Freescale Semiconductor
Application Note

Document Number: AN4752
Rev. 0, 5/2013

FreeMASTER Usage

Serial driver implementation

Radomir Kozub
Roznov, Czech Republic

The FreeMASTER serial driver is a piece of code that enables an embedded application to communicate with the FreeMASTER PC application. This application note shows how to add the FreeMASTER serial driver to your embedded code.

Contents

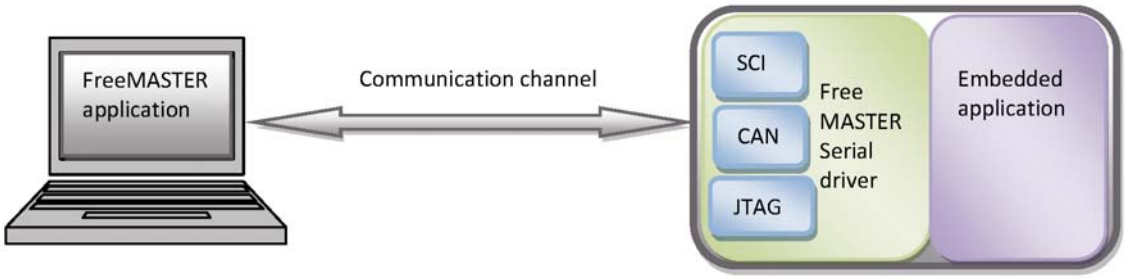
1 What are the FreeMASTER serial driver and FreeMASTER PC application?

FreeMASTER is a PC-based development tool serving as a real-time monitor, visualization tool, and graphical control panel of embedded applications based on Freescale Semiconductor processing units.

The FreeMASTER PC application repetitively sends a request for the immediate values of chosen variables used in the embedded application (the PC application acts as the master in this peer-to-peer communication).

The embedded application replies with the actual value of the variable (the embedded application acts as a slave in the communication). The FreeMASTER PC Application visualizes the variable value. The piece of embedded code that takes care of responding to a request is called the FreeMASTER serial driver. This driver carries out protocol parsing, prepares responses, and handles the communication periphery. The serial driver covers the UART SCI and CAN communication for all supported devices, and the EOnCE/JTAG communication for the 56F8xxx family of hybrid microcontrollers.

This document describes how to add the FreeMASTER serial driver software to your embedded application and configure it.

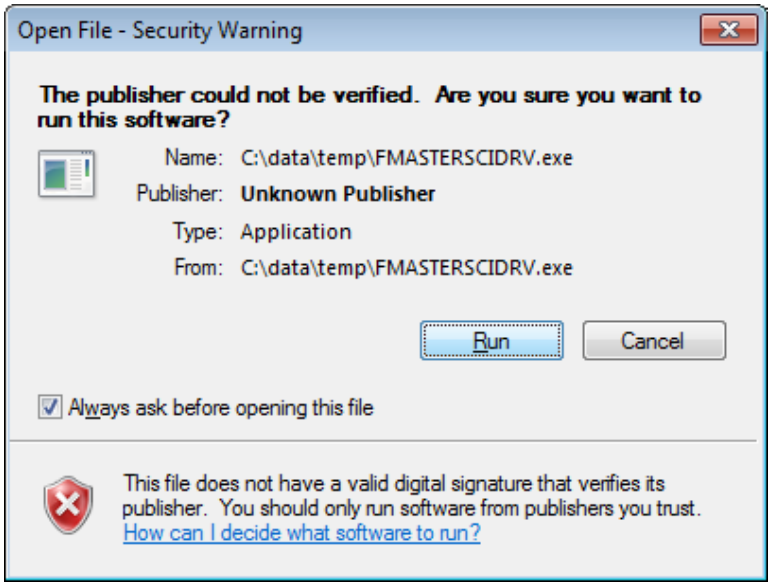


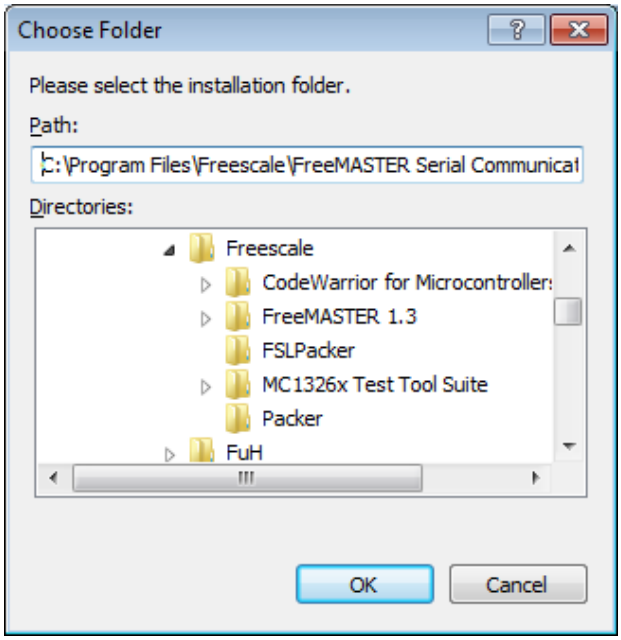
Go to http://www.freescale.com/Freemaster to get the latest version of the FreeMASTER PC application, as well as the FreeMASTER serial drivers.

All details about the communication protocol and the FreeMASTER PC application may be found in the FreeMASTER installation folder in the documents mcbcom.pdf and pcm_um.pdf. The default installation path is preset to c:\Program Files\Freescale\FreeMASTER 1.3\

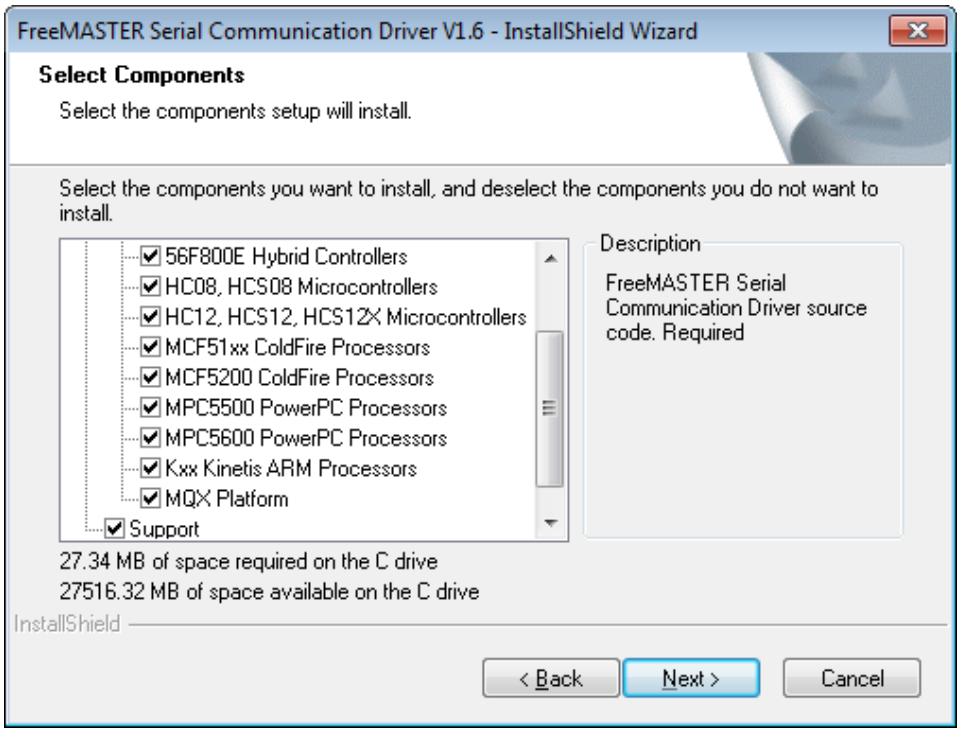
1.1 I want it — where do I get the FreeMASTER serial driver?

Go to the http://www.freescale.com/Freemaster download section, download the FMASTERSCIDRV.exe file and run the installation. Run through the welcome page and license agreement, then choose the path on where to install the serial driver files. The installer will unpack all the selected files — the driver source files, examples, and documentation — into the chosen directory.





You may either choose only those drivers for the platform you are planning to use (e.g. Kxx Kinetis ARM Processors) or choose a complete installation. Then, go through the installation wizard and finish it.



1.2 What can I find on the hard drive?

On the installation path is the ..\FreeMASTER Serial Communication V1.6 directory, which contains the following subfolders:

- doc — It's worth reading the document that outlines all the important details on the serial driver software.
- examples — The quickest way to run your first application with FreeMASTER is to open and modify a bareboard example application. Examples are already prepared for all the supported platforms (processors) for CodeWarrior and IAR development tools. For the Kinetis ARM processors, examples cover the K40 KWICSTICK and the K60N512 Tower board.
- src_common — This directory contains the common driver source files shared around all the supported platforms. All the *.c and *.h files in the directory should be added to your project, compiled and linked together with your application.
- src_platforms — This folder contains subfolders with platform-specific parts of the serial driver (Kxx, 56F8xxx, HC08, HC12, MPC55xx, etc.). Your application needs only those files in the platform-specific subfolder. For the Kinetis ARM processor, you must add only the *.c and *.h files from the Kxx subfolder.
- support — Contains those files needed for virtual serial com port usage.

1.3 What features does the serial driver offer?

The FreeMASTER serial driver implements all the features necessary to establish a communication between your embedded application and the FreeMASTER PC application. Communication may use one of the following hardware layers: SCI, EOnCE/JTAG (56F8xxx Hybrid DSC), CAN, or Packet-Driven BDM.

The most important serial driver functions are:

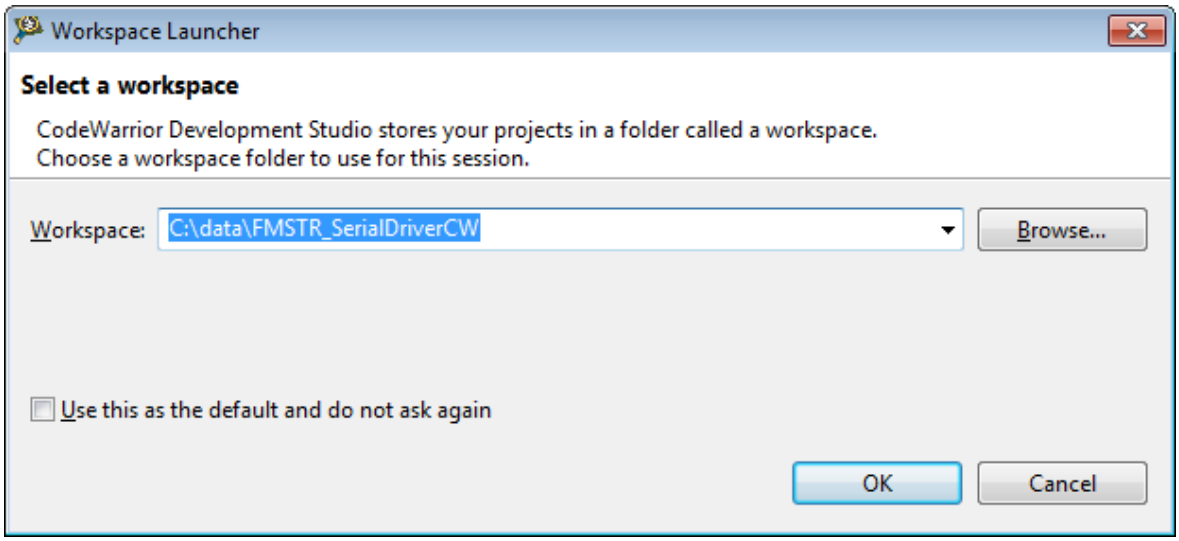
- Read/Write — Access to any memory location to read / modify an arbitrary variable or register.
- Oscilloscope access — Optimized real time reading of up to eight variables in one shot (all requested variables are read in a very short time), used for the graph visualization tool termed Scope in the FreeMASTER PC Application.
- Recorder — If observed variables are changing fast, the bandwidth limited serial line can't transfer the variable real time values. The recorder feature allows you to store the observed variables' real time data to the buffer in the embedded application and to transfer the buffer to the FreeMASTER PC application when the trigger event occurs. The transferred data is then visualized by the Recorder component of the FreeMASTER PC application. The buffer length is limited by the amount of available memory in the processor, or to 64KB.
- Application commands — High-level message delivery from the PC to the application.
- Target-side Addressing (TSA) — With this feature, you are able to describe the variables and structure data types directly in the application source code and make this information available for the FreeMASTER application tool. The tool may then use this information instead of reading it from the application's ELF/Dwarf executable file.

2 How do I create my first application with FreeMASTER in CodeWarrior 10.3?

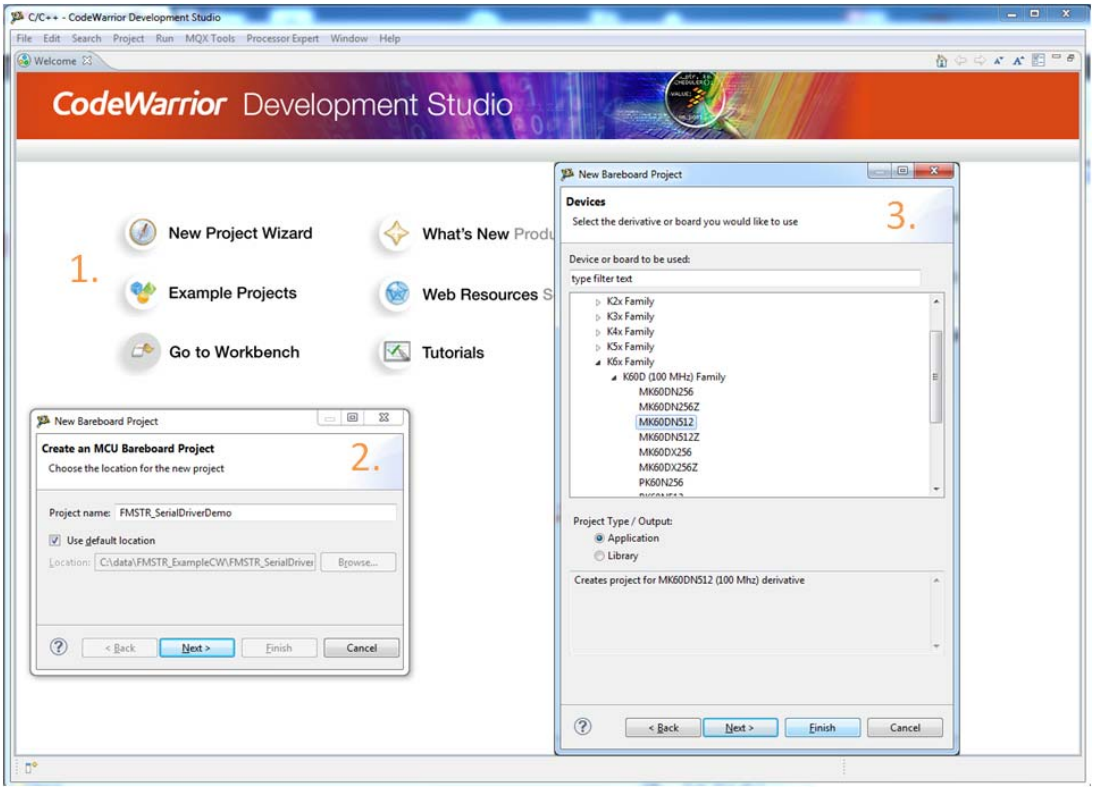
2.1 How to create an empty bareboard project stationery using CodeWarrior 10.3

We can use the create project wizard to create stationery containing all the necessary files to load and flash an empty project. The stationery contains processor header files, a linker file, start-up code, C initialization code, device initialization, and an empty main.

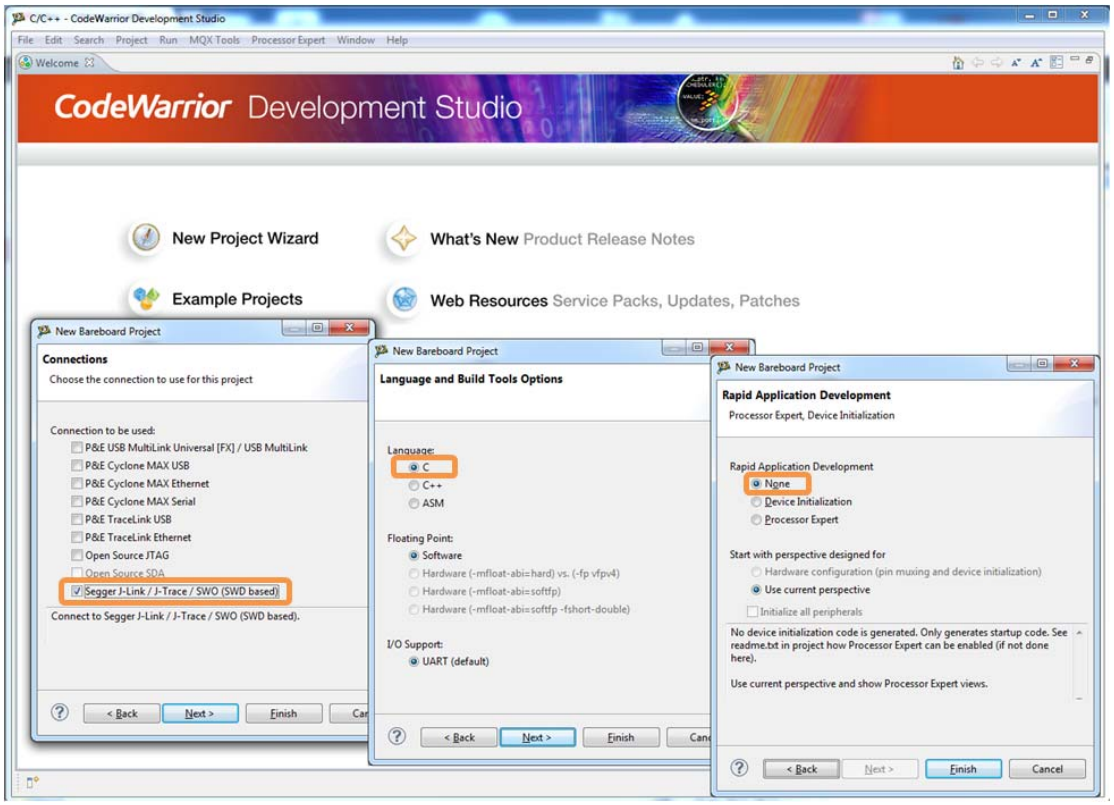
Select the folder where the project will be created.



Then choose the New Project Wizard, choose a project name, and pick the device to be used. This example uses the MK60DN512 processor on the TWR-K60D100M tower board:

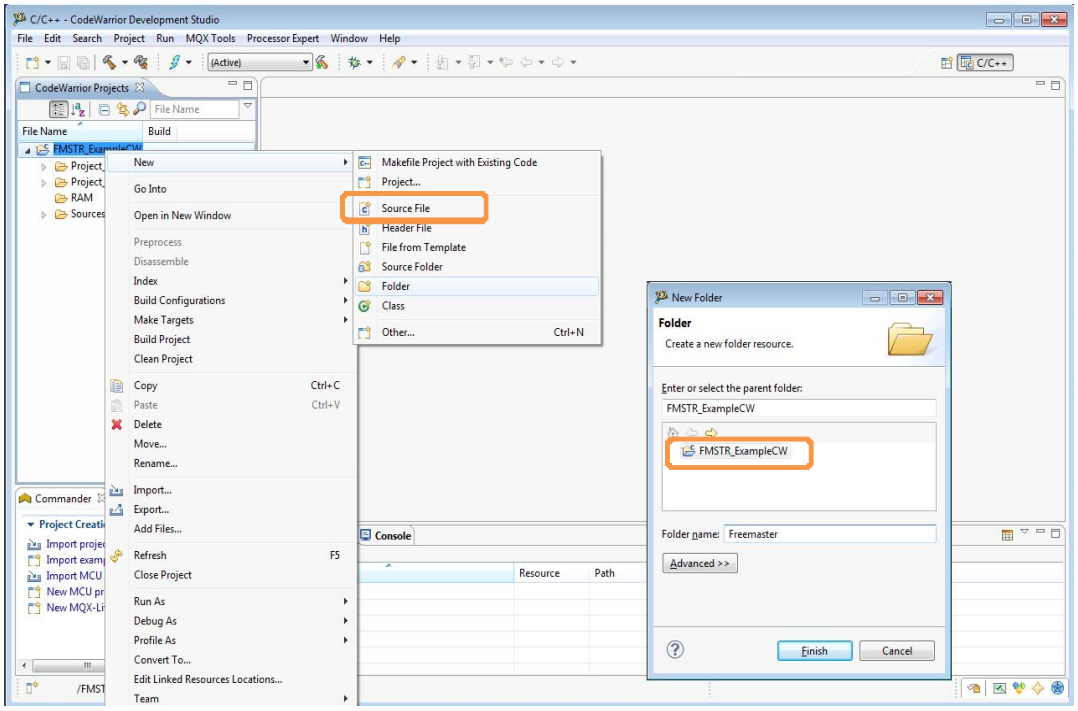


Then check the chosen connection board. This example doesn't use the Processor Expert Rapid Development tool, so select None in the Rapid Application Development window. The final step is to click on Finish and the empty project is created.

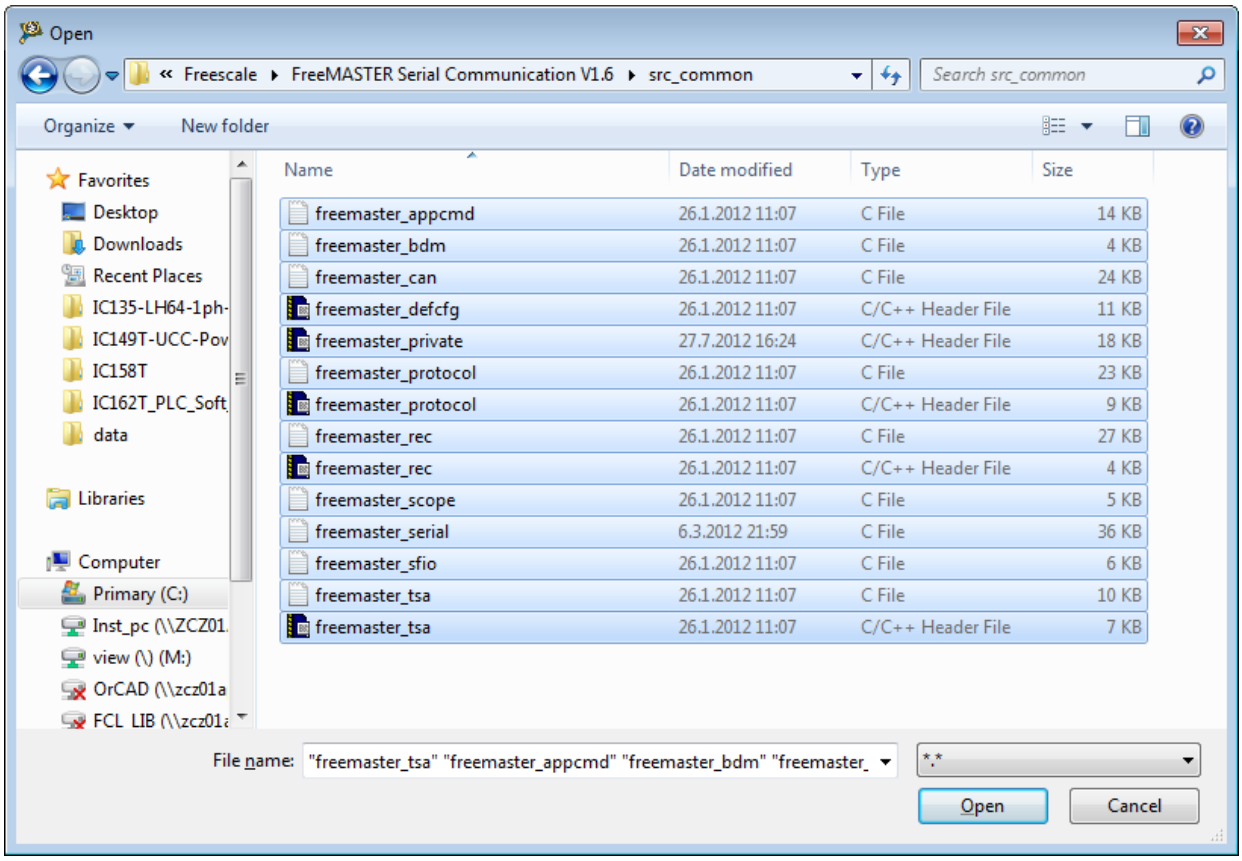


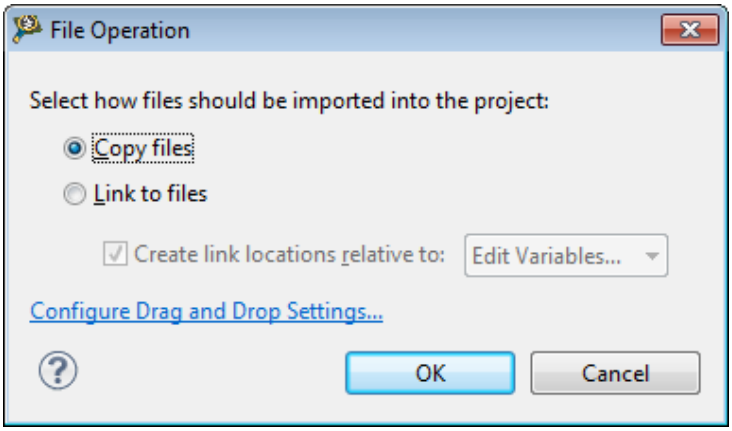
2.2 How to add the FreeMASTER communication driver files to the project

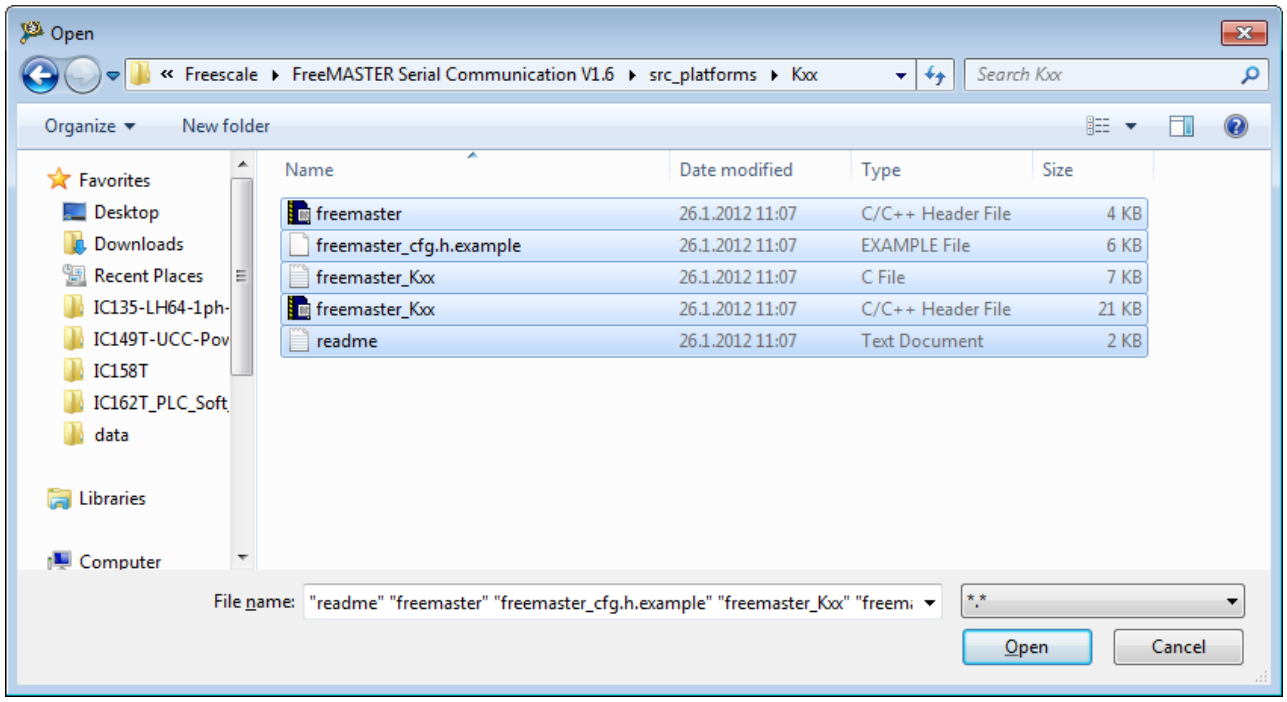
In the Project Explorer window, right mouse-click to FMSTR_SerialDriver project, select New -> Folder, and create a new folder Freemaster.



Right mouse-click to the Freemaster folder, choose Add files, and add all the files from the src_common directory along with the platform dependent files from the src_platform directory. In the Kinetis ARM processor case, we add files from the Kxx subdirectory.


Open
« Freescale ▸ FreeMASTER Serial Communication V1.6 ▸ src_common
Search src_common
Organize
New folder
Favorites
Desktop
Downloads
Recent Places
IC135-LH64-1ph-
IC149T-UCC-Pov
IC158T
IC162T_PLC_Soft
data
Libraries
Computer
Primary (C:)
Inst_pc (\\ZCZ01
view (\) (M:)
OrCAD (\\zcz01a
FCL_LIB (\\zcz01
Name
Date modified
Type
Size
freemaster_appcmd 26.1.2012 11:07 C File 14 KB
freemaster_bdm 26.1.2012 11:07 C File 4 KB
freemaster_can 26.1.2012 11:07 C File 24 KB
freemaster_defcfg 26.1.2012 11:07 C/C++ Header File 11 KB
freemaster_private 27.7.2012 16:24 C/C++ Header File 18 KB
freemaster_protocol 26.1.2012 11:07 C File 23 KB
freemaster_protocol 26.1.2012 11:07 C/C++ Header File 9 KB
freemaster_rec 26.1.2012 11:07 C File 27 KB
freemaster_rec 26.1.2012 11:07 C/C++ Header File 4 KB
freemaster_scope 26.1.2012 11:07 C File 5 KB
freemaster_serial 6.3.2012 21:59 C File 36 KB
freemaster_sfio 26.1.2012 11:07 C File 6 KB
freemaster_tsa 26.1.2012 11:07 C File 10 KB
freemaster_tsa 26.1.2012 11:07 C/C++ Header File 7 KB
File name: "freemaster_tsa" "freemaster_appcmd" "freemaster_bdm" "freemaster_
.
Open
Cancel


File Operation
Select how files should be imported into the project:
Copy files
Link to files
Create link locations relative to:
Edit Variables...
Configure Drag and Drop Settings...
OK
Cancel



Choose the Copy Files into the project option. The previous operation will also create a Freemaster folder in your workspace folder and copy the FreeMASTER serial driver files into the folder.

2.3 How to configure the FreeMASTER serial driver

All of the necessary FreeMASTER configuration is done in the freemaster_cfg.h file. You must do the configuration here. In the following text is a shortened list of the settings and a description.

As the first step before we start configuring the driver, we have to rename the freemaster_cfg.h.example file in the Project Explorer to freemaster_cfg.h and then open it. This file contains all the macro definitions available for the FreeMASTER configuration.

```
/*****************************************************************************
* Select interrupt or poll-driven serial communication
*****************************************************************************/

#define FMSTR_LONG_INTR         0    /* complete message processing in interrupt */
#define FMSTR_SHORT_INTR        1    /* only SCI FIFO - queuing done in interrupt */
#define FMSTR_POLL_DRIVEN       0    /* no interrupt needed, polling only */
```

Exactly one of the three macros must be defined non-zero. The others must be defined zero or left undefined. The non-zero-defined constant selects the interrupt mode of the driver.

```
FMSTR_LONG_INTR = 1
```

Serial communication and the FreeMASTER protocol decoding and execution is done in the FMSTR_Isr() interrupt service routine. As the protocol execution may be a lengthy task, it is recommended to use this

mode only if the interrupt prioritization scheme is possible in the application, and if the FreeMASTER interrupt is assigned to a lower (the lowest) priority.

```
FMSTR_SHORT_INTR = 1
```

The raw serial communication is handled by the FMSTR_Isr() interrupt service routine, while the protocol decoding and execution is handled in the FMSTR_Poll() routine. You typically call the FMSTR_Poll() during the idle time in the application ‘main loop’.

```
FMSTR_POLL_DRIVEN = 1
```

Both the serial (SCI/CAN) communication and the FreeMASTER protocol execution are done in the FMSTR_Poll() routine. No interrupts are needed: the FMSTR_Isr() code compiles to an empty function. When using this mode, you must ensure the FMSTR_Poll() function is called by an application at least once per ‘SCI character time’, which is the time needed to transmit or receive a single character.

In our example, we will use the short interrupt mode as this is the most versatile.

```
/*******************************************************************************
* Select communication interface (SCI, CAN, USB CDC or Packet Driven BDM)
*******************************************************************************/
#define FMSTR_SCI_BASE             0x4006D000   /* UART3 registers base address on K60 */
#define FMSTR_SCI_INTERRUPT    67                 /* UART3 interrupt vector on K60 */
```

Here we have to show to the FreeMASTER serial driver which SCI peripheral will be used for communication and its interrupt vector. The K60 Tower Board has UART3 mapped to the Tower Serial board (TWR-SER board). You have to modify the values if another platform / SCI channel is used.

```
#define FMSTR_DISABLE          0    /* Disable all the FreeMASTER functionalities */
```

We typically use FreeMASTER as a debugging tool, while we don’t want to have it in the release code. Setting the FMSTR_DISABLE to a non-zero value will remove FreeMASTER from your code.

```
#define FMSTR_USE_SCI         1    /* To select SCI communication interface */
#define FMSTR_USE_FLEXCAN     0    /* To select FlexCAN communication interface */
#define FMSTR_USE_USB_CDC     0    /* To select USB CDC communication interface */
#define FMSTR_USE_PDBDM        0   /* To select Packet Driven BDM comm. interface (optional) */
```

We will use UART communication in this example, so set FMSTR_USE_SCI to one while clearing the other to zero. We also have to enable the most important serial driver features

```
#define FMSTR_USE_READMEM      1    /* enable/disable memory read / write*/
#define FMSTR_USE_SCOPE        1    /* enable/disable scope support */
#define FMSTR_USE_RECORDER     1    /* enable/disable recorder support */
#define FMSTR_USE_APPCMD       1    /* enable/disable App.Commands support */
```

There are other options in the file not mentioned here for simplicity. Refer to the FMSTRSCIDRVUG.pgf document for more details.

2.4 FreeMASTER API short description

And finally, we must add some function calling to our code. Include the freemaster.h file in your code everywhere you are calling the FreeMASTER API. The function headers of all the functions you need to use are as follows:

For memory read/write and oscilloscope functionality, we need to use the three functions below.

```
FMSTR_BOOL FMSTR_Init(void);    /* general FreeMASTER internal vars. initialization */
```

```
void FMSTR_Poll(void);    /* polling call, use in SHORT_INTR and POLL_DRIVEN modes */
void FMSTR_Isr(void);     /* interrupt handler for LONG_INTR and SHORT_INTR modes */
```

```
FMSTR_BOOL FMSTR_Init(void)
```

This function initializes internal variables of the FreeMASTER driver and enables the communication interface (SCI, JTAG or CAN). This function does not change the configuration of the selected communication module; the module must be initialized before the FMSTR_Init() function is called.

The FMSTR_Init() function must be called before any other FreeMASTER driver API function.

```
void FMSTR_Poll(void)
```

In the poll-driven or short interrupt modes, this function handles the protocol decoding and execution. In the poll-driven mode, this function also handles the interface communication with the PC. Typically, you call the FMSTR_Poll() during the 'idle' time in the main application loop.

```
void FMSTR_Isr(void)
```

This is the interface to the interrupt service routine of the FreeMASTER serial driver. In the long or short interrupt modes, this function must be set as the interrupt vector calling address. On platforms where interface processing is split into multiple interrupts, this function should be set as a vector for each such interrupt.

For the recorder functionality, we need to use the following functions

```
/* Recorder API */
void FMSTR_Recorder(void)
void FMSTR_TriggerRec(void);
FMSTR_Recorder()
```

This function takes one sample of the variables being recorded using the FreeMASTER recorder. If the recorder is not active at the moment when FMSTR_Recorder is called, the function returns immediately. When the recorder is initialized and active, the values of the variables being recorded are copied to the recorder buffer and the trigger condition is evaluated.

```
FMSTR_TriggerRec()
```

This function forces the recorder trigger condition to happen, which causes the recorder to be automatically de-activated after post-trigger samples are sampled. This function can be used in the application when it needs to have the trigger occurrence under its control. This function is optional; the recorder can also be triggered by the PC tool or when the selected variable exceeds a threshold value.

2.5 What FreeMASTER API functions do I have to handle in my application, and where?

Right now, we have an empty project prepared with all the necessary serial driver files included. In this example, the short interrupt approach is used. We need to follow four steps to make the code functional. FreeMASTER has to be initialized, the interrupt service routine has to be assigned to the IRQ vector and the pooled part of the code must be called periodically. Of course, we need to include the freemaster.h file

in each file where the FreeMASTER API function is called: in this case, the main.c and kinetis_sysinit.c files.

1. Call FMSTR_Init(void) just once at the code start, typically after the start-up code, at the beginning of the main function.
2. Call FMSTR_Poll(void) periodically in your code. A typical place is in the main loop:

```
int main(void)
{
    ConfigureMCU();
    /* FreeMASTER internal variables initialization */
    FMSTR_Init();

    /* main loop */
    for(;;) {
          /* call function periodically*/
          FMSTR_Poll();
    }
}
```

3. FMSTR_Isr() must be assigned to the UART3 interrupt vector. The kinetis_sysinit.c file contains the vector table definition. Here, we can define the called FreeMASTER:

```
  (tIsrFunc) FMSTR_Isr,      /* 67 (0x0000010C) (prior: -) UART 3 status sources */
  (tIsrFunc) FMSTR_Isr,      /* 68 (0x00000110) (prior: -) UART 3 error sources */
```

4. The last thing is to configure all the necessary peripheries used in the example. The clock gates are enabled, and the interrupt controller and UART have to be configured. After this fourth step, the embedded code is ready to be compiled and run. Just build it, load it, and run the code. Now we have to run and configure the FreeMASTER PC Application. The counter variable is here in the code and will be visualized in FreeMASTER.

```
/* ARM Cortex M4 implementation for interrupt priority shift */
#define ARM_INTERRUPT_LEVEL_BITS          4
#define IRQ(x)              ((x)-16)
#define ICPR_VALUE(x)       (unsigned short)(IRQ(x)/32)
#define ICPR_SHIFT(x)       (unsigned short)(IRQ(x)%32)

/* Setting interrupt controller */
#define NVIC_SetIsr(src,ipr)\
{ NVIC_ICPR_REG(NVIC_BASE_PTR,ICPR_VALUE(src)) |= (unsigned long)(1l << ICPR_SHIFT(src));\
  NVIC_ISER_REG(NVIC_BASE_PTR,ICPR_VALUE(src)) |= (unsigned long)(1l << ICPR_SHIFT(src));\
  NVIC_IP_REG(NVIC_BASE_PTR,IRQ(src)) |= (unsigned long)((unsigned \
long)(ipr)<<ARM_INTERRUPT_LEVEL_BITS);}


/* UART3 baudrate */
#define brate 9600
/* Actual BUS CLOCK value */
#define bclk 21e6
/* Macro to calculate BAUDRATE register value */
#define CALC_SBR(brate,bclk)  (unsigned short)((double)bclk/(16.0*(double)brate))
```

```
#define CALC_BRFA(brate,bclk) ((unsigned
short)(((((double)bclk/(16.0*(double)brate))-(double)CALC_SBR(brate,bclk))*32.0)+0.5))

int counter = 0;

int main(void)
{

    /* Enable clock to UART3 */
    SIM_SCGC4 |= SIM_SCGC4_UART3_MASK;
    SIM_SCGC5 |= SIM_SCGC5_PORTC_MASK;

    /* Initialize pins shared with UART3 port */
    /* Set PTC16 to UART mode and high drive strength */
    PORTC_PCR16 |= PORT_PCR_MUX(3)|PORT_PCR_DSE_MASK;
    /* Set PTC17 to UART mode and high drive strength */
    PORTC_PCR17 |= PORT_PCR_MUX(3)|PORT_PCR_DSE_MASK;


    /* Enable UART interrupt vectors and set their priority to 5 */
    /* UART3 status sources number 51 */
    NVIC_SetIsr(INT_UART3_RX_TX, 5);
    /* UART3 error sources number 52  */
    NVIC_SetIsr(INT_UART3_ERR, 5);


    /* Set up UART3 periphery */
    UART3_BDH = ((CALC_SBR(brate,bclk)>>8)&0x1f);
    UART3_BDL = ((CALC_SBR(brate,bclk)>>0)&0xff);
    UART3_C4 = ((CALC_BRFA(brate,bclk)>>0)&0x1f);

    /* Enable UART3 Transmitter and receiver  */
    UART3_C2 = UART_C2_TE_MASK|UART_C2_RE_MASK;
```

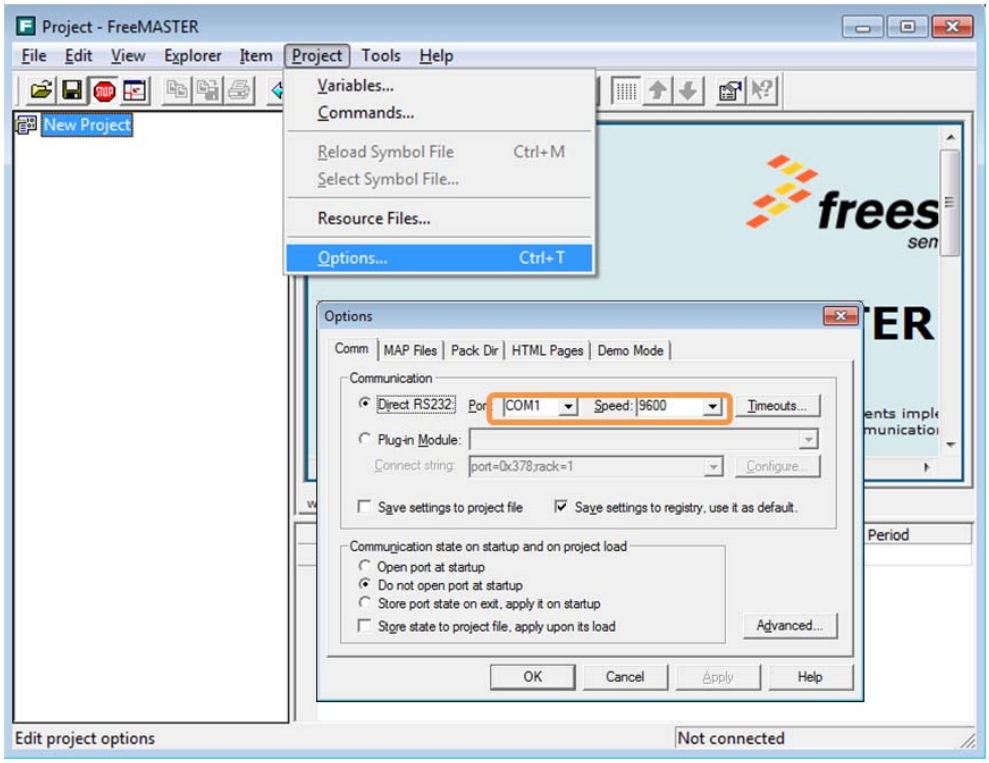
This was the last step in the embedded application. We are ready to read out variable values from our embedded application.

3 FreeMASTER PC application configuration

3.1 How to set up the communication channel

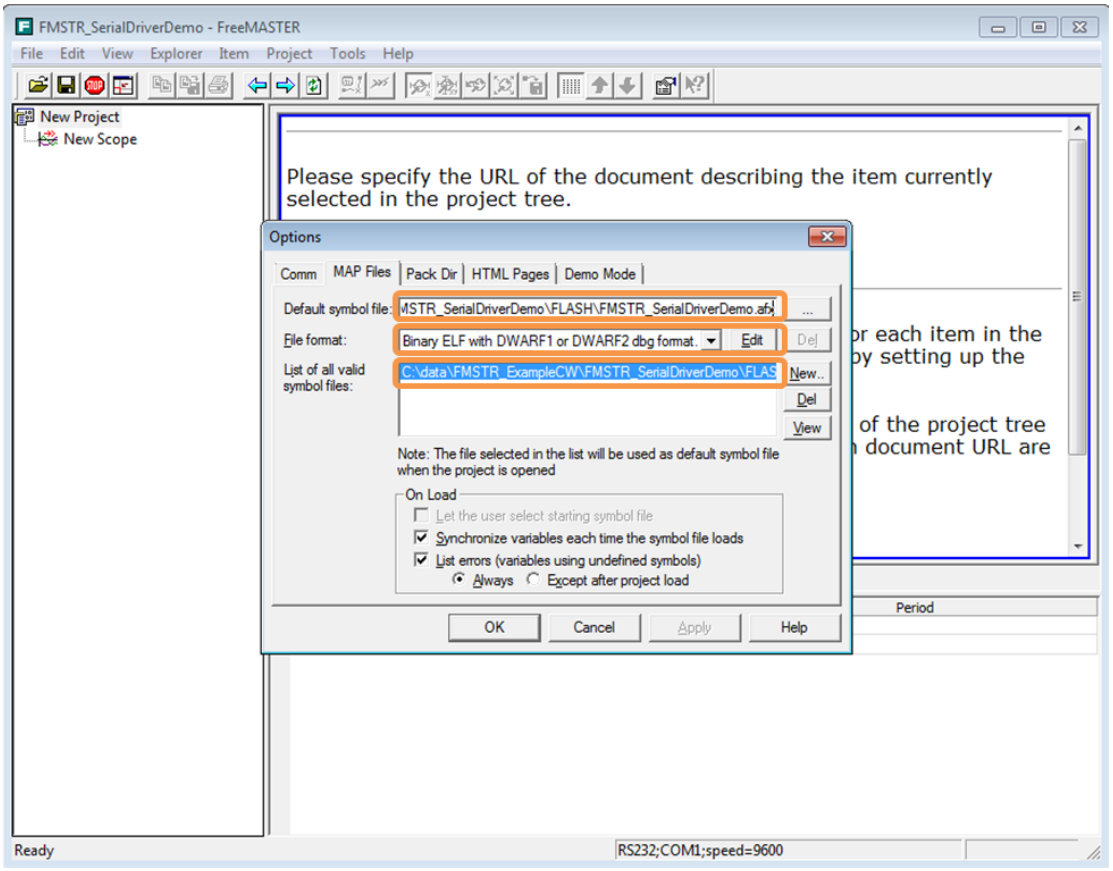
First of all, we have to run the FreeMASTER application from the Windows start menu. After the application is started, go to Project -> Options -> Communication

where we set the communication port number and baudrate.



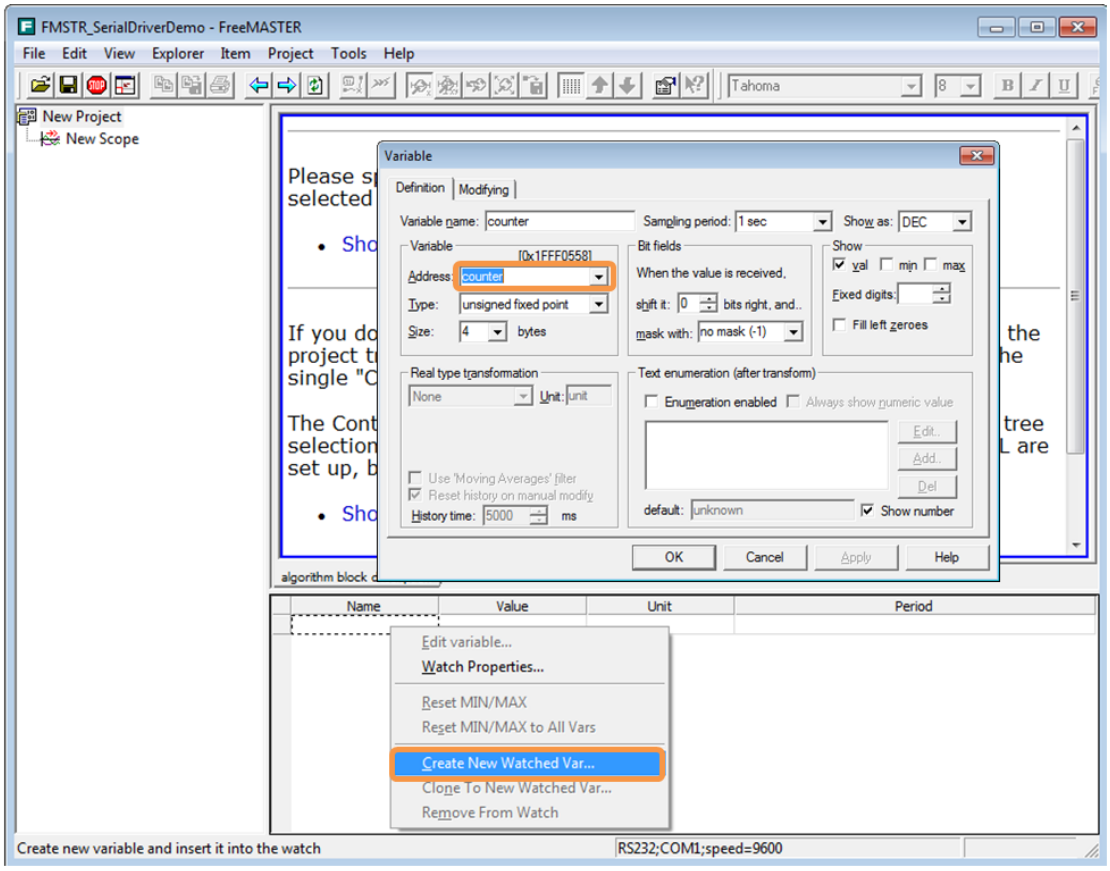
3.2 MAP file selection

In the FreeMASTER window, we can easily choose the observed variables used in the embedded application. FreeMASTER parses information stored in the MAP file during embedded application linkage. The MAP file has all the information about the variables, their names, types, and addresses used in the embedded application. The path to the MAP file must be set in the FreeMASTER PC application. Select the correct path to the file and the file format created by the development tool. In this case, the MAP file has an *.afx extension.

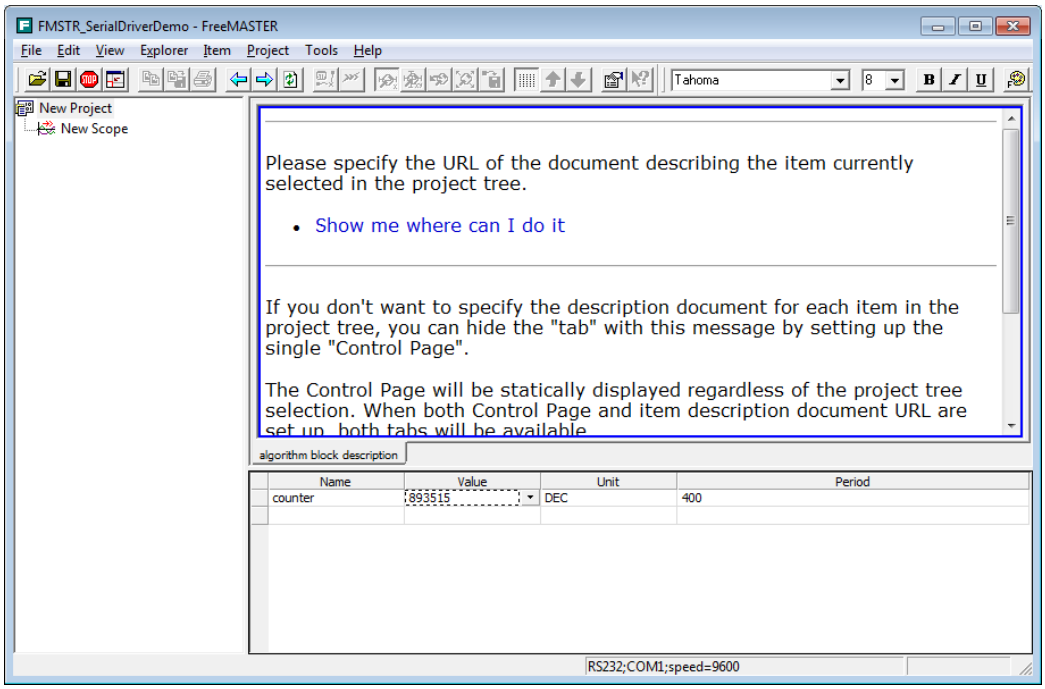


3.3 How to choose the observed variable

Now, we are ready to observe the arbitrary variable values used in the embedded application. Create a new watched variable by right mouse-clicking on the variable's grid and select Create New Watch Var. In the existing project, we have only a single variable named counter defined just above the main. Write the variable name to the address field. If the variable is selected by the drop-down menu, the "Variable name:" edit box is filled with the correct variable name. Then, click OK and the variable will appear in the grid. By clicking the STOP icon, the communication is started. Now, the counter variable will appear in the grid and its value is refreshed.

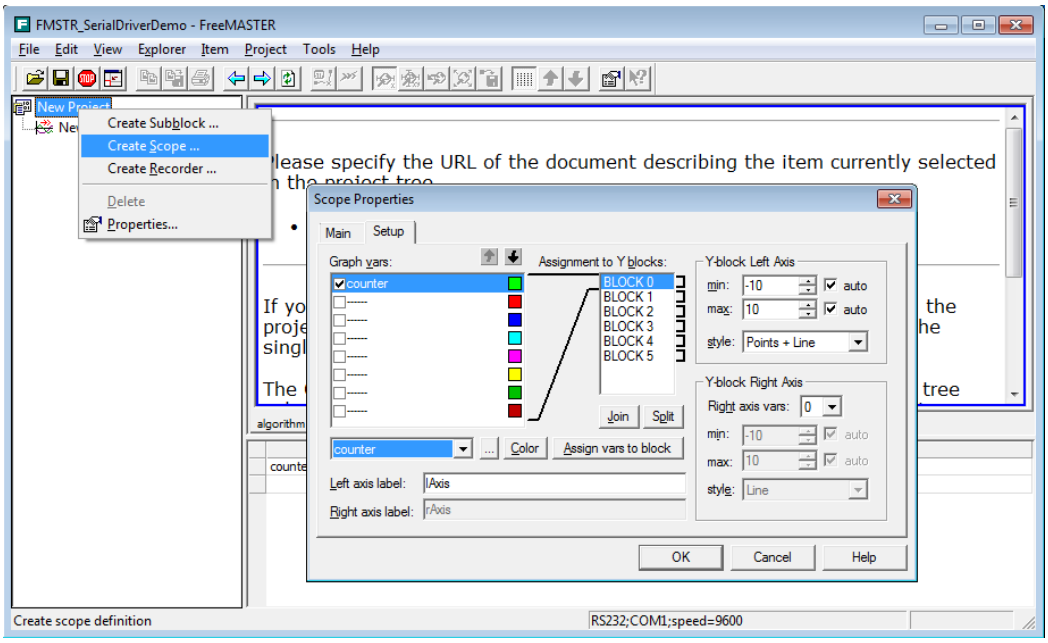


The period value shows how often the variable's value is refreshed in milliseconds. The 16-bit variable counter is incremented in the main loop, so the variable overflows faster than the 400 ms refresh rate, and the value shown in the grid seems to be chaotic. In fact, this is due to undersampling — Nyquist theorem is not fulfilled. The same situation can be observed for the Scope.

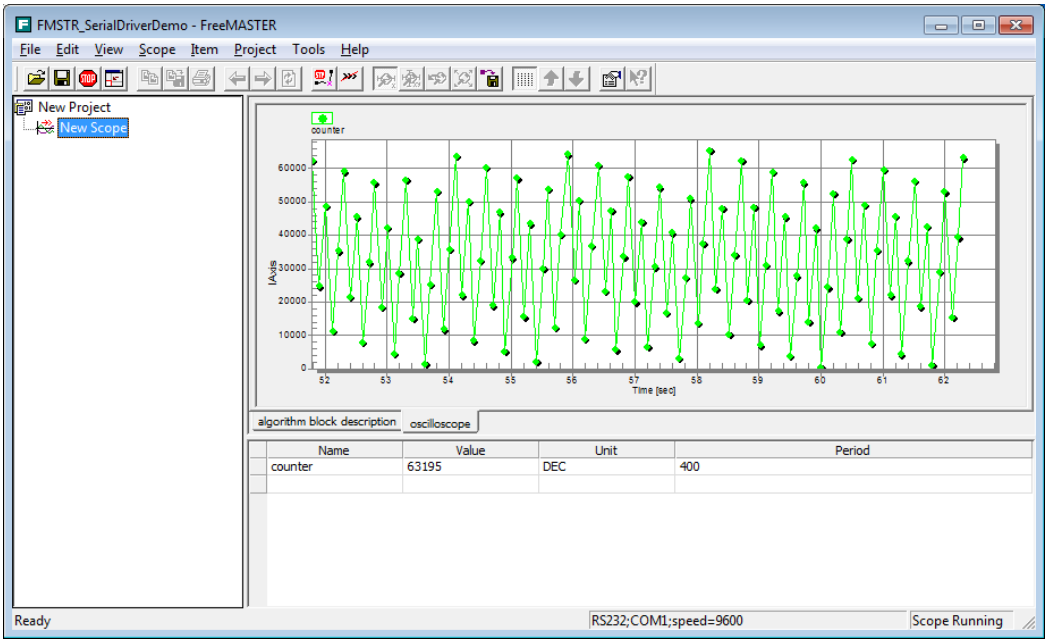


3.4 How to set up the Scope

We can create a new Scope just by right clicking on the project and selecting Create Scope. Then, in the drop-down menu, choose the counter variable, check it in the Graph vars list, and select the “Points+Line” style.



If we use a 400 ms refresh time, we have only three samples before counter overflows. Even if we have the fastest communication possible, we can see only a few values out of the 2^{16} states the counter variable can get into. If we want to see all the counter increments, we need to use the Recorder feature.



3.5 How to set up the Recorder

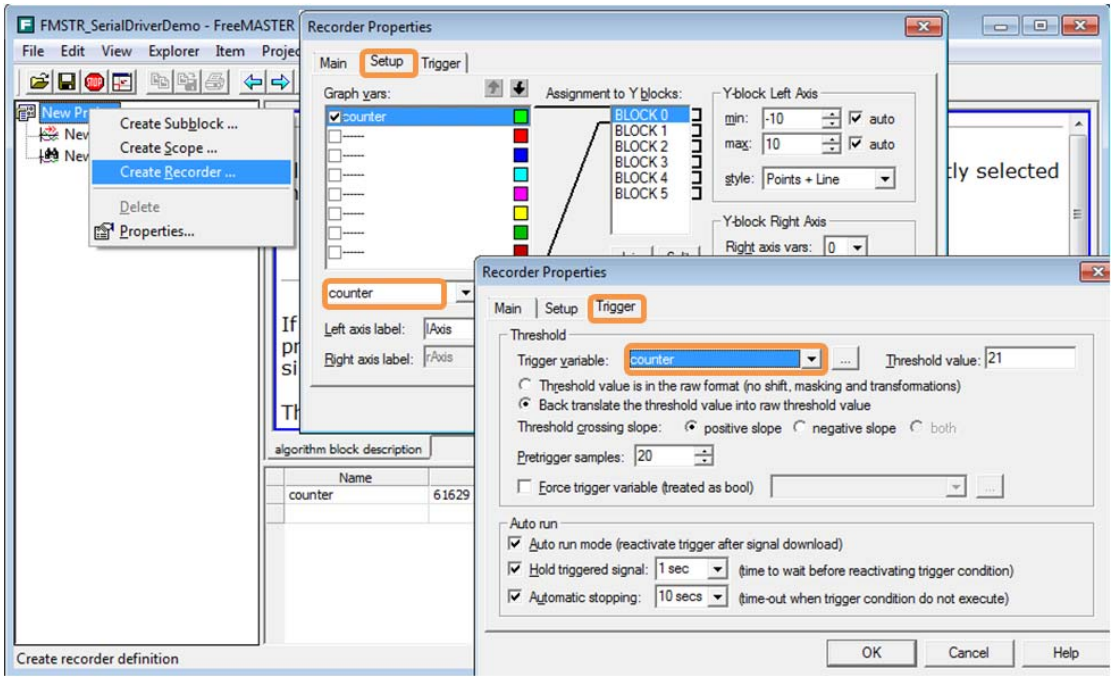
To enable the recorder in your embedded application, we have to show only a point in the embedded project software where the variable values will be buffered. Use the FMSTR_Recorder() function calling to do this. Modify the main loop simply by adding the Recorder function call

```
for(;;) {
                counter++;
                FMSTR_Recorder();
                FMSTR_Poll();
        }
```

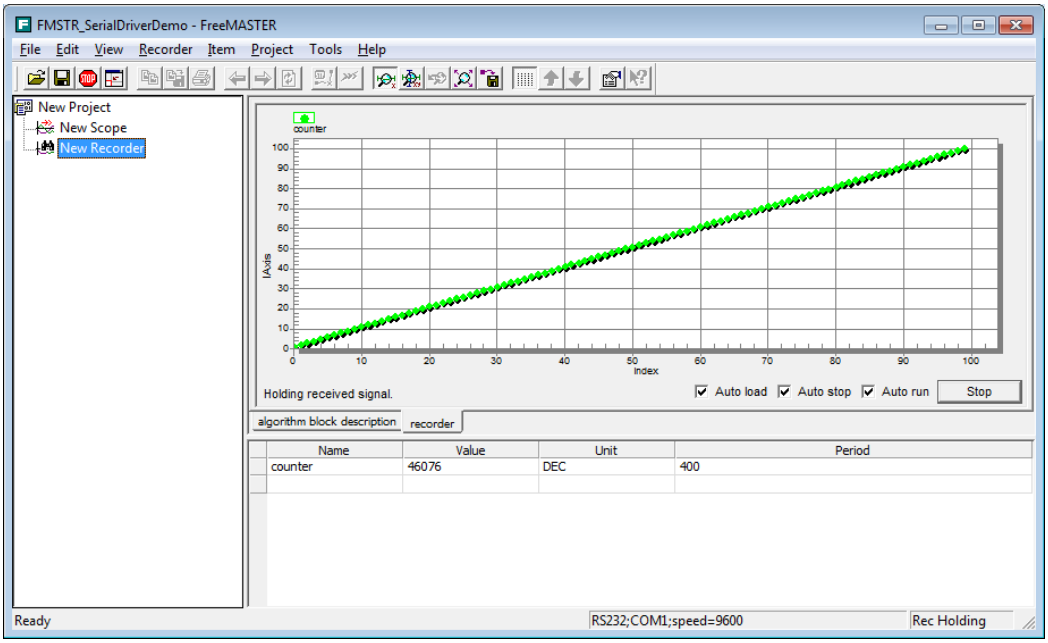
In each cycle after the counter variable is incremented, the FMSTR_Recorder() function is called and the counter value is buffered to the Recorder buffer. It depends on the FreeMASTER PC application setting when the buffer will be transferred to the PC. This may happen after the trigger conditions are fulfilled, or by calling the FMSTR_TriggerRec() function.

3.6 How to configure the Recorder in the FreeMASTER PC application

In the FreeMASTER PC application, right mouse-click to New Project and create Recorder. Then add the counter variable to the Graph vars in the setup and define the trigger in Trigger. We have only the counter variable, so we will select counter. The recorder buffer (in the embedded application) will be transferred to the FreeMASTER PC application after the counter reaches the “Threshold” value of 21. Also, 20 pre-triggered values will be transferred.



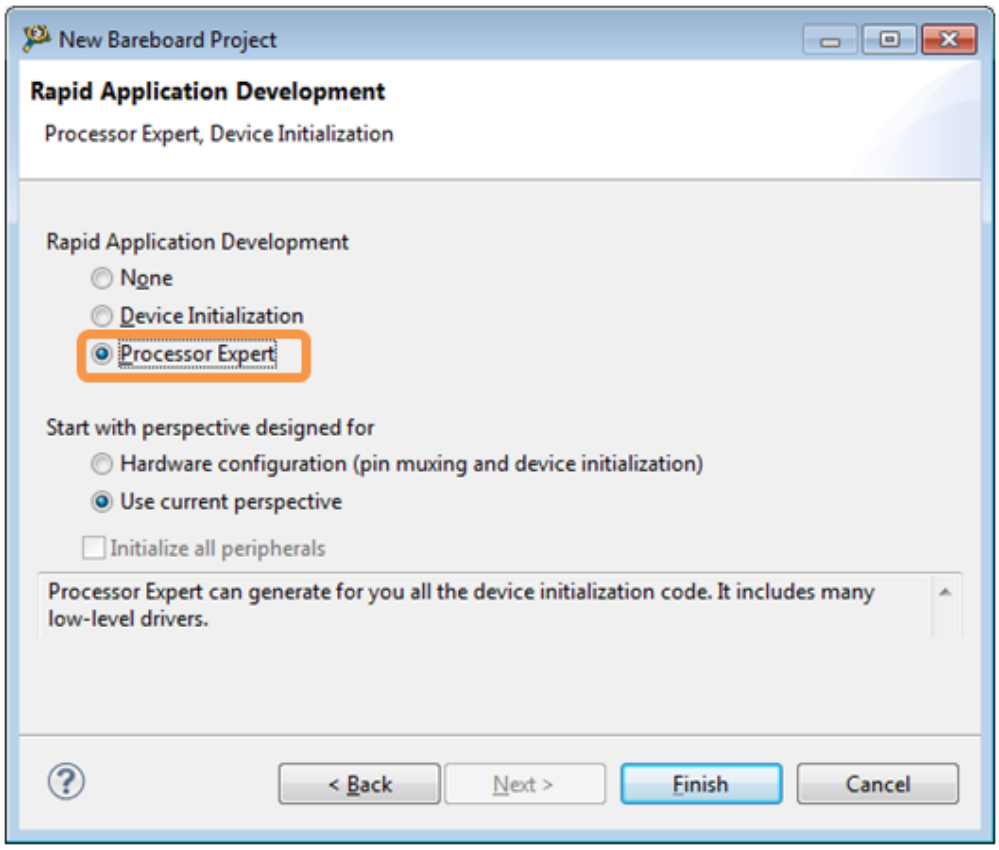
Using the Recorder, we can see all values of the counter variable from 0 to a defined buffer length, as opposed to the Scope view. This is the way to visualize fast actions. The buffer length is limited by the amount of RAM available in the microprocessor used.



4 Is there any easier way to integrate FreeMASTER into my project?

4.1 How to set up a new project with FreeMASTER drivers integrated by the Processor Expert

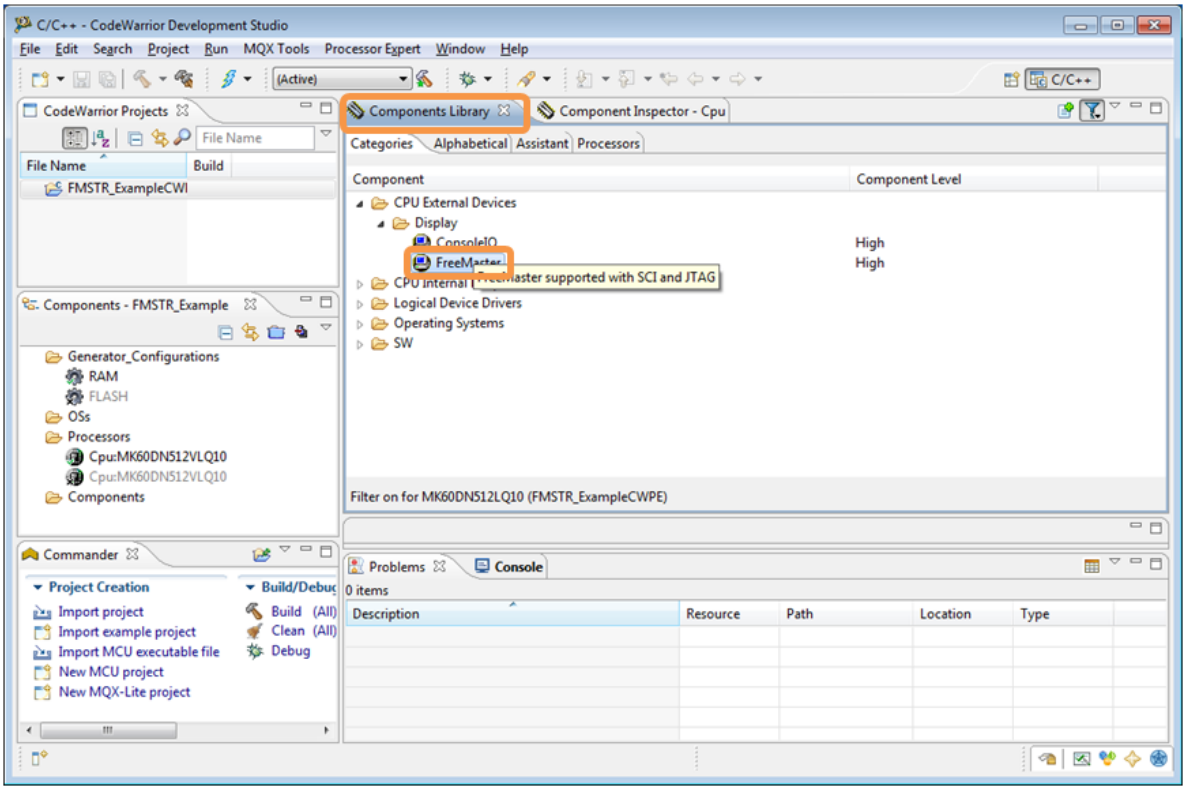
The Processor Expert is a rapid development tool integrated with CodeWarrior, which helps us with a graphical Device Initialization Tool and a set of predefined functions. One of the many offered components is the FreeMASTER serial driver. We need to simply click a few lines to set up FreeMASTER and UART to have a completely pre-defined project stationery with FreeMASTER functionality. Run the CodeWarrior in the same way as in the 2nd section and choose the New Project Wizard, but instead select Rapid Application Development -> Processor Expert and click finish.



4.2 How to add FreeMASTER serial drivers to the project

The Processor Expert will generate project stationery. Then, go to Component Library -> CPU External Devices -> Display -> FreeMASTER and double click the FreeMASTER component.

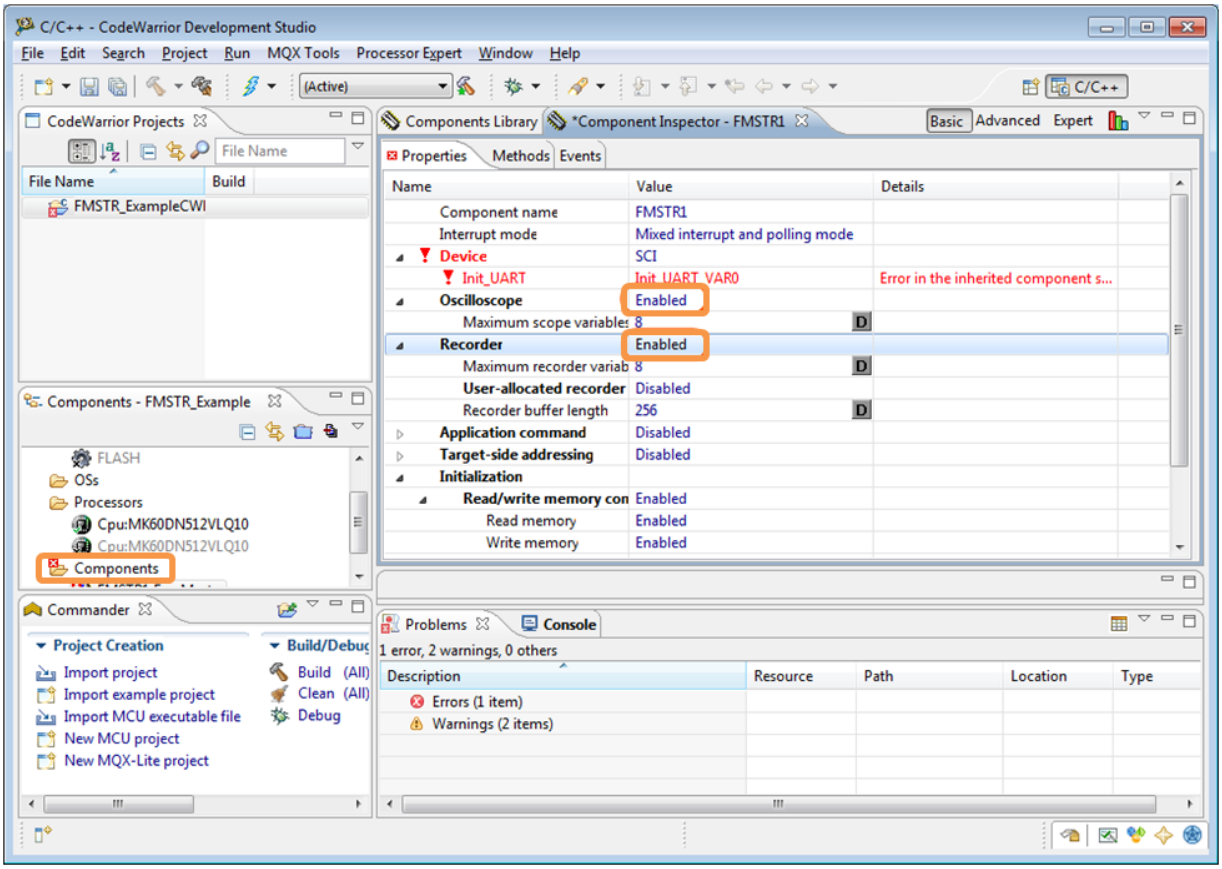
The component will be added to the Component Explorer window.



4.3 How to set up the FreeMASTER functionality

Go to the Component inspector and modify the FMSTR1 component. We can enable the Scope and Recorder functionality.

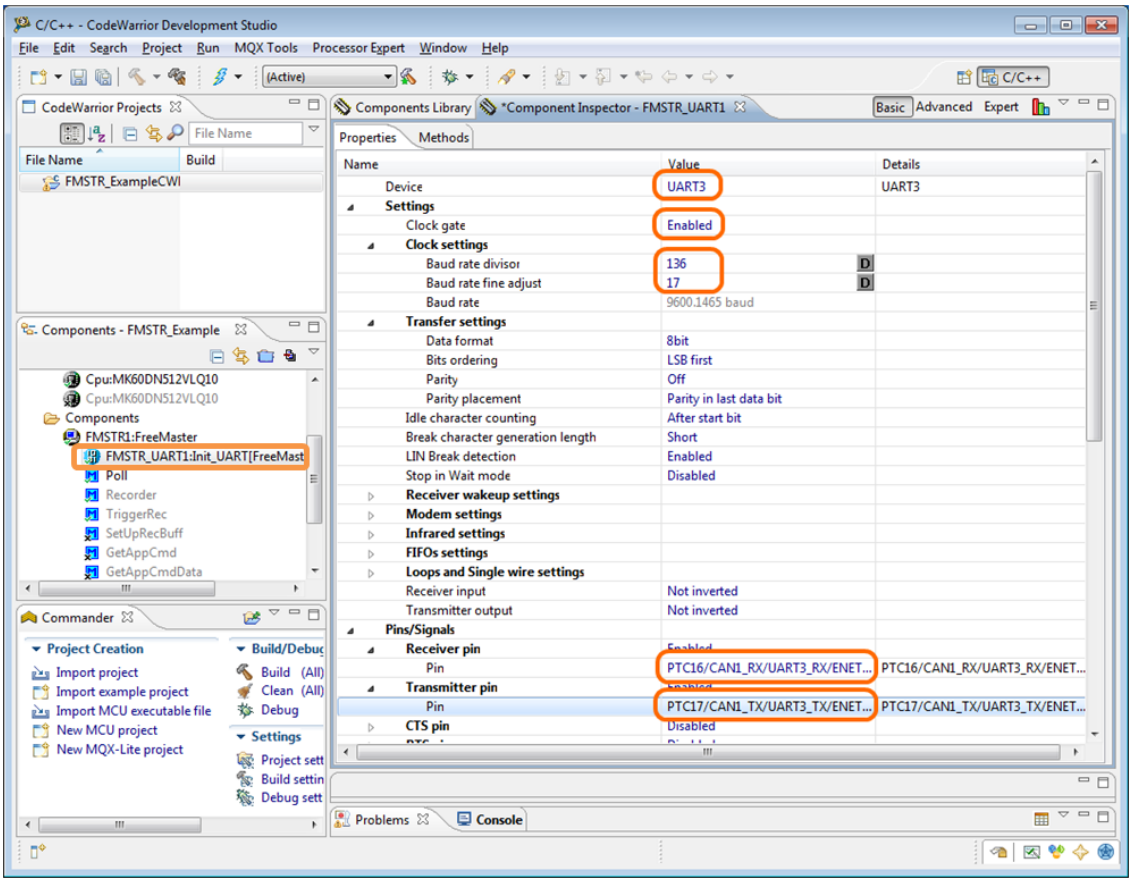
The short interrupt mode is preset. We can also see the red warning that the inherited UART component has an incorrect setting, so we have to modify it.



4.4 How to set up the UART parameters

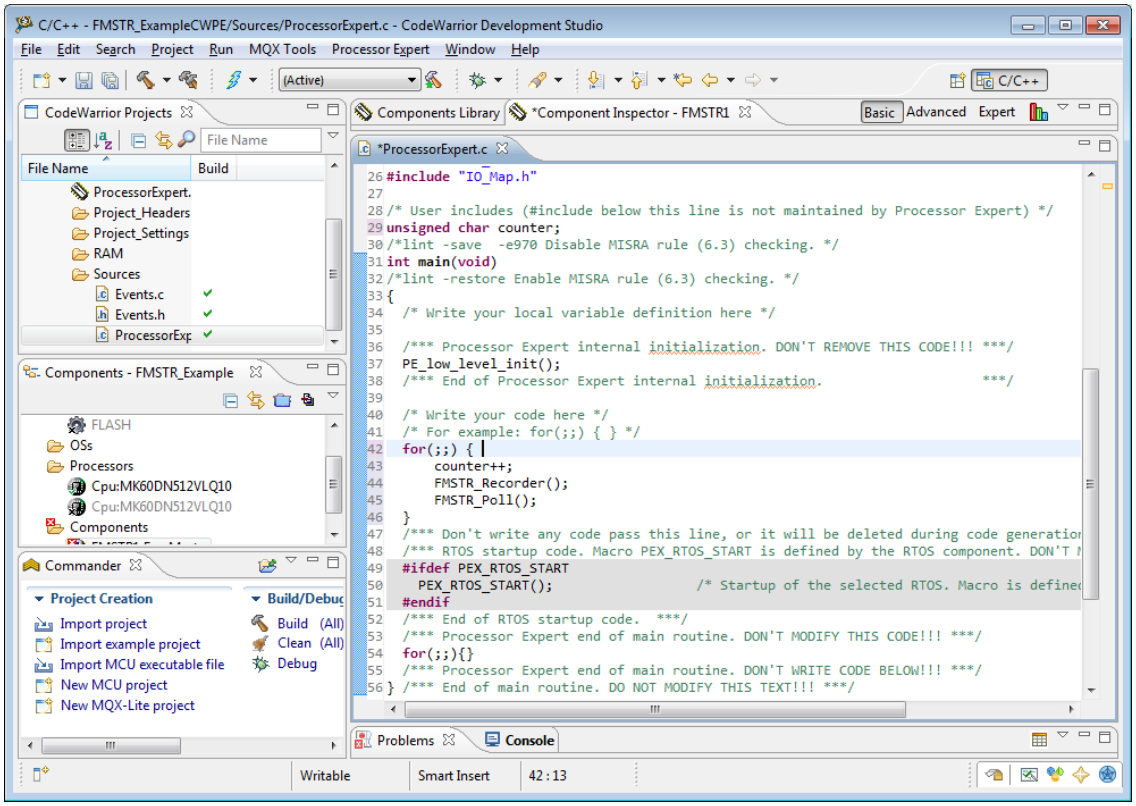
Click on the FMSTR_UART1 component in the Component Explorer and the component settings appear in the Component Inspector, where all the necessary settings are done.

We will select the UART3 module, enable the clock to the module, and set the baud rate divisor to get a proper communication speed (9600 bps). The pins used by UART3 must be selected and enabled:



4.5 How to modify code in the application

The only thing in the embedded code we have to do is to add the FMSTR_Poll() functional calling to the code where it will be called periodically. All the rest is resolved in the Processor Expert starting code. We can modify the ProcessorExpert.c file in the following way:



Then, the application is prepared and we can run it and again observe the counter variable in the FreeMASTER PC application.

5 Final words

There are situations when developing embedded applications where we need to see the application internal states in real time or where a graphical visualization helps with the debugging. FreeMASTER offers both of these in a very appropriate way and is easy to use. Adding serial drivers to the application requires only a few steps, and then we can enjoy the user friendly FreeMASTER application to observe those internal states.

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN4752
Rev. 0
5/2013