should reserve a proper space able to contain all the nested ISR Type 1 *for each* stack in the system.

To define an ISR Type 2 the developer has to write a C function in the following way:

```
#include "cpu/pic30/inc/ee_irqstub.h"
...
ISR2(_T3Interrupt)
{
  ...
}
```

Writing an ISR 2 in this way implies that:

- An assembler stub will generated for the ISR. The ISR stub will have the name of the ISR (in the example, `_T3Interrupt`). The assembler stub will call a C function named `ISR2_functionname` which content is specified as the content of the function (in the example, the function is called `ISR2__T3Interrupt`). The assembler function will be attached to the interrupt of the peripheral (in the example, timer 3). Every time an interrupt for the peripheral arrives, the assembler stub will execute,

which in turns calls the internal C function whose body has been specified by the developer.

- The assembler stub saves *all* the CPU registers on the current stack. After that, if a multistack configuration with private IRQ stack has been selected, the stack is changed to a private IRQ stack. Otherwise, the ISR will execute on the stack of the running task, as in the ISR1 case. At the end of the stub, the Erika Enterprise end IRQ function will be executed to choose which is the next task to run.

2.3.8 Configuring the usage of Microchip ICD2

dsPIC (R) DSC devices can be debugged using the Microchip product called Microchip ICD2, which is basically an In-Circuit debugger which directly connects to the microcontroller core. When connected, the ICD2 requires the usage of a set of memory locations, which must be left free by the application.

For this reason, when compiling an application which will be debugged using the Microchip ICD2, the user has to specify the following line inside the OIL file:

```
...
CPU_DATA = PIC30 {
  ...
  ICD2 = TRUE;
};
...
```

2.3.9 Configuring a particular dsPIC (R) DSC microcontroller

Microchip produces various versions of the Microchip microcontrollers, each one with different peripherals and memory sizes. To support the heterogeneity of these devices, Microchip offers, through the C30 Compiler toolchain, a set of files which can be used to configure the compiling process.

In particular, for each device, there are four files:

- A linker script, available under the directory `Support/gld` of the Microchip ASM30 Assembler, which contains the linking information such as the memory sizes, and the available interrupt handlers;
- An Assembler include file, available under the directory `Support/inc` of the Microchip ASM30 Assembler, which contains the declaration of the device's register addresses to be used inside assembler programs;
- A C include file, available under the directory `support/h` of the Microchip C30 Compiler, which contains the declaration of the device's addresses to be used inside C programs;
- A library, available under the directory `lib` of the Microchip C30 compiler, which contains a set of libraries for the usage of the microcontroller peripherals.

Every Erika Enterprise application which has to be compiled together with the Microchip C30 compiler needs the specification of these four files. To set which files have to be used for the particular device, the user can specify the following lines inside the OIL file.

If the device number is known, and the files to be used are the default files provided by Microchip, then the developer can directly specify the device name in the OIL file, as in the following example:

```
...
MCU_DATA = PIC30 {
  MODEL = PIC33FJ256GP710;
};
...
```

Currently, Erika Enterprise supports the following values for the `MODEL` attribute:

- PIC24 devices:
  `PIC24FJ128GA006`, `PIC24FJ128GA008`,
  `PIC24FJ128GA010`, `PIC24FJ32GA002`,
  `PIC24FJ32GA004`, `PIC24FJ64GA002`,
  `PIC24FJ64GA004`, `PIC24FJ64GA006`,
  `PIC24FJ64GA008`, `PIC24FJ64GA010`,
  `PIC24FJ96GA006`, `PIC24FJ96GA008`,
  `PIC24FJ96GA010`, `PIC24HJ128GP206`,
  `PIC24HJ128GP210`, `PIC24HJ128GP306`,
  `PIC24HJ128GP310`, `PIC24HJ128GP506`,
  `PIC24HJ128GP510`, `PIC24HJ256GP206`,
  `PIC24HJ256GP210`, `PIC24HJ256GP610`,
  `PIC24HJ64GP206`, `PIC24HJ64GP210`,
  `PIC24HJ64GP506`, `PIC24HJ64GP510`.

- PIC30 devices:
  `PIC30F1010`, `PIC30F2010`,
  `PIC30F2011`, `PIC30F2012`,
  `PIC30F2020`, `PIC30F2021`,
  `PIC30F2022`, `PIC30F2023`,
  `PIC30F3010`, `PIC30F3011`,
  `PIC30F3012`, `PIC30F3013`,
  `PIC30F3014`, `PIC30F4011`,
  `PIC30F4012`, `PIC30F4013`,
  `PIC30F5011`, `PIC30F5013`,
  `PIC30F5015`, `PIC30F5016`,
  `PIC30F6010`, `PIC30F6010A`,
  `PIC30F6011`, `PIC30F6011A`,
  `PIC30F6012`, `PIC30F6012A`,
  `PIC30F6013`, `PIC30F6013A`,

PIC30F6014, PIC30F6014A,
PIC30F6015.

- PIC33 devices:
  PIC33FJ128GP206, PIC33FJ128GP306,
  PIC33FJ128GP310, PIC33FJ128GP706,
  PIC33FJ128GP708, PIC33FJ128GP710,
  PIC33FJ128MC506, PIC33FJ128MC510,
  PIC33FJ128MC706, PIC33FJ128MC708,
  PIC33FJ128MC710, PIC33FJ256GP506,
  PIC33FJ256GP510, PIC33FJ256GP710,
  PIC33FJ256MC510, PIC33FJ256MC710,
  PIC33FJ64GP206, PIC33FJ64GP306,
  PIC33FJ64GP310, PIC33FJ64GP706,
  PIC33FJ64GP708, PIC33FJ64GP710,
  PIC33FJ64MC506, PIC33FJ64MC508,
  PIC33FJ64MC510, PIC33FJ64MC706,
  PIC33FJ64MC710.

Please note that we did not have the possibility to directly test *all* the possible devices produced by Microchip. In general this is not a problem, because the various devices are directly mapped to appropriate compiler flags. The following is the list of devices we tested directly: PIC30F2010, PIC30F6014A, PIC33FJ256GP710, PIC24FJ128GA010, which basically are the devices mounted on the Microchip Evaluation boards supported by Erika Enterprise.

If the device is not supported by the particular version of RT-Druid or if the developer needs to use a custom file, then the four files can be specified separately as in the following example:

```
...
MCU_DATA = PIC30 {
  MODEL = CUSTOM {
    MODEL = "33FJ256GP710";
    LINKERSCRIPT = "p33FJ256GP710.gld";
    DEV_LIB = "libp33FJ256GP710-elf.a";
    INCLUDE_C = "p33FJ256GP710.h";
    INCLUDE_S = "p33FJ256GP710.inc";
  };
};
...
```

As a result of this specification, the correct include files, libraries and linker scripts will be used when compiling an Erika Enterprise application.

2.4 Configuring the EDF scheduler

When configuring the EDF kernel for an Erika Enterprise application, the user has the possibility to specify the tick length in the OIL file to allow the specification of a relative deadline using a temporal value.

In particular, the user can specify a tick value as follows:

```
KERNEL_TYPE = EDF {TICK_TIME = "25ns";};
```

and then specify a relative deadline using a timing value as follows:

```
TASK myTask1 {
  REL_DEADLINE = "10ms";
};
```

The RT-Druid code generator will handle the the conversion between the relative deadline value in the corresponding timing value automatically.

The important thing in this process is to correctly specify the `TICK_TIME`. In general, that value depends on the timing reference which is made available by the dsPIC (R) DSC. The current timing reference implemented in the EDF kernel is based on the value of a 32 bit timer.

Having a 32 bit timing reference helps implementing a long lifetime for the circular timer, allowing the support of relatively long relative deadlines.

The 32 bit timer is obtained by concatenating the two 16 bit timers `TMR8` and `TMR9` available on the dsPIC (R) DSC. The clock used for the timers is the system clock, with a tick time equal to $\frac{1}{F_{cy}}$, where $F_{cy} = \frac{F_{osc}}{2}$. F_{osc}, the frequency of the oscillator, depends on its configuration which is typically done in the application by using appropriate macros shown below.

The following paragraphs describe two typical oscillator values which can be used, and the correspondent parameters which has to be put in the OIL file. The same settings are available in two template applications distributed with Erika Enterprise which work on the FLEX boards featuring a dsPIC (R) DSC model `PIC33FJ256MC710`.

Finally, always remember that the `EE_time_init` function has to be called inside the `main` function before using any primitive of the EDF kernel.

2.4.1 Primary oscillator without PLL

In this configuration, a primary oscillator at 4 MHz is used without any moltiplication. In this case, the tick duration is 500 ns, and the dsPIC (R) DSC is running at 2 MIPS.

To configure the system in this way, the C language has to contain the following compiler directive:

```
_FOSCSEL(FNOSC_PRI);
```

The OIL file will contain the following line:

```
KERNEL_TYPE = EDF {TICK_TIME = "500ns";};
```

The maximum relative deadline which can be expressed in the OIL file is $500ns * 2^{31}$, which is slightly more than 1073 secs.

A template application which initializes an EDF periodic task for this case is available under the RT-Druid templates.

2.4.2 Primary oscillator with PLL

In this configuration, the dsPIC (R) DSC is configured to provide its maximum computational power. To do that, the internal PLL is used to push the internal frequency F_{osc} to 80 MHz. As a consequence, F_{cy} become 40 MHz, which is the nominal maximum computational power of 40 MIPS declared by Microchip. In this case, the tick duration is 25 ns.

To configure the system in this way, the C language has to contain the following compiler directive:

```
_FOSCSEL(FNOSC_PRIPLL);
```

Moreover, at the beginning of the `main` function, the PLL multiplier registers needs to be set with the following code:

```
/* Clock setup for 40MIPS */
CLKDIVbits.DOZEN   = 0;
CLKDIVbits.PLLPRE  = 0;
CLKDIVbits.PLLPOST = 0;
PLLFBDbits.PLLDIV  = 78;

/* Wait for PLL to lock */
while(OSCCONbits.LOCK!=1);
```

Finally, the OIL file will contain the following line:

```
KERNEL_TYPE = EDF {TICK_TIME = "25ns";};
```

The maximum relative deadline which can be expressed in the OIL file is $25ns * 2^{31}$, which is slightly more than 53 secs.

2.4.3 EE_time_init

Synopsis

```
void EE_time_init(void);
```

Description

The function programs `TMR8` and `TMR9` as a 32 bit timer which is then used by the EDF Kernel to take the timing reference.

The function *must* be called before calling any Erika Enterprise primitive.

3 Flex Board

3.1 Introduction

This chapter describes the support done in Erika Enterprise for the Evidence/Embedded solutions Flex Board.

Flex is an embedded board which can be used by all the developers who want to fully exploit the potential of the latest Microchip micro-controllers: the dsPIC (R) DSC family.

Flex is born as a development board where to easily develop and test real-time applications for the dsPIC (R) DSC micro-controller. The main features of Flex are:

- robust electronic design;
- modular architecture;
- availability of a growing number of application notes;
- the full support of Erika Enterprise.

To configure the usage of the Flex Board, the user has to specify an appropriate `BOARD_DATA`, as in the following example:

```
...
BOARD_DATA = EE_FLEX {
  ...
}
...
```

The Flex board supports a set of devices which are directly mounted on it, plus a set of additional devices mounted on specific add-on boards. These devices can be configured by adding attributes inside the `BOARD_DATA` section.

The supported devices and the API functions needed to use them are described in the following sections.

> **Warning:** The current version of the board support for Flex only supports the dsPIC (R) DSC model 33FJ256GP710.

3.2 System LED

The Flex Board has a system LED attached to a GPIO pin of the microcontroller. To use the system LED on the Flex Board, the developer should specify the `USELEDS` attribute as TRUE, as in the following example:

```
...
BOARD_DATA = EE_FLEX {
  USELEDS = TRUE;
  ...
}
...
```

The following subsections will describe the functions available to control the Flex System LED.

3.2.1 EE_leds_init

Synopsis

```
void EE_leds_init(void);
```

Description

The function configures the GPIO pin. The LED starts turned off.

3.2.2 EE_led_sys_on

Synopsis

```
void EE_led_sys_on(void);
```

Description

The function turns on the LED.

3.2.3 EE_led_sys_off

Synopsis

```
void EE_led_sys_off(void);
```

Description

The function turns off the LED.

3.2.4 EE_led_on

Synopsis

```
void EE_led_on(void);
```

Description

The function turns on the LED.

3.2.5 EE_led_off

Synopsis

```
void EE_led_off(void);
```

Description

The function turns off the LED.

4 Flex Demo Daughter Board

4.1 Introduction

This chapter describes the support done in Erika Enterprise for the Evidence/Embedded solutions Flex Demo Daughter Board.

The Demo Daughter board is a small board that plugs on the Flex Light / Flex Full connectors and which provides a set of devices useful to implement small demos and demonstrators of control algorithms.

The main features of the Flex Demo Daughter board are:

- 8 leds;
- 4 buttons;
- 1 accelerometer;
- 8 Analog Inputs;
- 1 buzzer;
- 1 DAC;
- direct availability of the MCU encoder pins
- Infrared receiver;
- 16x2 characters LCD;
- PWM outputs;
- Temperature sensor;
- Light sensor;
- Trimmer;
- Connections to implement the USB communication on the FLEX Full
- Zigbee connector for Microchip or Easybee modules.

To configure the usage of the Flex Demo Daughter Board, the user has to specify an appropriate `BOARD_DATA`, as in the following example:

```
...
BOARD_DATA = EE_FLEX {
  TYPE = DEMO {
      ...
  };
  ...
}
...
```

The supported devices and the API functions needed to use them are described in the following sections.

4.2 LEDS

The Flex Demo Daughter board hosts 8 LEDs which are attached to GPIO pins. To use the LEDs on the Flex Demo Daughter Board, the developer should include the following fragment in the application OIL file:

```
...
BOARD_DATA = EE_FLEX {
  TYPE = DEMO { OPTIONS = LEDS; };
  ...
}
...
```

The following subsections will describe the functions available to control the Flex Demo Daughter Board LEDs.

4.2.1 EE_demoboard_leds_init

Synopsis

```
void EE_demoboard_leds_init(void);
```

Description

The function configures the LEDs, which starts turned off.

4.2.2 EE_leds

Synopsis

```
void EE_leds(EE_UINT8 data);
```

Description

The function sets the led values using the `data` parameter.

4.2.3 EE_leds_on

Synopsis

```
void EE_leds_on(void);
```

Description

The function turns all the LEDs on.

4.2.4 EE_leds_off

Synopsis

```
void EE_leds_off(void);
```

Description

The function turns all the LEDs off.

4.2.5 EE_led_0_on

Synopsis

```
void EE_led_0_on(void);
```

Description

The function turns LED 0 on.

4.2.6 EE_led_0_off

Synopsis

```
void EE_led_0_off(void);
```

Description

The function turns LED 0 off.

4.2.7 EE_led_1_on

Synopsis

```
void EE_led_1_on(void);
```

Description

The function turns LED 1 on.

4.2.8 EE_led_1_off

Synopsis

```
void EE_led_1_off(void);
```

Description

The function turns LED 1 off.

4.2.9 EE_led_2_on

Synopsis

```
void EE_led_2_on(void);
```

Description

The function turns LED 2 on.

4.2.10 EE_led_2_off

Synopsis

```
void EE_led_2_off(void);
```

Description

The function turns LED 2 off.

4.2.11 EE_led_3_on

Synopsis

```
void EE_led_3_on(void);
```

Description

The function turns LED 3 on.

4.2.12 EE_led_3_off

Synopsis

```
void EE_led_3_off(void);
```

Description

The function turns LED 3 off.

4.2.13 EE_led_4_on

Synopsis

```
void EE_led_4_on(void);
```

Description

The function turns LED 4 on.

4.2.14 EE_led_4_off

Synopsis

```
void EE_led_4_off(void);
```

Description

The function turns LED 4 off.

4.2.15 EE_led_5_on

Synopsis

```
void EE_led_5_on(void);
```

Description

The function turns LED 5 on.

4.2.16 EE_led_5_off

Synopsis

```
void EE_led_5_off(void);
```

Description

The function turns LED 5 off.

4.2.17 EE_led_6_on

Synopsis

```
void EE_led_6_on(void);
```

Description

The function turns LED 6 on.

4.2.18 EE_led_6_off

Synopsis

```
void EE_led_6_off(void);
```

Description

The function turns LED 6 off.

4.2.19 EE_led_7_on

Synopsis

```
void EE_led_7_on(void);
```

Description

The function turns LED 7 on.

4.2.20 EE_led_7_off

Synopsis

```
void EE_led_7_off(void);
```

Description

The function turns LED 7 off.

4.3 Buttons

The Flex Demo Daughter Board has a set of four buttons attached to GPIO pins of the microcontroller. To use the buttons, the developer should include the following fragment in the application OIL file:

```
...
BOARD_DATA = EE_FLEX {
  TYPE = DEMO { OPTIONS = BUTTONS; };
  ...
}
...
```

The following subsections will describe the functions available to control the Flex Demo Daughter Board buttons.

4.3.1 EE_buttons_init

Synopsis

```
void EE_buttons_init( void (*isr_callback)(void), EE_UINT8 mask );
```

Description

The function configures the GPIO pins used by the buttons. Buttons can be configured to be controlled only by using polling functions (no `isr_callback` is specified), or can be configured to raise an interrupt (if `isr_callback` is specified).

When the `isr_callback` is specified, the `mask` parameter is used to control for which buttons the interrupt will be generated.

Parameters

- `isr_callback` The function is called inside an ISR2 upon a button press.
- `mask` If `isr_callback` is specified, then this parameter controls which buttons will generate an interrupt request. In particular, bit `0x01` is used for button S1, bit `0x02` is used for button S2, bit `0x04` is used for button S3, bit `0x08` is used for button S4.

Return Values

- `void` The function does not return a value.

4.3.2 EE_button_get_S1

Synopsis

```
EE_UINT8 EE_button_get_S1(void);
```

Description

The function returns the status of the button number S1.

Return Values

- `unsigned char` 1 of the button is pressed, 0 otherwise.

4.3.3 EE_button_get_S2

Synopsis

```
EE_UINT8 EE_button_get_S2(void);
```

Description

The function returns the status of the button number S2.

Return Values

- `unsigned char` 1 of the button is pressed, 0 otherwise.

4.3.4 EE_button_get_S3

Synopsis

```
EE_UINT8 EE_button_get_S3(void);
```

Description

The function returns the status of the button number S3.

Return Values

- `unsigned char` 1 of the button is pressed, 0 otherwise.

4.3.5 EE_button_get_S4

Synopsis

```
EE_UINT8 EE_button_get_S4(void);
```

Description

The function returns the status of the button number S4.

Return Values

- `unsigned char` 1 of the button is pressed, 0 otherwise.

4.4 LCD

The Flex Demo Daughter Board has an alpha-numeric 16 x 2 LCD display mounted on the board attached to the GPIO pins of the microcontroller. To use the LCD on the board, the developer should include the following fragment in the application OIL file:

```
...
BOARD_DATA = EE_FLEX {
  TYPE = DEMO { OPTIONS = LCD; };
  ...
}
...
```

The functions available can be used to print character and strings to the LCD Display, and to select the current cursor position (which is the position where the next character will be printed).

To specify a character position in the LCD, the functions provided uses an integer. Numbers from 0 to 15 represents the first line, whereas numbers from 15 to 31 represents the second line.

The following subsections will describe the functions available to control the Explorer 16 LCD.

4.4.1 EE_lcd_init

Synopsis

```
void EE_lcd_init(void);
```

Description

The function initializes the LCD display.

4.4.2 EE_lcd_command

Synopsis

```
void EE_lcd_command(EE_UINT8 cmd);
```

Description

The function sends a command to the LCD. Most of the LCD functions described in this chapters basically remap to this function. The developer can use this function to implement features which are currently not supported by the LCD API.

Parameters

- `cmd` The LCD command.

4.4.3 EE_lcd_putc

Synopsis

```
void EE_lcd_putc(EE_INT8 data);
```

Description

The function puts a character on the LCD display, at the current cursor position.

4.4.4 EE_lcd_puts

Synopsis

```
void EE_lcd_puts(EE_INT8 *buf);
```

Description

The function prints a string to the display.

Parameters

- `buf` The string to display. It must be a valid C-language string.

4.4.5 EE_lcd_busy

Synopsis

```
unsigned char EE_lcd_busy(void);
```

Description

The function returns 1 if the display is busy, 0 otherwise. This function can be used to check if the application can send a new command to the LCD, or if the command can not be sent because the LCD is still busy processing the previous command.

Return Values

- `unsigned char` 1 if the display is busy, 0 otherwise.

4.4.6 EE_lcd_clear

Synopsis

```
void EE_lcd_clear(void);
```

Description

The function clears the LCD.

4.4.7 EE_lcd_home

Synopsis

```
void EE_lcd_home(void);
```

Description

The function sets the current cursor position to the top left display character.

4.4.8 EE_lcd_line2

Synopsis

```
void EE_lcd_line2(void);
```

Description

The function sets the current cursor position to the bottom left display character.

4.4.9 EE_lcd_curs_right

Synopsis

```
void EE_lcd_curs_right(void);
```

Description

The function sets the current cursor position on the next character on the right.

4.4.10 EE_lcd_curs_left

Synopsis

```
void EE_lcd_curs_left(void);
```

Description

The function sets the current cursor position on the next character on the left.

4.4.11 EE_lcd_shift

Synopsis

```
void EE_lcd_shift(void);
```

Description

The function can be used to enable the shift mode of the LCD. When in shift mode, each character sent provoke the shifting of all the characters of the LCD.

4.4.12 EE_lcd_goto

Synopsis

```
void EE_lcd_goto(EE_UINT8 posx, EE_UINT8 posy);
```

Description

The function sets the current cursor position to $(posx, posy)$.

Parameters

- `posx` The LCD column, from 0 to 15.
- `posy` The LCD row, 0 or 1.

4.5 Analog sensors

The Flex Demo Daughter Board has a set of analog channels available, in particular (in parenthesis the pin assignments):

- Temperature sensor (AN12/RB12);
- Light sensor (AN13/RB13);
- Trimmer (AN15/RB15);
- Accelerometer X axis (AN16/RC1);
- Accelerometer Y axis (AN17/RC2);
- Accelerometer Z axis (AN18/RC3);

- ADC Aux (AN19/RC4).

To use these inputs, the developer should include the following fragment in the application OIL file:

```
...
BOARD_DATA = EE_FLEX {
  TYPE = DEMO {
    OPTIONS = ADC_IN;         // for the analog inputs
    OPTIONS = ACCELEROMETER; // for the accelerometer
    OPTIONS = SENSORS;        // for the sensors
    OPTIONS = TRIMMER;        // for the potentiometer
};
...
```

The functions available can be used to start and stop the A/D converter, and to read the values from the various sensors.

4.5.1 EE_analog_init

Synopsis

```
void EE_analog_init(void);
```

Description

The function initializes the A/D converter. The ADC is initialized in polling mode.

4.5.2 EE_analog_close

Synopsis

```
void EE_analog_close(void);
```

Description

The function turns off the A/D converter.

4.5.3 EE_adcin_init

Synopsis

```
void EE_adcin_init(void);
```

Description

The function initializes the A/D converter. It has the same functionality as `EE_analog_init`.

4.5.4 EE_adcin_get_volt

Synopsis

```
EE_UINT16 EE_adcin_get_volt(void);
```

Description

The function reads the ADC Aux channel and returns its value. The A/D converter should have been already initialized using `EE_adcin_init`.

Return Values

- `EE_UINT16` The voltage read from the ADC Aux channel, in millivolt.

4.5.5 EE_trimmer_init

Synopsis

```
void EE_trimmer_init(void);
```

Description

The function initializes the A/D converter. It has the same functionality as `EE_analog_init`.

4.5.6 EE_trimmer_get_volt

Synopsis

```
EE_UINT16 EE_trimmer_get_volt(void);
```

Description

The function reads the Trimmer channel and returns its value. The A/D converter should have been already initialized using `EE_adcin_init`.

Return Values

- `EE_UINT16` The voltage read from the Trimmer channel, in millivolt.

4.5.7 EE_analogsensors_init

Synopsis

```
void EE_analogsensors_init(void);
```

Description

The function initializes the A/D converter. It has the same functionality as `EE_analog_init`.

4.5.8 EE_analog_get_temperature

Synopsis

```
EE_UINT16 EE_nalog_get_temperature(void);
```

Description

The function reads the temperature sensor and returns its value. The A/D converter should have been already initialized using `EE_adcin_init`.

Return Values

- `EE_UINT16` The voltage read from the temperature sensor, in millivolt.

4.5.9 EE_analog_get_light

Synopsis

```
EE_UINT16 EE_analog_get_light(void);
```

Description

The function reads the Light sensor and returns its value. The A/D converter should have been already initialized using `EE_adcin_init`.

Return Values

- `EE_UINT16` The voltage read from the Light sensor, in millivolt.

4.5.10 EE_accelerometer_init

Synopsis

```
void EE_accelerometer_init(void);
```

Description

The function initializes the A/D converter. It has the same functionality as `EE_analog_init`, plus some initialization specific for the 3-axis accelerometer..

4.5.11 EE_accelerometer_getglevel

Synopsis

```
EE_UINT8 EE_accelerometer_getglevel(void);
```

Description

Description to be done.

Return Values

- EE_UINT8

4.5.12 EE_accelerometer_setglevel

Synopsis

```
void EE_accelerometer_setglevel(EE_UINT8 level);
```

Description

Description to be done.

Return Values

- EE_UINT8

4.5.13 EE_accelerometer_sleep

Synopsis

```
void EE_accelerometer_sleep(void);
```

Description

Description to be done.

Return Values

- EE_UINT8

4.5.14 EE_accelerometer_wakeup

Synopsis

```
void EE_accelerometer_wakeup(void);
```

Description

Description to be done.

Return Values

- `EE_UINT8`

4.5.15 EE_accelerometer_gety

Synopsis

```
float EE_accelerometer_gety(void);
```

Description

Description to be done.

Return Values

- `EE_UINT8`

4.5.16 EE_accelerometer_getz

Synopsis

```
float EE_accelerometer_getz(void);
```

Description

Description to be done.

Return Values

- `EE_UINT8`

4.6 Buzzer

The Flex Demo Daughter Board has a buzzer which can be used to produce simple sounds.

To use the buzzer, the developer should include the following fragment in the application OIL file:

```
...
BOARD_DATA = EE_FLEX {
  TYPE = DEMO {
    OPTIONS = BUZZER;         // for the analog inputs
```

```
};
...
```

The functions available can be used to setup the buzzer, and to play notes.

> **Warning:** The buzzer driver uses timer T3.

4.6.1 EE_buzzer_init

Synopsis

```
void EE_buzzer_init(void);
```

Description

The function initializes the buzzer.

4.6.2 EE_buzzer_set_freq

Synopsis

```
void EE_buzzer_set_freq(EE_UINT16 new_freq);
```

Description

The function sets an output frequency for the buzzer. Frequencies should be higher than 10 Hz. No action is taken if the new frequency differs from the previous one by less than 10 Hz.

Parameters

- `new_freq` The new buzzer frequency, in Hz.

4.6.3 EE_buzzer_get_freq

Synopsis

```
EE_UINT16 EE_buzzer_get_freq(void);
```

Description

The function returns the current buzer frequency.

Return Values

- `EE_UINT16` The current buzzer frequency.

4.6.4 EE_buzzer_mute

Synopsis

```
void EE_buzzer_mute(void);
```

Description

The function mutes the buzzer.

4.6.5 EE_buzzer_unmute

Synopsis

```
void EE_buzzer_unmute(void);
```

Description

The function unmutes the buzzer.

4.6.6 EE_buzzer_close

Synopsis

```
void EE_buzzer_close(void);
```

Description

The function resets the buzzer.

4.7 PWM Output

The Flex Demo Daughter Board has a PWM output which is attached to the Output Compare 3 of the Timer 2.

To use the PWM, the developer should include the following fragment in the application OIL file:

```
...
BOARD_DATA = EE_FLEX {
  TYPE = DEMO {
    OPTIONS = PWM_OUT;
};
...
```

The functions available can be used to start, stop and set the duty cycle of the PWM.

4.7.1 EE_pwm_init

Synopsis

```
void EE_pwm_init(EE_UINT16 Period);
```

Description

The function initializes the PWM, setting also its period.

Parameters

- `Period` The PWM period.

4.7.2 EE_pwm_set_duty

Synopsis

```
void EE_pwm_set_duty(float duty);
```

Description

The function sets the duty cycle of the PWM.

Parameters

- `duty` The PWM duty cycle.

4.7.3 EE_pwm_close

Synopsis

```
void EE_pwm_close(void);
```

Description

The function shuts down the PWM.

4.8 DAC Output

The Flex Demo Daughter Board has a DAC output connected to the Microcontroller I2C port which can be used to convert digital signals to analog values.

To use the DAC, the developer should include the following fragment in the application OIL file:

```
...
BOARD_DATA = EE_FLEX {
  TYPE = DEMO {
    OPTIONS = DAC;
};
...
```

The functions available can be used to start, stop and set the duty cycle of the PWM.

4.8.1 EE_dac_general_call

Synopsis

```
EE_INT8 EE_dac_general_call(EE_UINT8 second);
```

Description

Description to be done.

Parameters

- second

Return Values

- EE_INT8

4.8.2 EE_dac_fast_write

Synopsis

```
EE_INT8 EE_dac_fast_write(EE_UINT16 data, EE_UINT8 port, EE_UINT8 power);
```

Description

Description to be done.

Parameters

- data
- port
- power

Return Values

- EE_INT8

4.8.3 EE_dac_write

Synopsis

```
EE_INT8 EE_dac_write(EE_UINT16 data, EE_UINT8 port, EE_UINT8 power, EE_UINT8 save);
```

Description

Description to be done.

Parameters

- `data`
- `port`
- `power`
- `save`

Return Values

- `EE_INT8`

4.8.4 EE_dac_init

Synopsis

```
void EE_dac_init(void);
```

Description

Description to be done.

5 Explorer16 Board

5.1 Introduction

This chapter describes the support done in Erika Enterprise for the Microchip Explorer16 Board (see Figure 5.1).

The Explorer 16 is a low cost, efficient development board produced by Microchip hosting an alpha-numeric 16 x 2 LCD display, with interfaces to MPLAB ICD 2, USB, and RS-232 [4].

To configure the usage of the Explorer 16 Board, the user has to specify an appropriate `BOARD_DATA`, as in the following example:

```
...
BOARD_DATA = MICROCHIP_EXPLORER16 {
  ...
}
...
```

The Explorer 16 board supports a set of devices which are directly mounted on it. These devices can be configured by adding attributes inside the `BOARD_DATA` section.

The supported devices and the API functions needed to use them are described in the following sections.

The current version of the board support for Flex supports both the dsPIC (R) DSC model 33FJ256GP710 and the PIC24 model 24FJ128GA010.

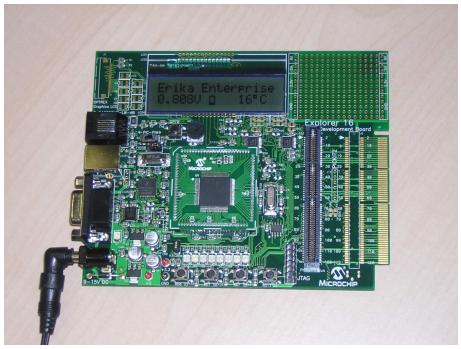

Figure 5.1: The Microchip Explorer 16 board running Erika Enterprise.

5.2 Buttons

The Explorer 16 Board has a set of four buttons attached to GPIO pins of the microcontroller. To use the buttons on the Explorer 16 Board, the developer should specify the `USEBUTTONS` attribute as TRUE, as in the following example:

```
...
BOARD_DATA = MICROCHIP_EXPLORER16 {
  USEBUTTONS = TRUE;
  ...
}
...
```

The following subsections will describe the functions available to control the Explorer 16 buttons.

5.2.1 EE_buttons_init

Synopsis

```
void EE_buttons_init( void (*isr_callback)(void), EE_UINT8 mask );
```

Description

The function configures the GPIO pins used by the buttons. Buttons can be configured to be controlled only by using polling functions (no `isr_callback` is specified), or can be configured to raise an interrupt (if `isr_callback` is specified).

When the `isr_callback` is specified, the `mask` parameter is used to control for which buttons the interrupt will be generated.

Parameters

- `isr_callback` The function is called inside an ISR2 upon a button press.
- `mask` If `isr_callback` is specified, then this parameter controls which buttons will generate an interrupt request. In particular, bit `0x01` is used for button S3, bit `0x02` is used for button S4, bit `0x04` is used for button S5, bit `0x08` is used for button S6.

Return Values

- `void` The function does not return a value.

5.2.2 EE_button_get_S3

Synopsis

```
EE_UINT8 EE_button_get_S3(void);
```

Description

The function returns the status of the button number S3.

Return Values

- `unsigned char` 1 of the button is pressed, 0 otherwise.

5.2.3 EE_button_get_S4

Synopsis

```
EE_UINT8 EE_button_get_S4(void);
```

Description

The function returns the status of the button number S4.

Return Values

- `unsigned char` 1 of the button is pressed, 0 otherwise.

5.2.4 EE_button_get_S5

Synopsis

```
EE_UINT8 EE_button_get_S5(void);
```

Description

The function returns the status of the button number S5.

Return Values

- `unsigned char` 1 of the button is pressed, 0 otherwise.

5.2.5 EE_button_get_S6

Synopsis

```
EE_UINT8 EE_button_get_S6(void);
```

Description

The function returns the status of the button number S6.

Return Values

- `unsigned char` 1 of the button is pressed, 0 otherwise.

5.3 LEDs

The Explorer 16 Board has a set of 8 LEDs attached to the GPIO pins of the microcontroller. To use the LEDs on the Explorer 16 Board, the developer should specify the `USELEDS` attribute as TRUE, as in the following example:

```
...
BOARD_DATA = MICROCHIP_EXPLORER16 {
  USELEDS = TRUE;
  ...
}
...
```

The following subsections will describe the functions available to control the Explorer 16 LEDs.

5.3.1 EE_leds_init

Synopsis

```
void EE_leds_init(void);
```

Description

The function configures the GPIO pins. The LEDs start turned off.

5.3.2 EE_leds_on

Synopsis

```
void EE_leds_on(void);
```

Description

The function turns on all the LEDs.

5.3.3 EE_leds_off

Synopsis

```
void EE_leds_off(void);
```

Description

The function turns off all the LEDs.

5.3.4 EE_led_on

Synopsis

```
void EE_led_on(void);
```

Description

The function turns on LED 3 (which is the first led on the board).

5.3.5 EE_led_off

Synopsis

```
void EE_led_off(void);
```

Description

The function turns off LED 3 (which is the first led on the board).

5.3.6 EE_led_3_on

Synopsis

```
void EE_led_3_on(void);
```

Description

The function turns on LED 3.

5.3.7 EE_led_3_off

Synopsis

```
void EE_led_3_off(void);
```

Description

The function turns off LED 3.

5.3.8 EE_led_4_on

Synopsis

```
void EE_led_4_on(void);
```

Description

The function turns on LED 4.

5.3.9 EE_led_4_off

Synopsis

```
void EE_led_4_off(void);
```

Description

The function turns off LED 4.

5.3.10 EE_led_5_on

Synopsis

```
void EE_led_5_on(void);
```

Description

The function turns on LED 5.

5.3.11 EE_led_5_off

Synopsis

```
void EE_led_5_off(void);
```

Description

The function turns off LED 5.

5.3.12 EE_led_6_on

Synopsis

```
void EE_led_6_on(void);
```

Description

The function turns on LED 6.

5.3.13 EE_led_6_off

Synopsis

```
void EE_led_6_off(void);
```

Description

The function turns off LED 6.

5.3.14 EE_led_7_on

Synopsis

```
void EE_led_7_on(void);
```

Description

The function turns on LED 7.

5.3.15 EE_led_7_off

Synopsis

```
void EE_led_7_off(void);
```

Description

The function turns off LED 7.

5.3.16 EE_led_8_on

Synopsis

```
void EE_led_8_on(void);
```

Description

The function turns on LED 8.

5.3.17 EE_led_8_off

Synopsis

```
void EE_led_8_off(void);
```

Description

The function turns off LED 8.

5.3.18 EE_led_9_on

Synopsis

```
void EE_led_9_on(void);
```

Description

The function turns on LED 9.

5.3.19 EE_led_9_off

Synopsis

```
void EE_led_9_off(void);
```

Description

The function turns off LED 9.

5.3.20 EE_led_10_on

Synopsis

```
void EE_led_10_on(void);
```

Description

The function turns on LED 10.

5.3.21 EE_led_10_off

Synopsis

```
void EE_led_10_off(void);
```

Description

The function turns off LED 10.

5.4 LCD

The Explorer 16 Board has an alpha-numeric 16 x 2 LCD display mounted on the board attached to the GPIO pins of the microcontroller. To use the LCD on the Explorer 16 Board, the developer should specify the `USELCD` attribute as TRUE, as in the following example:

```
...
BOARD_DATA = MICROCHIP_EXPLORER16 {
  USELCD = TRUE;
  ...
}
...
```

The functions available can be used to print character and strings to the LCD Display, and to select the current cursor position (which is the position where the next character will be printed).

To specify a character position in the LCD, the functions provided uses an integer. Numbers from 0 to 15 represents the first line, whereas numbers from 15 to 31 represents the second line.

The following subsections will describe the functions available to control the Explorer 16 LCD.

5.4.1 EE_lcd_init

Synopsis

```
void EE_lcd_init(void);
```

Description

The function initializes the LCD display.

5.4.2 EE_lcd_command

Synopsis

```
void EE_lcd_command(EE_UINT8 cmd);
```

Description

The function sends a command to the LCD. Most of the LCD functions described in this chapters basically remap to this function. The developer can use this function to implement features which are currently not supported by the LCD API.

Parameters

- `cmd` The LCD command.

5.4.3 EE_lcd_putc

Synopsis

```
void EE_lcd_putc(EE_INT8 data);
```

Description

The function puts a character on the LCD display, at the current cursor position.

5.4.4 EE_lcd_getc

Synopsis

```
EE_INT8 EE_lcd_getc(void);
```

Description

The function returns the character which is present at the current cursor position.

Return Values

- `char` The character which is displayed at the current cursor position.

5.4.5 EE_lcd_puts

Synopsis

```
void EE_lcd_puts(EE_INT8 *buf);
```

Description

The function prints a string to the display.

Parameters

- `buf` The string to display. It must be a valid C-language string.

5.4.6 EE_lcd_busy

Synopsis

```
unsigned char EE_lcd_busy(void);
```

Description

The function returns 1 if the display is busy, 0 otherwise. This function can be used to check if the application can send a new command to the LCD, or if the command can not be sent because the LCD is still busy processing the previous command.

Return Values

- `unsigned char` 1 if the display is busy, 0 otherwise.

5.4.7 EE_lcd_clear

Synopsis

```
void EE_lcd_clear(void);
```

Description

The function clears the LCD.

5.4.8 EE_lcd_home

Synopsis

```
void EE_lcd_home(void);
```

Description

The function sets the current cursor position to the top left display character.

5.4.9 EE_lcd_line2

Synopsis

```
void EE_lcd_line2(void);
```

Description

The function sets the current cursor position to the bottom left display character.

5.4.10 EE_lcd_curs_right

Synopsis

```
void EE_lcd_curs_right(void);
```

Description

The function sets the current cursor position on the next character on the right.

5.4.11 EE_lcd_curs_left

Synopsis

```
void EE_lcd_curs_left(void);
```

Description

The function sets the current cursor position on the next character on the left.

5.4.12 EE_lcd_shift

Synopsis

```
void EE_lcd_shift(void);
```

Description

The function can be used to enable the shift mode of the LCD. When in shift mode, each character sent provoke the shifting of all the characters of the LCD.

5.4.13 EE_lcd_goto

Synopsis

```
void EE_lcd_goto(EE_UINT8 posx, EE_UINT8 posy);
```

Description

The function sets the current cursor position to $(posx, posy)$.

Parameters

- `posx` The LCD column, from 0 to 15.
- `posy` The LCD row, 0 or 1.

5.5 Analog sensors

The Explorer 16 Board has two analog inputs which are connected to the board. The first is a temperature sensor, whereas the second is a potentiometer.

To use these two analog inputs on the Explorer 16 Board, the developer should specify the `USEANALOG` attribute as TRUE, as in the following example:

```
...
BOARD_DATA = MICROCHIP_EXPLORER16 {
  USEANALOG = TRUE;
  ...
}
...
```

The functions available can be used to start and stop the A/D converter, and to read the values from the two sensors.

The following subsections will describe the functions available to control the Explorer 16 on-board sensors.

5.5.1 EE_analog_init

Synopsis

```
void EE_analog_init(void);
```

Description

The function initializes the A/D converter. As a result, a periodic interrupt is raised. An interrupt handler is also internally provided by the A/D handler to read the sensor values and make them available to the user using the functions `EE_analog_get_volt` and `EE_analog_get_temp`.

5.5.2 EE_analog_get_volt

Synopsis

```
EE_UINT16 EE_analog_get_volt(void);
```

Description

The function returns the last voltage read from the potentiometer installed on the Explorer 16 board. The A/D converter should have been already initialized using `EE_analog_init`.

Return Values

- `EE_UINT16` The voltage read from the potentiometer installed on the Explorer 16 board, in millivolt.

5.5.3 EE_analog_get_temp

Synopsis

```
EE_UINT16 EE_analog_get_temp(void);
```

Description

The function returns the last temperature read from the temperature sensor installed on the Explorer 16 board. The A/D converter should have been already initialized using `EE_analog_init`.

Return Values

- `EE_UINT16` The temperature read from the sensor, in Celsius degrees.

5.5.4 EE_analog_start

Synopsis

```
void EE_analog_start(void);
```

Description

The function turns on A/D conversion. Please note that the A/D conversion is automatically started when `EE_analog_init` is called.

5.5.5 EE_analog_stop

Synopsis

```
void EE_analog_stop(void);
```

Description

The function turns off A/D conversion. To turn on again the A/D conversion, the developer can call `EE_analog_start`.

6 dsPICDEM 1.1 Plus Board

6.1 Introduction

This chapter describes the support done in Erika Enterprise for the Microchip dsPICDEM 1.1 Plus Board (see Figure 6.1).

The dsPICDEM 1.1 Plus is a low cost, efficient development board produced by Microchip hosting a graphic display, with interfaces to MPLAB ICD 2, and RS-232 [3].

To configure the usage of the dsPICDEM 1.1 Plus Board, the user has to specify an appropriate `BOARD_DATA`, as in the following example:

```
...
BOARD_DATA = MICROCHIP_DSPICDEM11PLUS {
  ...
}
...
```

The dsPICDEM 1.1 Plus board supports a set of devices which are directly mounted on it. These devices can be configured by adding attributes inside the `BOARD_DATA` section.

The supported devices and the API functions needed to use them are described in the following sections.

6.2 Buttons

The dsPICDEM 1.1 Plus Board has a set of four buttons attached to GPIO pins of the microcontroller. To use the buttons on the board, the developer should specify the `USEBUTTONS` attribute as TRUE, as in the following example:

```
...
BOARD_DATA = MICROCHIP_DSPICDEM11PLUS {
  USEBUTTONS = TRUE;
  ...
}
...
```

The following subsections will describe the functions available to control the buttons.

6.2.1 EE_buttons_init

Synopsis

```
void EE_buttons_init( void (*isr_callback)(void), EE_UINT8 mask );
```

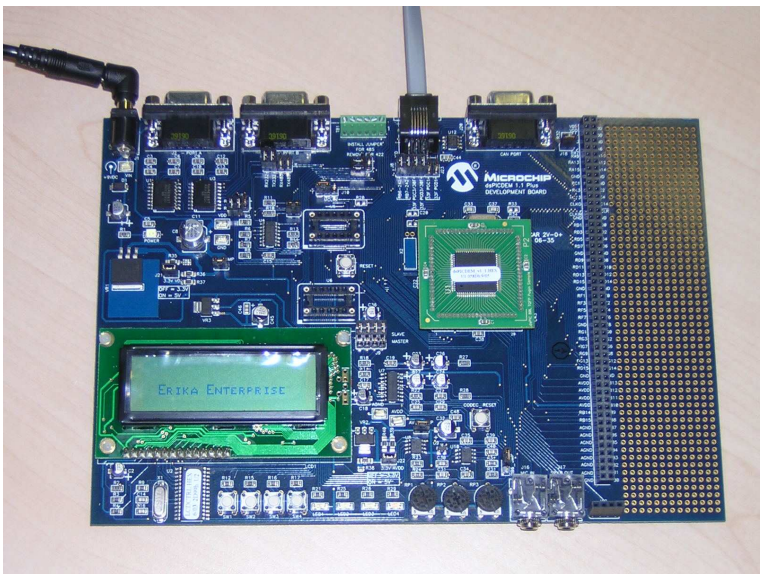

Figure 6.1: The Microchip dsPICDEM 1.1 Plus board running Erika Enterprise.

Description

The function configures the GPIO pins used by the buttons. Buttons can be configured to be controlled only by using polling functions (no `isr_callback` is specified), or can be configured to raise an interrupt (if `isr_callback` is specified).

When the `isr_callback` is specified, the `mask` parameter is used to control for which buttons the interrupt will be generated.

Parameters

- `isr_callback` The function is called inside an ISR2 upon a button press.
- `mask` If `isr_callback` is specified, then this parameter controls which buttons will generate an interrupt request. In particular, bit `0x01` is used for button S1, bit `0x02` is used for button S2, bit `0x04` is used for button S3, bit `0x08` is used for button S4.

Return Values

- `void` The function does not return a value.

6.2.2 EE_button_get_S1

Synopsis

```
EE_UINT8 EE_button_get_S1(void);
```

Description

The function returns the status of the button number S1.

Return Values

- `unsigned char` 1 of the button is pressed, 0 otherwise.

6.2.3 EE_button_get_S2

Synopsis

```
EE_UINT8 EE_button_get_S2(void);
```

Description

The function returns the status of the button number S2.

Return Values

- `unsigned char` 1 of the button is pressed, 0 otherwise.

6.2.4 EE_button_get_S3

Synopsis

```
EE_UINT8 EE_button_get_S3(void);
```

Description

The function returns the status of the button number S3.

Return Values

- `unsigned char` 1 of the button is pressed, 0 otherwise.

6.2.5 EE_button_get_S4

Synopsis

```
EE_UINT8 EE_button_get_S4(void);
```

Description

The function returns the status of the button number S4.

Return Values

- `unsigned char` 1 of the button is pressed, 0 otherwise.

6.3 LEDs

The dsPICDEM 1.1 Plus Board has a set of 4 LEDs attached to the GPIO pins of the microcontroller. To use the LEDs on the board, the developer should specify the `USELEDS` attribute as TRUE, as in the following example:

```
...
BOARD_DATA = MICROCHIP_DSPICDEM11PLUS {
  USELEDS = TRUE;
  ...
}
...
```

The following subsections will describe the functions available to control the LEDs.

6.3.1 EE_leds_init

Synopsis

```
void EE_leds_init(void);
```

Description

The function configures the GPIO pins. The LEDs start turned off.

6.3.2 EE_leds_on

Synopsis

```
void EE_leds_on(void);
```

Description

The function turns on all the LEDs.

6.3.3 EE_leds_off

Synopsis

```
void EE_leds_off(void);
```

Description

The function turns off all the LEDs.

6.3.4 EE_led_on

Synopsis

```
void EE_led_on(void);
```

Description

The function turns on LED 1.

6.3.5 EE_led_off

Synopsis

```
void EE_led_off(void);
```

Description

The function turns off LED 1.

6.3.6 EE_led_1_on

Synopsis

```
void EE_led_1_on(void);
```

Description

The function turns on LED 1.

6.3.7 EE_led_1_off

Synopsis

```
void EE_led_1_off(void);
```

Description

The function turns off LED 1.

6.3.8 EE_led_2_on

Synopsis

```
void EE_led_2_on(void);
```

Description

The function turns on LED 2.

6.3.9 EE_led_2_off

Synopsis

```
void EE_led_2_off(void);
```

Description

The function turns off LED 2.

6.3.10 EE_led_3_on

Synopsis

```
void EE_led_3_on(void);
```

Description

The function turns on LED 3.

6.3.11 EE_led_3_off

Synopsis

```
void EE_led_3_off(void);
```

Description

The function turns off LED 3.

6.3.12 EE_led_4_on

Synopsis

```
void EE_led_4_on(void);
```

Description

The function turns on LED 4.

6.3.13 EE_led_4_off

Synopsis

```
void EE_led_4_off(void);
```

Description

The function turns off LED 4.

6.4 LCD

The dsPICDEM 1.1 Plus Board has a 132x32 graphical LCD display mounted on the board attached to the GPIO pins of the microcontroller. To use the LCD, the developer should specify the `USELCD` attribute as TRUE, as in the following example:

```
...
BOARD_DATA = MICROCHIP_DSPICDEM11PLUS {
  USELCD = TRUE;
  ...
}
...
```

The functions available in Erika Enterprise are a direct implementation of the LCD commands described in the "LCD Controller specification" of the "dsPICDEM 1.1 development Board User's Guide". Please refer to that document to have a complete description of the LCD hardware.

The LCD has three data types, each based on its own independent coordinate systems. The data types are characters, pixels and columns. Associated with each coordinate system is a current position of which each is independent of the other.

The following subsections will describe the functions available to control the LCD.

6.4.1 EE_lcd_init

Synopsis

```
void EE_lcd_init(void);
```

Description

The function initializes the LCD display.

6.4.2 EE_lcd_command

Synopsis

```
void EE_lcd_command(EE_UINT8 cmd);
```

Description

The function sends a command to the LCD. Most of the LCD functions described in this chapter are similar to this function. The developer can use this function to implement features which are currently not supported by the LCD API.

Parameters

- `cmd` The LCD command.

6.4.3 EE_lcd_Reset

Synopsis

```
void EE_lcd_Reset( void );
```

Description

The function resets the LCD to its initial power-up state.

6.4.4 EE_lcd_Home

Synopsis

```
void EE_lcd_Home( void );
```

Description

The function sets all coordinate variables to their home values and leaves the display unchanged.

6.4.5 EE_lcd_HomeClear

Synopsis

```
void EE_lcd_HomeClear( void );
```

Description

The function clears the entire display and then sets all coordinate variables to their home values.

6.4.6 EE_lcd_Scroll

Synopsis

```
void EE_lcd_Scroll( EE_UINT8 lines );
```

Description

The function rolls the display in the vertical axis by the amount `lines`. The LCD data array consists of 32 lines of 122 pixels each. If scrolling, value `lines` is set to zero and the top row of the data array is displayed on the top row of the display.

Parameters

- `lines` The number of display lines to scroll.

6.4.7 EE_lcd_ChrPos

Synopsis

```
void EE_lcd_ChrPos( EE_UINT8 col, EE_UINT8 row );
```

Description

The function sets the character position to `col`, `row`. This command has no effect on the display except for moving character cursor if it is turned on.

Parameters

- `col` The new column position of the cursor.
- `row` The new row position of the cursor.

6.4.8 EE_lcd_ChrPosInc

Synopsis

```
void EE_lcd_ChrPosInc( void );
```

Description

The function increments the character position. This command has no effect on the display except for moving the character cursor if it is turned on.

6.4.9 EE_lcd_WrtChr

Synopsis

```
void EE_lcd_WrtChr( EE_UINT8 chr, EE_UINT8 col, EE_UINT8 row );
```

Description

The function sets the character position to (`col`,`row`), then writes the ASCII character `chr`.

Parameters

- `chr` The character to bet printed on the display.
- `col` The new column position of the cursor.
- `row` The new row position of the cursor.

6.4.10 EE_lcd_WrtChrInc

Synopsis

```
void EE_lcd_WrtChrInc( EE_UINT8 chr, EE_UINT8 col, EE_UINT8 row );
```

Description

The function sets the character position to (`col`,`row`), writes the ASCII character `chr`, and then increments the character position.

Parameters

- `chr` The character to be printed on the display.
- `col` The column position of the cursor.
- `row` The row position of the cursor.

6.4.11 EE_lcd_WrtChrNext

Synopsis

```
void EE_lcd_WrtChrNext( EE_UINT8 chr );
```

Description

The function command writes the ASCII character `chr` to the current character position, then increments it.

Parameters

- `chr` The character to be printed on the display.

6.4.12 EE_lcd_ChrClearRow

Synopsis

```
void EE_lcd_ChrClearRow( EE_UINT8 row );
```

Description

The function clears the entire `row` and leaves the current character column position = 0.

Parameters

- `row` The row to be cleared.

6.4.13 EE_lcd_ChrClearEOL

Synopsis

```
void EE_lcd_ChrClearEOL( void );
```

Description

The function clears the current row from the current location to the end of the line and leaves the character position unchanged.

6.4.14 EE_lcd_ChrCursorOff

Synopsis

```
void EE_lcd_ChrCursorOff( void );
```

Description

The function command turns off the character cursor.

6.4.15 EE_lcd_ChrCursorOn

Synopsis

```
void EE_lcd_ChrCursorOn( void );
```

Description

The function turns on the character cursor at the current character position.

6.4.16 EE_lcd_ChrCursorBlink

Synopsis

```
void EE_lcd_ChrCursorBlink( EE_UINT8 tick );
```

Description

The function controls cursor blinking. If the time is set to zero, the cursor will not blink, else the cursor blinks with equal on and off times, with the on time being given by the blink time.

Parameters

- `tick` From 0 to 7, controls the blinking time which is equal to $tick * 0.125 sec$.

6.4.17 EE_lcd_PixPos

Synopsis

```
void EE_lcd_PixPos( EE_UINT8 posx, EE_UINT8 posy );
```

Description

The function sets the current pixel position to (`posx`,`posy`) and leaves the display unchanged. This command is intended to be used in conjunction with the function `EE_lcd_PixLine`.

Parameters

- `posx` The x pixel position.
- `posy` The y pixel position.

6.4.18 EE_lcd_PixOn

Synopsis

```
void EE_lcd_PixOn( EE_UINT8 posx, EE_UINT8 posy );
```

Description

The function sets the pixel position to (`posx`,`posy`) and turns on the pixel at that location. This command does not increment the pixel position.

Parameters

- `posx` The x pixel position.
- `posy` The y pixel position.

6.4.19 EE_lcd_PixOff

Synopsis

```
void EE_lcd_PixOff( EE_UINT8 posx, EE_UINT8 posy );
```

Description

The function sets the pixel position to (posx,posy) and turns off the pixel at that location. This command does not increment the current pixel position.

Parameters

- posx The x position of the pixel to be turned off.
- posy The y position of the pixel to be turned off.

6.4.20 EE_lcd_PixLine

Synopsis

```
void EE_lcd_PixLine( EE_UINT8 posx, EE_UINT8 posy );
```

Description

The function draws a straight line from the current pixel position to the specified location and leaves the pixel position set to the new location.

Parameters

- posx The x position of the end of the line.
- posy The y position of the end of the line.

6.4.21 EE_lcd_ColPos

Synopsis

```
void EE_lcd_ColPos( EE_UINT8 col, EE_UINT8 row );
```

Description

The function sets the column position to (col,row).

Parameters

- col The column position of the column.
- row The row position of the column.

6.4.22 EE_lcd_WrtColNext

Synopsis

```
void EE_lcd_WrtColNext( EE_UINT8 data );
```

Description

The function writes column data to the current column position and then increments the column position.

Parameters

- `data` The data to be displayed.

6.4.23 EE_lcd_WrtColNextOR

Synopsis

```
void EE_lcd_WrtColNextOR( EE_UINT8 data );
```

Description

The function ORs column data with existing data and writes the result to the current column position, then increments it.

Parameters

- `data` The data to be displayed.

6.4.24 EE_lcd_WrtColNextAND

Synopsis

```
void EE_lcd_WrtColNextAND( EE_UINT8 data );
```

Description

The function ANDs column data with existing data and writes the result to the current column position, then increments it.

Parameters

- `data` The data to be displayed.

6.4.25 EE_lcd_WrtColNextXOR

Synopsis

```
void EE_lcd_WrtColNextXOR( EE_UINT8 data );
```

Description

The function XORs column data with existing data and writes the result to the current column position, then increments it.

Parameters

- `data` The data to be displayed.

6.4.26 EE_lcd_putc

Synopsis

```
void EE_lcd_putc(EE_INT8 data);
```

Description

The function puts a character on the LCD display, at the current cursor position. The function remaps to EE_lcd_WrtChrNext.

6.4.27 EE_lcd_home

Synopsis

```
void EE_lcd_home(void);
```

Description

The function sets the current cursor position to the top left display character. The function remaps to EE_lcd_Home.

6.4.28 EE_lcd_goto

Synopsis

```
void EE_lcd_goto(EE_UINT8 posx, EE_UINT8 posy);
```

Description

The function sets the current cursor position to $(posx, posy)$. The function remaps to EE_lcd_ChrPos.

Parameters

- `posx` The LCD column, from 0 to 15.
- `posy` The LCD row, 0 or 1.

6.4.29 EE_lcd_clear

Synopsis

```
void EE_lcd_clear(void);
```

Description

The function clears the LCD. The function remaps to `EE_lcd_HomeClear`.

7 History

Version	Comment
1.0.0	Initial revision of this document.
1.1.2	New dsPIC (R) DSC macro; updated several typos; automatic php generation added; new versioning system.
1.1.3	Updated device list, added new boards.
1.1.4	Added screenshot of the configuration parameters, added the OIL compilation parameters for SRC, VERBOSE e NODEPS.
1.1.5	Added description of the EDF implementation, timers and PLL configuration.
1.1.6	Typos.
1.1.7	Added demo board functions.
1.1.8	Updated screenshots to Erika Enterprise 1.4.3. Erika Enterprise Basic renamed to Erika Enterprise.
1.1.9	Updated MODEL for custom MCUs.

Bibliography

[1] Eclipse Consortium. The eclipse platform. http://www.eclipse.org, 2005.

[2] The Apache Software Foundation. The apache ant project. http://ant.apache.org, 2005.

[3] Arizona Microchip Inc. The dsPICDEM 1.1 Plus Development Board. http://www.microchip.com, 2006.

[4] Arizona Microchip Inc. The Explorer 16 Development Board. http://www.microchip.com, 2006.

Index