
TN8000.16 Technical Note

Coding with RIDE Quick Start

CONTENTS

TABLE OF CONTENTS

TN8000.16 Technical Note	1
Contents	2
Table of contents.....	2
Table of figures	2
Table of tables.....	2
1 Introduction	3
1.1 Purpose of the document	3
1.2 Requirements	3
1.3 About the given example.....	3
1.4 Document conventions.....	3
2 RIDE working directory structure.....	4
2.1 Short description of relation ships between tools	4
2.2 Short description of tools.....	4
3 Basic files for the XE88LCXX.....	6
3.1 Template files	6
3.2 Types.h file	6
3.3 XE88LCxxx.h file	6
3.4 Startup files	7
3.5 Interrupts handling files	8
3.6 datainit.s file	9
4 Setting-up the project	10
5 Starting your application coding	11
6 Compiling and a downloading the code in the SoC memory	13
7 Available application & TEchnical notes.....	15
8 Documentation provided with Raisonance RIDE	16

TABLE OF FIGURES

Figure 1 Working directory structure	4
Figure 2 Tools relationship	4
Figure 2 Changing startup file	7
Figure 3 Changing linker script.....	7
Figure 4 Renaming the project	10
Figure 5 Renaming the application file	10
Figure 6 Changing the target type	10
Figure 7 Code example	11
Figure 8 Compile the code.....	13
Figure 9 Setting the XELoader	13
Figure 10 Launching debug.....	14
Figure 11 Download window	14

TABLE OF TABLES

Table 1 Types definition.....	6
Table 2 List of current application & technical notes	15
Table 3 List of current application & technical notes (cont'd).....	16

1 INTRODUCTION

1.1 PURPOSE OF THE DOCUMENT

This document describes how to start programming an XE8000 family SoC (System on Chip) using RIDE.

In this technical note you will find:

- An example of the project structure to be used.
- A short description of the function of each file involved in the project.
- Where to find startup files and how to customize them.
- A list of the available technical notes and application note with a short description of each.
- A small code example from “unzipping the project” to download onto the XE8000 SoC.

1.2 REQUIREMENTS

To complete the points described in this document you will need:

- A PC with RIDE installed.
- The zip file template.zip
- This document.

1.3 ABOUT THE GIVEN EXAMPLE

Example files: The files can be found on the website under www.xemics.com/support download files section, and then click on “[XE8000 - Data Acquisition](#)”.

The zip file available with this technical note is an example of how you can set up your project. It provides templates of the basic files that are in nearly all projects.

It is not mandatory to follow the given method exactly; this is only an example for users in order to start to use the XE88XX quickly.

This code is designed to ease the start of a project with RIDE, and is user friendly oriented rather than memory efficient.

To save memory space you can reduce the « InitUART() » function. This function is a switch made to take into account all the possible configurations of the UART, once you have decided on one configuration you can suppress all the other cases.

1.4 DOCUMENT CONVENTIONS

This document refers to the most up to date version of RIDE, it is possible that the tool's name appears to be different on your current RIDE installation (e.g. LXE instead of LC816). If this is the case it is strongly advised that you update your RIDE tool following the procedure described in the technical note TN8000.19, available in the Semtech web site.

All the reference to the menus are written as follows: ***menu|menu_level1| menu_level2 function***

2 RIDE WORKING DIRECTORY STRUCTURE

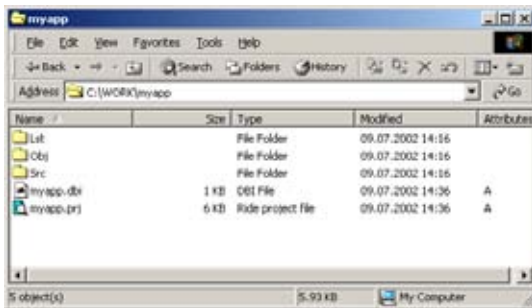


Figure 1 Working directory structure

There are three directories in your working directory

- **Lst** All the *.lst files generated by the tool are placed here. Listing files contain all the assembler code generated, memory requirements of the module, a symbol table and how the tool was set up.
- **Obj** All the *.o files generated by the tool are placed here. Object file are generated by the C compiler and the Assemble tool, they are used by the linker to generate the final object file.
- **Src** All your source files *.c; *.h *.s must be placed here.

2.1 SHORT DESCRIPTION OF RELATION SHIPS BETWEEN TOOLS

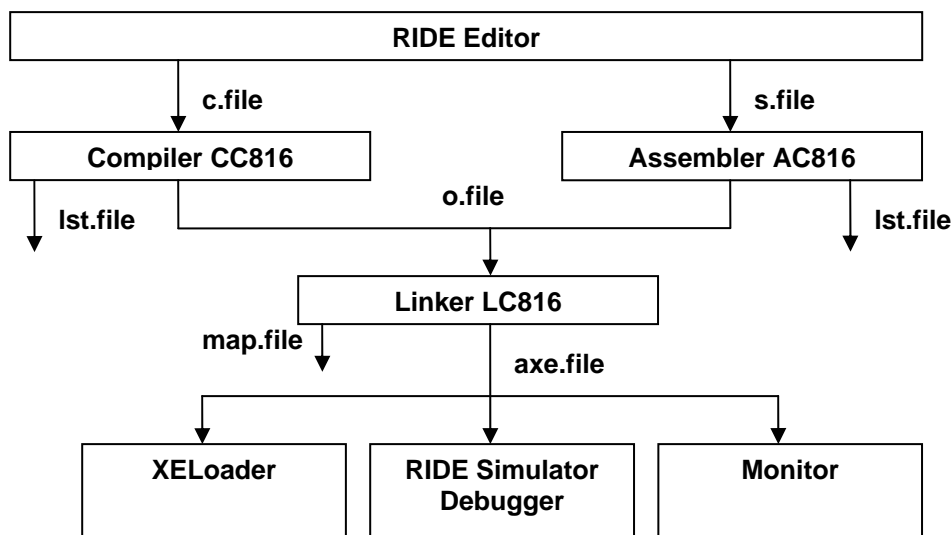


Figure 2 Tools relationship

Note In the diagram above, the library manager tool is omitted for readability, please consult Raisonance Getting Started C816 document (*[RIDE menu bar help/PDF/C816 Tools](#)*)

2.2 SHORT DESCRIPTION OF TOOLS

- **XELoader** Used to physically download the code in the SoC memory, this is the link between RIDE and the hardware. For more information see the Prostart II users guide.

- **RIDE Simulator** Used to simulate the code written in RIDE, the simulator allows to view memory dumps, and most of the peripherals. For more information see the Raisonance Getting Started C816 document.
- **Monitor** Used to verify the code in real hardware conditions, the monitor allows setting of breakpoints and memory dumps. Note that the XELoader is also used by the monitor tool. For more information see the Raisonance Getting Started C816 document.

3 BASIC FILES FOR THE XE88LCXX

3.1 TEMPLATE FILES

- **Main.c** This file contains the starting point of your application, here you can manage the initialization of the SoC, and call all the functions of your application.
- **Initialisation.c & .h** This file contains all the initialization functions that may be called by the main function (manually), the .h file contains the prototypes of these functions.
Note: InitMicro() function is for the ports, counters and RC/UART initialization.
- **DFLLDriver.c & .h** This file contains the DFLL function, the function that allows the user to calibrate the RC oscillator using the XTAL with a precision of 2% (see TN8000.09 on our [website](#) for more information). There is no need to modify the file, it is called by the UART initialization.
- **Globals.h** In this file you may include all the other header files that you want to be visible in the project. In this file you may declare global variables and definitions for code generation.

Note : These files are included in the **template.zip** file

3.2 TYPES.H FILE

This file contains a redefinition of the standard types; the main reason for redefining these is to ease the understanding of the types' sizes and also ease the readability.

The types redefined are the described in the table below:

Standard type name	Defined type name	Comment
bool (unsigned char)	false/true – 0/1	8 bit quantity
unsigned char	_U8	8 bit quantity
char	_S8	8 bit signed quantity
unsigned short	_U16	16 bit quantity
short	_S16	16 bit signed quantity
unsigned long	_U32	32 bit quantity
long	_S32	32 bit signed quantity
float	_F24	24 bit float quantity
double	_F32	32 bit float quantity

Table 1 Types definition

Note : The file location is **C:\RIDE\GNU\C816\INCLUDE\Types.h**

3.3 XE88LCXXX.H FILE

This file contains all the registers definitions of the chosen target.

Note¹: This file is loaded automatically using the preprocessing code in “globals.h” file under « Soft generation control target type ».

Note² : The file location is **C:\RIDE\GNU\C816\INCLUDE\XE88LCxxx.h**

STARTUP FILES

As for every SoC, a set of files is necessary to handle the registers definition, and the initialization at startup. By default these files are not visible to user but they can be seen and modified by the user. See the next lines for more information on each of them.

IMPORTANT NOTE: These file are already optimized and dedicated to the different XE8000 targets. Advanced users can modify these files with the support of the datasheets and GNU tools documentation. This chapter is designed to demonstrate how the startup management is done rather than to explain how to modify this set of files.

crt0.s This file contains the startup code, it handles the reset, the interrupts and also the variables initialization.

The standard file can be found under C:\RIDE\TEMPLATES\COMMON

If you want to customize it to your needs, you'll need to copy it in your working directory and add it to the project, then you need to specify in the linker options that you want to use your own startup file **menu bar options/project/LC816** and uncheck the "use default" check box as shown in picture below and change the path to your own crt0.o file using the [...] browse button.



Figure 3 Changing startup file

NOTE: You need to translate the crt0.s file at least once in order to create the crt0.o file (to translate: right click on the file name then select "translate")

crt0.ld This file is the linker script, there is one ".ld" file dedicated by target, it defines memory areas, physical addresses of the registers and predefined memory sections. The linker scripts can be found under C:\RIDE\TEMPLATES\

If you want to customize it to your needs, you'll need to copy it in your working directory and add it to the project, then you need to specify in the linker options that you want to use your own linker file **menu bar options/project/LC816**, uncheck the "use default" check box as shown in picture below and change the path to your own crt0.ld file using the [...] browse button.



Figure 4 Changing linker script

3.4 INTERRUPTS HANDLING FILES

The following files manage the interrupt mechanism, by default they are hidden to the user and the handler prototypes are defined in a library.

All the interrupt process is managed through these files and functions; there is no need to clear bits or check bits.

IMPORTANT NOTE: The startup files are already optimized and dedicated to the different XE8000 targets. Advanced users can modify these files with the support of the datasheets and GNU tools documentation. This following chapter is more to allow users to see how the interrupt management is done rather than to encourage them to modify this set of files.

IRQComon.s This file contains the code that saves & restores the context for any interrupt of the SoC, it also determines which interrupt routine to call.

The file can be found under C:\RIDE\TEMPLATES\

IRQSave0.s This file contains the code that calls the interrupt handlers for high priority level interrupts.

The file can be found under C:\RIDE\TEMPLATES\

IRQSave1.s This file contains the code that calls the interrupts handlers for mid priority level interrupts.

The file can be found under C:\RIDE\TEMPLATES\

IRQSave2.s This file contains the code that calls the interrupts handlers for low priority level interrupts.

The file can be found under C:\RIDE\TEMPLATES\

If you want to customize these files to your needs, you will need to copy them into your working directory and add them to the project.

The files that are in your project will replace the ones that are placed in the templates directory.

NOTE: The “Default IRQ handler” check box (*options/project/LC816*) includes the library in the project and predefines all the headers for IRQ handlers.

The IRQ handlers are always called “Handle_Irq_<irq name>” Where <irq name> is the name of the IRQ as specified in the relevant datasheet. If you plan to rename these handlers you may uncheck this checkbox.

When “Default IRQ handler” is checked the unused interrupts of your project do not need to be declared, they will already have been declared in the library. When you want to use an interrupt handler, just add the corresponding declaration in your code. All the interrupt handlers can be found under C:\RIDE\TEMPLATES\COMMON

3.5 DATAINIT.S FILE

The C816 core does not allow data to be moved from (nor to) the program memory. This is a limitation when considering the initialization of the global variables.

In RIDE there is a way to hard-code the initialization, by inserting some “move” commands between the startup and the main() function.

This is done in RIDE by selecting the “generate data init” option in **Option /Project/Linker**. The drawback of using this option is that the link phase is longer (about 2 to 3 times) and that it generates two instructions for every byte to be initialized. So if you can avoid initializing global variables, do. (Note: variables not initialized are set to zero by the default startup...)

If you select this option but do not use the default startup files, please note that you should call the Datalnit function (“calls Datalnit”) **before** you use any initialized global variable. A good place to do that is just before calling the main function, or at its beginning. See the default crt0.s files for a working example. On the other hand, if you de-select this option but still want to use the default startup file, you must provide a Datalnit function or the link will not succeed. It will be called before the main function, but maybe also before some other initializations are done (like the heap pointer for dynamic memory allocation, so you should not use malloc() in this function).

You can perform your own initializations here, but remain aware of the fact that the startup is not completed when it is called so be careful.

This function doesn't have to do anything, “Datalnit: rets” would be a valid implementation, but then the variables would not be initialized.

How it works:

When you select the “generate data init” option in **Options/Project/Linker**, the link phase will be modified this way: A first link (pre-link) will be done.

Then, the tools ‘objcopy’ and ‘srec2rom’ will be run on the resulting binary file and they will produce an assembler file containing a “Datalnit” function that does the necessary initializations for the application. This file will then be assembled and the link will be done again with this one instead of the one included for the pre-link.

If you don't modify the initialized data in your program from one link to the other, you may not want to generate the data every time you link. In this case, link once using the option. Then, unselect the option and include the generated assembler file in your project's directory named like this (assuming your application's name is “app.axe”): “app.axe.Datalnit.s”. Note: Optimization is possible with careful analysis.

4 SETTING-UP THE PROJECT

1. Unzip the file to your working directory
2. Rename the directory from « template » to your project name.
3. Rename the template.prj file to your project name

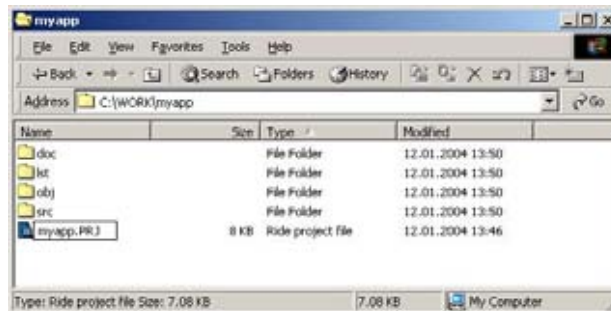


Figure 5 Renaming the project

4. Then you can launch RIDE and open the created project.
5. In RIDE you may change the application file to yours (right click on the name).

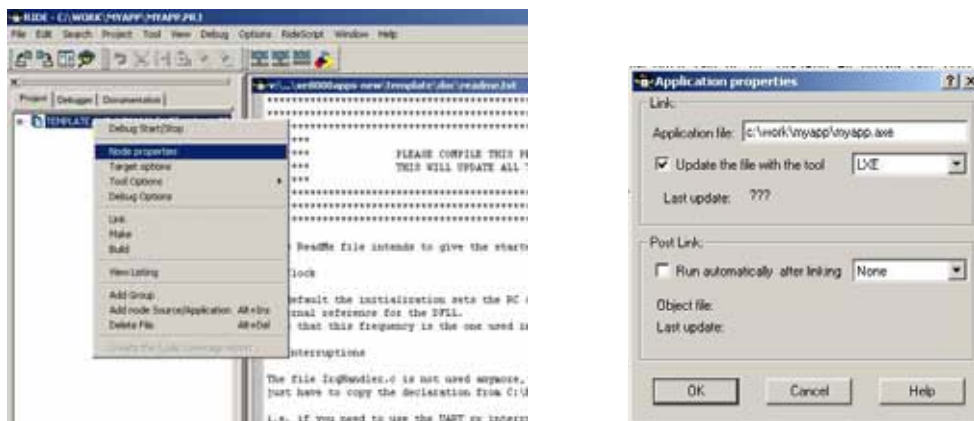


Figure 6 Renaming the application file

6. When completed you need to set the target

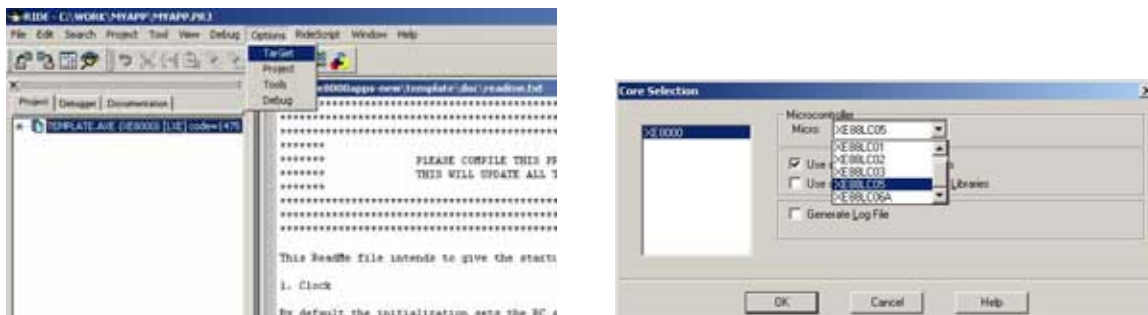


Figure 7 Changing the target type

7. Build the project (shift+F9)
8. Now you are ready to start coding!

5 STARTING YOUR APPLICATION CODING

Now you can start coding your application. Following is an example of coding for the main application:

```

/*****
** File      : main.c
**
**
** Version   :
**
** Written by :
**
** Date      : XX-XX-XXXX
**
** Project   : -
**
** Changes   :
** Description : Main program
**
***/
#include "Globals.h"

/*****
** Global variables declaration
**
***/

/*****
** main : Main program function
**
** In    : -
** Out   : -
**
***/
int main (void){
    /* Ports with Std config PB & PC or PD as outputs */
    /* RC running @ 1 MHz +/- 2% (using DFLL) Xtal On */
    InitMicro();
    /* Initializes the monitor when used options/project LC816
    /* "monitor library" checked & "#define usemonitor 1" in globals.h file*/
    Monitor_Init();
    /* First break point when monitor used (mandatory) */
    _Monitor_SoftBreak;

    /* Enables the 1Hz interrupt */
    RegIrqEnMid = 0x08;

    /* Infinite loop */
    while(1){
        /* Waits for 1Hz interrupt */
        asm ("halt");
    }

    return 0;
}

void Handle_Irq_1Hz (void){
    /* performs an XOR on PB 3 output the LED labelled PB3 should blink */
    /* every second on every XE8000EVxxx */
    RegPBOut ^= 0x08;
} //End Handle_Irq_1Hz

```

Figure 8 Code example

In the example above, we use the 1Hz interrupt to toggle the signal on PB3 output, the LED of the XE8000Evxx labeled PB3 should blink every second.

Note¹ : The code in **BOLD** should be added by the user.

Note² : All the peripherals basic configuration can be changed in the initialization.c file.

6 COMPILING AND A DOWLOADING THE CODE IN THE SOC MEMORY

Now the code is written and needs to be tested on real hardware.

To compile the code press the “make all” button (see the picture below)



Figure 9 Compile the code

To configure the downloader (XEloader) see the picture below:

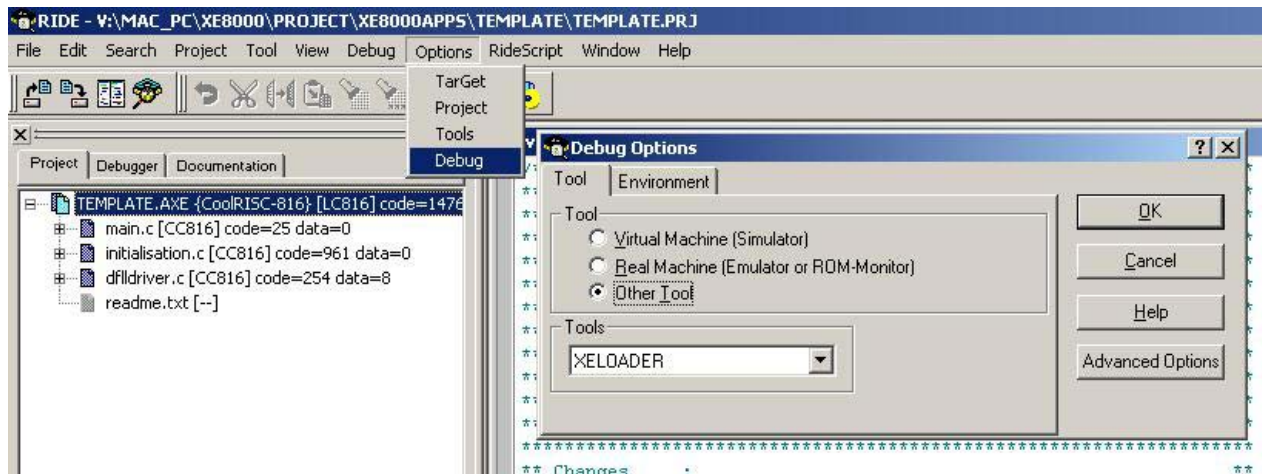


Figure 10 Setting the XEloader

Click on “Advanced Options” to set the COM port that is connected on the XE8000MP.

Then connect the XE8000MP as described in the “XE8000 Prostart II Users Guide” Chapter 2.6 and launch the programmer using either [ctrl + d], the debug icon or a double click on the project tree root (see following picture)

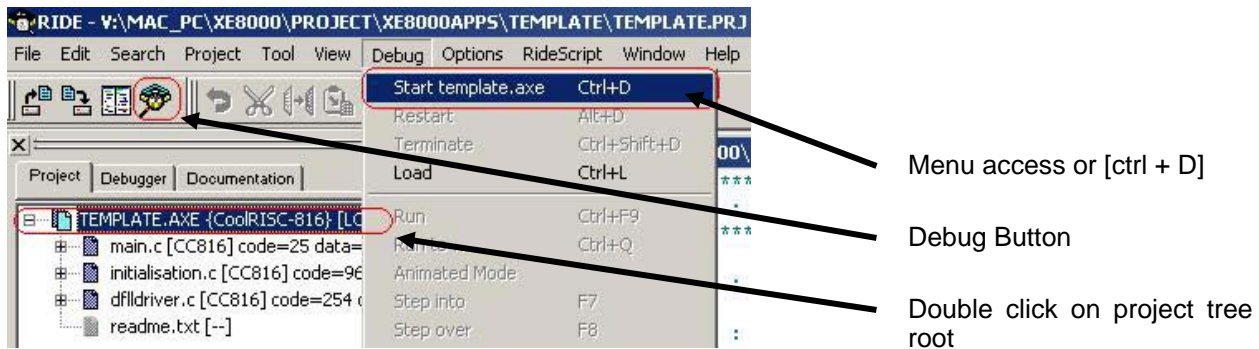


Figure 11 Launching debug

Once the download is launched the following pop up window should appear:

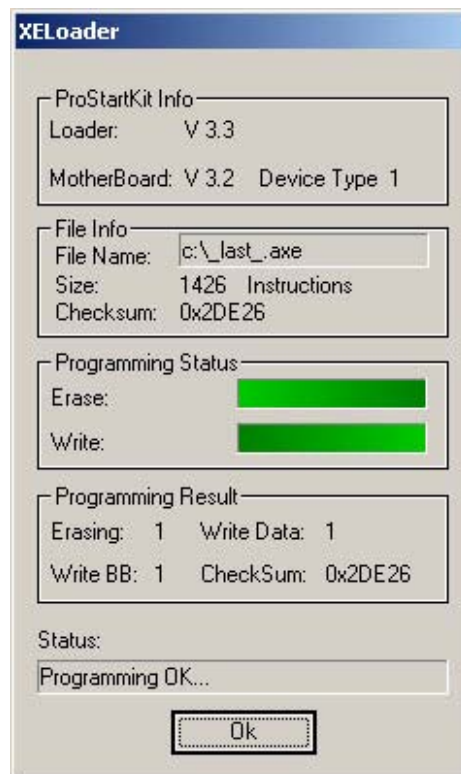


Figure 12 Download window

And the LED labeled PB3 on the XE8000EVxxx should now be blinking every second.

Congratulations! You have just finished your first program using XE8000 tools.

Remember that there are already a lot of examples and applications on our web site:
www.xemics.com/support/XE8000 section download files

See next page for a short description of each application & technical notes available at this time

7 AVAILABLE APPLICATION & TECHNICAL NOTES

Short list of applications & technical notes available

	Number	Name	Description
Basics	TN8000.16	Coding with RIDE quick start	[This technical note] Shows the basic files used in most RIDE projects, and has a simple example project that makes a LED blink on XE8000EVxxx [source code available]
	TN8000.09	D.F.L.L. Digital Frequency Locked Loop	Shows how to get an accurate RC frequency using external 32kHz XTAL [source code available]
	TN8000.04	CoolRISC 816 instruction codes and examples	Short description of the C816 core and instructions.
	TN8000.06	interleaved interrupts handling	Technical note explaining the C816 core interrupt mechanism and how to handle nested interrupts if needed.
	TN8000.10	Guidelines on utilization troubleshooting and buglist	All known bugs or troubleshooting are listed in this technical note their workaround are also described
Using internal peripherals / complete applications	AN8000.05	Using the Zooming ADC TM of the Sensing Machines	Description of the ZoomingADC and its driver files, provided with a sample code that sends ZoomingADC output to UART [source code available]
	AN8000.08	Guidelines on how to build a barometer around the XE88LC05A and a piezoresistive pressure sensor	Description on how to set up an application using a piezoresistive sensor using the full ZoomingADC versatility
	AN8000.12	Wireless Compass	Description on how to consider the use of magneto resistive sensors when wanting to set up a compass application.
	AN8000.11	A general purpose air quality monitor system with RF link	Description on how to set up chemical sensors on air quality monitor applications.
	AN8000.14	XEMICS Sensor Demo	Complete application of the "Sensor Demo" This technical note gives all the information to rebuild this demonstrator, Source codes and gerber files included [database available]
	TN8000.17	BitJockey™ Basic Driver	Shows how to take advantage of the BitJockey™ possibilities using a dedicated driver. [source code available]
Driving ext. peripherals	AN8000.13	Driving LCD Displays using XE8000 standard I/O ports	Describes how to develop a low cost LCD driver using standard XE8000 I/O's [source code available]
	TN8000.13	How to interface an I2C EEPROM with the XE8000 Series	Software driver for I2C protocol using std XE8000 I/O's. [source code available]
	TN8000.14	How to interface an SPI EEPROM with the XE8000 Series	Software driver for SPI protocol using std XE8000 I/O's. [source code available]
Driving Semtech products	TN8000.22	Using an XE3005 CODEC with an XE88LC07A	Software driver of the XE3005 CODEC combined with the use of a XE88CL07A. This tech note provides an optimized driver for XE3005 configuration and audio data communication. [source code available]
	TN8000.18	XE8000 driving XE1200 transceivers standard API definitions	Provides a comprehensive set of functions to drive each Semtech RF transceiver. [source code available]

Table 2 List of current application & technical notes

Mech.	TN8000.01	Delivery Formats for IC Die	Describes all the die formats available for XE8000 SoCs
	TN8000.07	XE8000 packaging information	Gives a mechanical description of every package of the XE8000 family
Tools	Number	Name	Description
	TN8000.19.	XE8000MP and RIDE Firmware - Software Update Process	Describes how to upgrade your development tools
	TN8000.02	XE88LC01(A)/05(A) MTP programming	Describes the algorithms to program the XE88LC01A / 05A flash arrays, this tech note is intended to programmers manufacturers
	TN8000.20	XE88LC02/06A MTP programming	Describes the algorithms to program the XE88LC06A / 07A & XE88LC02 flash arrays, this tech note is intended to programmers manufacturers
	TN8000.23	Converting a binary file to a .rom file for industrial programmers	This tech note shows how to convert files for industrial programmer using GNU tools.

Table 3 List of current application & technical notes (cont'd)

NOTE: Please also consult the users guides & datasheet that may contain the answers to most of the basic questions

8 DOCUMENTATION PROVIDED WITH RAISONANCE RIDE

We also encourage you to consult the documentation provided by Raisonance RIDE under [help/PDF/C816-Tools](#)
Here you will find:

Users guides:

- Getting Started C816 This file explains how to use ROM monitor and simulator; it provides also a good overview of the tools interactions in RIDE.

Gnu tools reference

- AC816 C816 assembler reference guide
- CC816 C816 c compiler reference guide
- LC816 C816 linker reference guide

RIDE tools reference

- CodeCompressorC816 Code compressor users guide
- RIDE RIDE IDE users guide
- BN728_XE_new Add-on to the users guide new features since release 728

SEMTECH specific documentation

- XE8000 Is a link sheet that contains reference to the following documents
 - C816 user's guide C816 core users guide, full processor documentation
 - Preprocessor (cpp) GNU tools preprocessor documentation.
 - C816 C compiler (gcc) GNU tools reference docs (same as described above)
 - C816 Assemble (as) GNU tools reference docs (same as described above)
 - C816 linker (ld) GNU tools reference docs (same as described above)
 - Make tool (make) Obsolete (no more included in toolpack)
 - C816 C library (libc) C library reference documentation
 - C816 Math library (libm) Math library reference documentation
 - XE8000 driver libraries (XE800drv) Obsolete (no more used)
 - Binary utility documentation (binutil) GNU binary utilities reference documentation
 - Frequently asked questions (faq) FAQ about how to minimize Program memory and data memory spaces and also how to interact between C and asm.

© Semtech 2005

All rights reserved. Reproduction in whole or in part is prohibited without the prior written consent of the copyright owner. The information presented in this document does not form part of any quotation or contract, is believed to be accurate and reliable and may be changed without notice. No liability will be accepted by the publisher for any consequence of its use. Publication thereof does not convey nor imply any license under patent or other industrial or intellectual property rights. Semtech. assumes no responsibility or liability whatsoever for any failure or unexpected operation resulting from misuse, neglect improper installation, repair or improper handling or unusual physical or electrical stress including, but not limited to, exposure to parameters beyond the specified maximum ratings or operation outside the specified range.

SEMTECH PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF SEMTECH PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE UNDERTAKEN SOLELY AT THE CUSTOMER'S OWN RISK. Should a customer purchase or use Semtech products for any such unauthorized application, the customer shall indemnify and hold Semtech and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs damages and attorney fees which could arise.

Contact Information

Semtech Corporation
Wireless and Sensing Products Division
200 Flynn Road, Camarillo, CA 93012
Phone (805) 498-2111 Fax : (805) 498-3804