ERIKA Enterprise Manual for the Atmel AVR5 targets

AVR microcontrollers gets real-time!

version: 1.0.0
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.

This manual describes the porting details of the Erika Enterprise kernel for the AVR family of microcontrollers. The AVR family produced by Atmel represents a widely used 8-bit RISC microcontroller, with a full range of interfaces available.

1.0.1 Erika Enterprise **and** RT-Druid **for** AVR

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 AVR family.

Erika Enterprise provides AVR 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;

AVR integration features

- Installation setup which integrates the AVR gcc compiler and AVRStudio together with Evidence Erika Enterprise and RT-Druid;
- Support for the Atmel JTAG debugger;
- Full support for AVR microcontroller series avr5;
- Full support for the development boards Atmel STK50X boards;
- Full support of the RF230 RF solution from Atmel, with usage of the IEEE 802.15.4 MAC.
- Full support for Crossbow MIB5X0 boards used to program Mica motes;

1.0.2 Integration with other tools for AVR

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

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.

Erika Enterprise also supports the compiling environments provided by Atmel, providing also direct support for the programming and JTAG solutions of Atmel.

1.1 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 for the AVR 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.

As a final note, all the settings which are explained in this document apply both to Erika Enterprise if not otherwise stated.

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

2 Erika Enterprise for AVR devices

2.1 The RT-Druid and Erika Enterprise design flow

The typical development environment provided by Atmel for the software development for the AVR microcontrollers is composed by the Atmel AVR Studio. Atmel AVR Studio is 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, 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 the Linux platform, easing in this way the porting of the application to a free development environment such as Linux.

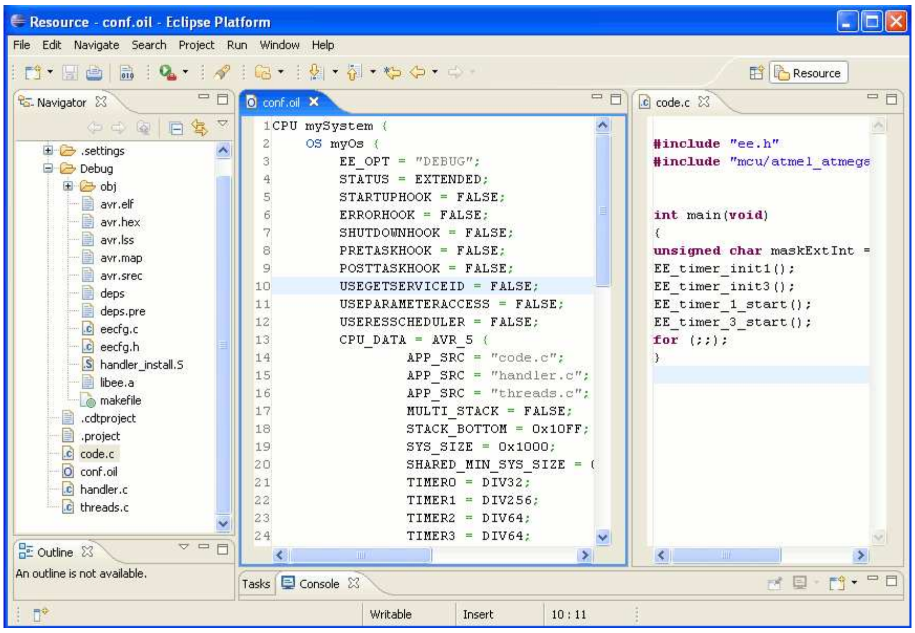
2.1.1 Building an application from command line

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 [4], which is the default scripting environment used in the Eclipse Framework.

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



An outline is not available.

```
CPU mySystem {
    OS myOs {
        EE_OPT = "DEBUG";
        STATUS = EXTENDED;
        STARTUPHOOK = FALSE;
        ERRORHOOK = FALSE;
        SHUTDOWNHOOK = FALSE;
        PRETASKHOOK = FALSE;
        POSTTASKHOOK = FALSE;
        USEGETSERVICEID = FALSE;
        USEPARAMETERACCESS = FALSE;
        USERESSCHEDULER = FALSE;
        CPU_DATA = AVR_5 {
                APP_SRC = "code.c";
                APP_SRC = "handler.c";
                APP_SRC = "threads.c";
                MULTI_STACK = FALSE;
                STACK_BOTTOM = 0x10FF;
                SYS_SIZE = 0x1000;
                SHARED_MIN_SYS_SIZE = (
                TIMER0 = DIV32;
                TIMER1 = DIV256;
                TIMER2 = DIV64;
                TIMER3 = DIV64;
```

```
#include "ee.h"
#include "mcu/atmel_atmega


int main(void)
{
unsigned char maskExtInt =
EE_timer_init1();
EE_timer_init3();
EE_timer_1_start();
EE_timer_3_start();
for (;;);
}
```

Figure 2.1: The Eclipse workspace and the RT-Druid plugins for AVR.

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 AVR

Erika Enterprise has been designed to be compiled using the GNU gcc toolchain. The AVR porting of Erika Enterprise in particular can be compiled using the GNU tools for AVR. The porting provides both the `binutils` package and the `gcc` package, plus a set of libraries which can be used to control the various peripherals provided by the AVR 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 AVRGCC package.

The GNU AVR-GCC. The GNU assembler, binutils and compiler are available as a part of the WinAVR suite *AVRGCC Compiler*, which is available as project on surceforge `http://winavr.sourceforge.net/`.

C Libraries. A set of libraries which can be used to control the peripherals implemented on the particular avr5 chip in use. These libraries are packaged together with the AVR-libc.

To compile an Erika Enterprise application, the development environment needs to be configured to correctly recognize the AVR-Gcc compiler. For doing so, please go to the "Preference" menu, as shown in Figure 2.2, and find the "RT-Druid/Oil/Avr5 Configurator" form as depicted in Figure 2.3. The first textbox, labeled `Gcc path`, refers to the installation directory of the AVR-Gcc compiler. The second and third textboxes are useful if you are using a development environment based on the Crossbow Mib5x0 board. In particular, the second textbox, labeled `Uisp path`, refers to the installation directory of the uisp programmer for Crossbow Mib5x0 board, and the third textbox, labeled `Serial port device`, refers to the COM serial port where the board is attached.

Warning: The install directories specified in the two textboxes of Figure 2.3 does *not* include the `bin` directory!
That is, `c:\WinAVR-20071221` is correct, wheras `c:\Programmi\WinAVR-20071221\bin` is not.

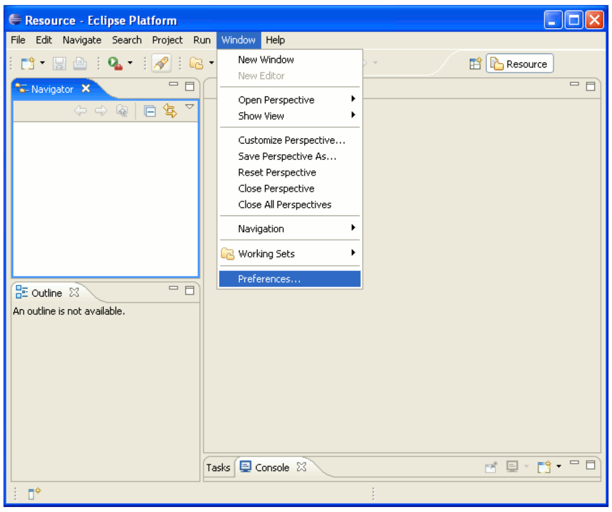
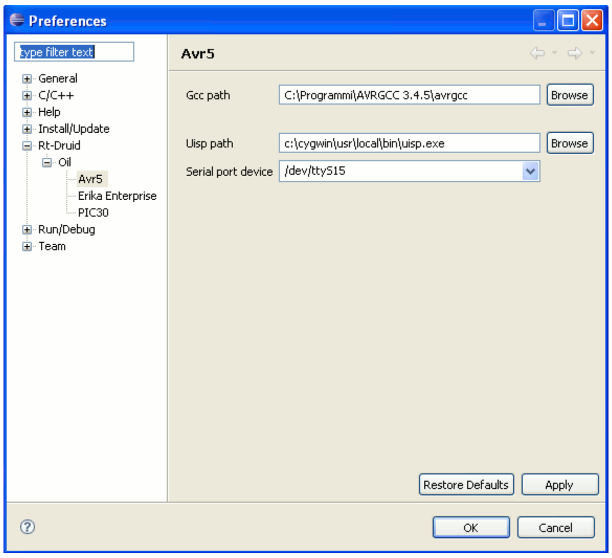


Figure 2.2: Go to the "Preference" menu.

2.3 Writing software for AVR using Erika Enterprise



Avr5

Gcc path: C:\Programmi\AVRGCC 3.4.5\avrgcc

Uisp path: c:\cygwin\usr\local\bin\uisp.exe

Serial port device: /dev/ttyS15

Figure 2.3: Select paths for compiler and assembler.

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

This section describes the details about the various configuration options which are available to create and compile an Erika Enterprise application for a AVR 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 Avr-Gcc compiler. In particular, from each .C file, a .SRC file is produced containing the corresponding assembler listing, which is then compiled by the Avr-Gcc 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 = AVR5 {
  ...
  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 = AVR5 {
  ...
  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 AVR architecture, the shared stack works as in the monostack configuration, that is it is allocated at the end of the SRAM data memory, growing towards lower 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 = AVR5 {
  ...
  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). In this example is showed also the task0 that uses a private stack.

```
...
CPU_DATA = AVR5 {
  ...
  MULTI_STACK = TRUE {
    IRQ_STACK = TRUE {
      SYS_SIZE=64;
    };
  };
};
...
TASK Task0 {
...
 STACK = PRIVATE {
      SYS_SIZE=64;
 };
...
};
...
```

2.3.6 Interrupt handling

Erika Enterprise for AVR 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*.

To declare an ISR of tipe 1 or 2, the user have to specify inside the OIL file a set of parameters: the interrupt source used by the interrupt, the type of ISR, and finally the handler function associated with the interrupt. For example if we want to declare a ISR type 2 associated to the timer1 overflow interrupt we have to type the following code inside the OIL file:

```
...
```

```
CPU_DATA = AVR5 {
  ...
  APP_SRC = "handler.c";
  ...
  MULTI_STACK = FALSE;
  HANDLER = HANDLER_T1_OVERFLW {
    FUNCTION = "myfunction";
    TYPE = 2;
  };
};
...
```

In the previous example the function `myfunction__isr2` is a simple C function, that is called everytime the timer1 overflow interrupt are raised. The strings that have to be used in the ATmega128 model of AVR5 family are:

```
 / * Interrupt Handlers  strings * /
  STRING HANDLER_IRQ0;        // external interrupt request 0
  STRING HANDLER_IRQ1;        // external interrupt request 1
  STRING HANDLER_IRQ2;        // external interrupt request 2
  STRING HANDLER_IRQ3;        // external interrupt request 3
  STRING HANDLER_IRQ4;        // external interrupt request 4
  STRING HANDLER_IRQ5;        // external interrupt request 5
  STRING HANDLER_IRQ6;        // external interrupt request 6
  STRING HANDLER_IRQ7;        // external interrupt request 7
  STRING HANDLER_T0_MATCH;    // Timer/Counter 0 Compare Match
  STRING HANDLER_T0_OVERFLW;  // Timer/Counter 0 Overflow
  STRING HANDLER_T1_EVENT;    // Timer/Counter 1 Capture Event
  STRING HANDLER_T1_MATCH_A;  // Timer/Counter 1 Compare Match A
  STRING HANDLER_T1_MATCH_B;  // Timer/Counter 1 Compare Match B
  STRING HANDLER_T1_MATCH_C;  // Timer/Counter 1 Compare Match C
  STRING HANDLER_T1_OVERFLW;  // Timer/Counter 1 Overflow
  STRING HANDLER_T2_MATCH;    // Timer/Counter 2 Compare Match
  STRING HANDLER_T2_OVERFLW;  // Timer/Counter 2 Overflow
  STRING HANDLER_T3_EVENT;    // Timer/Counter 3 Capture Event
  STRING HANDLER_T3_MATCH_A;  // Timer/Counter 3 Compare Match A
  STRING HANDLER_T3_MATCH_B;  // Timer/Counter 3 Compare Match B
  STRING HANDLER_T3_MATCH_C;  // Timer/Counter 3 Compare Match C
  STRING HANDLER_T3_OVERFLW;  // Timer/Counter 3 Overflow
  STRING HANDLER_SPI;         // SPI Serial Transfer Complete
  STRING HANDLER_US0_RX;      // USART0 Rx complete
  STRING HANDLER_US0_EMPTY;   // USART0 Data Register Empty
  STRING HANDLER_US0_TX;      // Usart0 Tx complete
  STRING HANDLER_US1_RX;      // USART1 Rx complete
  STRING HANDLER_US1_EMPTY;   // USART1 Data Register Empty
  STRING HANDLER_US1_TX;      // Usart1 Tx complete
  STRING HANDLER_ADC;         // ADC Conversion Complete
  STRING HANDLER_EEPROM;      // EEPROM Ready
```

```
    STRING HANDLER_ANALOG_COMP; // Analog Comparator
    STRING HANDLER_2WSI;        // Two-wire serial Interface
    STRING HANDLER_SPM_READY;   // Store program Memory Ready
```

Please remember that the AVR family of microcontroller has an interrupt vector table which is stored in the flash memory. The lowest addresses of the interrupt vector table is allocated to the Reset and to the interrupt vectors. Interrupt handlers placed at lower addresses has higher hardware priorities than interrupt handlers placed at higher addresses.

Writing an ISR 2 or ISR 1 with Erika Enterprise implies that :

- An assembler stub will be generated for the ISR. The ISR stub will have the name of the ISR (in the example, `__isr2_myfunction`). The assembler stub will call a C function named `myfunction`, which is provided by the user inside the application C files. The assembler stub will be attached to the interrupt of the peripheral (in the example, timer 1). 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 user.
- 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.

To use the external interrupt source the user must setting the external interrupt mask with the following function.

2.3.7 EE_IC_enable_external_IRQ

Synopsis

```
void EE_IC_enable_external_IRQ(EE_TYPEIRQMASK i);
```

Description

Enables the external interrupt source that are specified in the 8-bit bitmask. For example, if the third bit in the bitmask is set, the third external interrupt will be enabled.

2.3.8 EE_IC_clear_pending_IRQ

Synopsis

```
void EE_IC_clear_pending_IRQ(void);
```

Description

Clears ALL the pending interrupts.

2.3.9 AVR**timers**

The number of hardware timers in the AVRfamily is highly dependent on the particular chip used. For example, the ATmega128 device has 4 timers (2 8-bit timers and 2 16-bit timers).

The timers used in the application with the corresponding prescalers can be specified in the OIL file (these settings applies to AVR atmega128.

The following example show how to configure a set of timers for an AVR atmega128.

```
...
CPU_DATA = AVR5 {
  ...
  TIMER0 = DIV32;
  TIMER1 = DIV8;
  TIMER2 = DIV64;
  TIMER3 = DIV1;
  ...
};
...
```

The prescaler factor for timers in ATmega128 model are `DIV1`, `DIV8`, `DIV64`, `DIV256`, and `DIV1024`. The timer0 have also the prescaler factor `DIV32`. The User can use inside the source code of application the following functions to use the timer.

2.3.10 EE_timer_initx

Synopsis

`void EE_timer_initx(void);`

Description

Inizialize the timerx (x = 0,1,2,3)

2.3.11 EE_timer_x_start

Synopsis

`void EE_timer_x_start(void);`

Description

Start the timerx (x = 0,1,2,3)

2.3.12 EE_timer_x_stop

Synopsis

`void EE_timer_x_stop(void);`

Description

Stop the timerx (x = 0,1,2,3)

2.3.13 EE_timer_x_get

Synopsis

```
EE_UREG EE_timer_x_get(void);
```

Description

Return the value of the counter register of the timerx (x = 0,1,2,3)

2.3.14 Configuring a particular AVR microcontroller

Atmel produces various versions of the AVR microcontrollers, each one with different peripherals and memory sizes.

The AVR-Gcc specifies the various peripherals and memory areas by passing an appropriate command line parameter which selects the right linker script for the specified device. For that reason, the OIL file allows the specification of the microcontroller used, as in the following example:

```
...
MCU_DATA = AVR5 {
  MODEL = ATMEGA128;
};
...
```

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

- AVR5 devices:
  `ATMEGA128`,

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 = "125ns";};
```

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 AVR. The current timing reference implemented in the EDF kernel for AVR is based on the value of the 16 bit timer Timer1.

The 16 bit timer is obtained by using the timer1 available on the AVR. The clock used for the timers is the system clock, with a tick time equal to $\frac{1}{F_c}$. F_c is the frequency of the oscillator, and depends on the application configuration.

In the case of the Atmel ATmega128, the default frequency is 8 MHz. In that case, the OIL file should contain the following line:

```
KERNEL_TYPE = EDF {TICK_TIME = "125ns";};
```

and the `main()` function should have the following line before using any Erika Enterprise primitive:

```
  EE_timer1_init();
  EE_timer1_start();
```

3 Atmel STK500/1 Board

3.1 Introduction

This chapter describes the support done in Erika Enterprise for the Atmel STK500/1 Board (see Figure 3.1).

The STK500/1 is a low cost, efficient development board produced by Atmel with interfaces to JTAG, SPI, and RS-232 [2].

To configure the usage of the STK500/1 Board, the user has to specify an appropriate `BOARD_DATA`, as in the following example:

```
...
BOARD_DATA = ATMEL_STK50X {
   ...
}
...
```

The STK500/1 board supports a set of devices which are directly available and 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.

Please note that the current version of the board support only supports the AVR model ATMEGA128.

Please remember that you have to connect ISP6PIN plug of the STK500 with the SPROG plug of the STK501 through 6-wires cable.

Figure 3.1: The Atmel STK500/1 board running Erika Enterprise.

3.2 LEDs

The Atmel STK500 Board has a set of 8 LEDs attached to the PORTB 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 = ATMEL_STK50X {
  USELEDS = TRUE;
  ...
}
...
```

The following subsections will describe the functions available to control the Atmel STK500 LEDs.

3.2.1 EE_led_1_on

Synopsis

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

Description

The function turns on LED 1.

3.2.2 EE_led_1_off

Synopsis

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

Description

The function turns off LED 1.

3.2.3 EE_led_2_on

Synopsis

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

Description

The function turns on LED 2.

3.2.4 EE_led_2_off

Synopsis

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

Description

The function turns off LED 2.

3.2.5 EE_led_3_on

Synopsis

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

Description

The function turns on LED 3.

3.2.6 EE_led_3_off

Synopsis

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

Description

The function turns off LED 3.

3.2.7 EE_led_4_on

Synopsis

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

Description

The function turns on LED 4.

3.2.8 EE_led_4_off

Synopsis

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

Description

The function turns off LED 4.

3.2.9 EE_led_5_on

Synopsis

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

Description

The function turns on LED 5.

3.2.10 EE_led_5_off

Synopsis

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

Description

The function turns off LED 5.

3.2.11 EE_led_6_on

Synopsis

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

Description

The function turns on LED 6.

3.2.12 EE_led_6_off

Synopsis

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

Description

The function turns off LED 6.

3.2.13 EE_led_7_on

Synopsis

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

Description

The function turns on LED 7.

3.2.14 EE_led_7_off

Synopsis

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

Description

The function turns off LED 7.

3.2.15 EE_led_8_on

Synopsis

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

Description

The function turns on LED 8.

3.2.16 EE_led_8_off

Synopsis

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

Description

The function turns off LED 8.

4 Crossbow MIB5X0 Board

4.1 Introduction

This chapter describes the support done in Erika Enterprise for the Crossbow MIB5X0 Board. Please note that both boards mib510 and mib520 are supported, enabling the programming of the mica2 and micaz nodes (see Figure 4.1).

The Mib520 is a low cost, efficient development board produced by Crossbow with interfaces to JTAG and USB [3].

To configure the usage of both MIB510 Board and MIB520 board, the user has to specify an appropriate `BOARD_DATA`, as in the following example:

```
...
BOARD_DATA = XBOW_MIB5X0 {
  ...
}
...
```

4.2 LEDs

The MIB5X0 Board has a set of 3 LEDs attached to the PORTA 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 = XBOW_MIB5X0 {
  USELEDS = TRUE;
```

Figure 4.1: Crossbow MIB520 Board and MicaZ.

```
      ...
    }
    ...
```

The following subsections will describe the functions available to control the Crossbow MIB5X0 LEDs.

4.2.1 EE_led_1_on

Synopsis

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

Description

The function turns on LED 1.

4.2.2 EE_led_1_off

Synopsis

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

Description

The function turns off LED 1.

4.2.3 EE_led_2_on

Synopsis

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

Description

The function turns on LED 2.

4.2.4 EE_led_2_off

Synopsis

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

Description

The function turns off LED 2.

4.2.5 EE_led_3_on

Synopsis

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

Description

The function turns on LED 3.

4.2.6 EE_led_3_off

Synopsis

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

Description

The function turns off LED 3.

5 History

Version	Comment
1.0.0	Initial revision of this document.

Bibliography

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

[2] Atmel Corp. The atmel corp. home page. http://www.atmel.com/, 2007.

[3] CrosBow. The crossbow home page. http://www.xbow.com/, 2007.

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

Index