EFM32®

... the world's most energy friendly microcontrollers

EFM32 Debug and Trace

AN0043 - Application Note

Introduction

This application note gives an overview of the different software debug functions available with the EFM32 microcontrollers. Both the hardware connection and software configuration for Serial Wire Debug and Instruction Trace are described and demonstrated in the included software examples.

This application note includes:

- This PDF document
- Source files (zip)
  - Example C-code
  - Multiple IDE projects

ARM ZERO TINY GECKO LEOPARD GIANT WONDER

SILICON LABS

1 Introduction

The EFM32 microcontrollers use the ARM CoreSight™ on-chip debug and trace interface. Serial Wire Debug technology (SWD), specifically the Serial Wire Debug Port (SW-DP) for the EFM32, is used as the interface between the on-chip debug module and the development environment on a computer. SW-DP is a 2-pin debug interface which offers a high performance and low pin count alternative to JTAG. SWD is often used as an acronym for both the SW-DP and SWJ-DP (Serial Wire JTAG Debug Port). The EFM32 does not implement SWJ-DP.

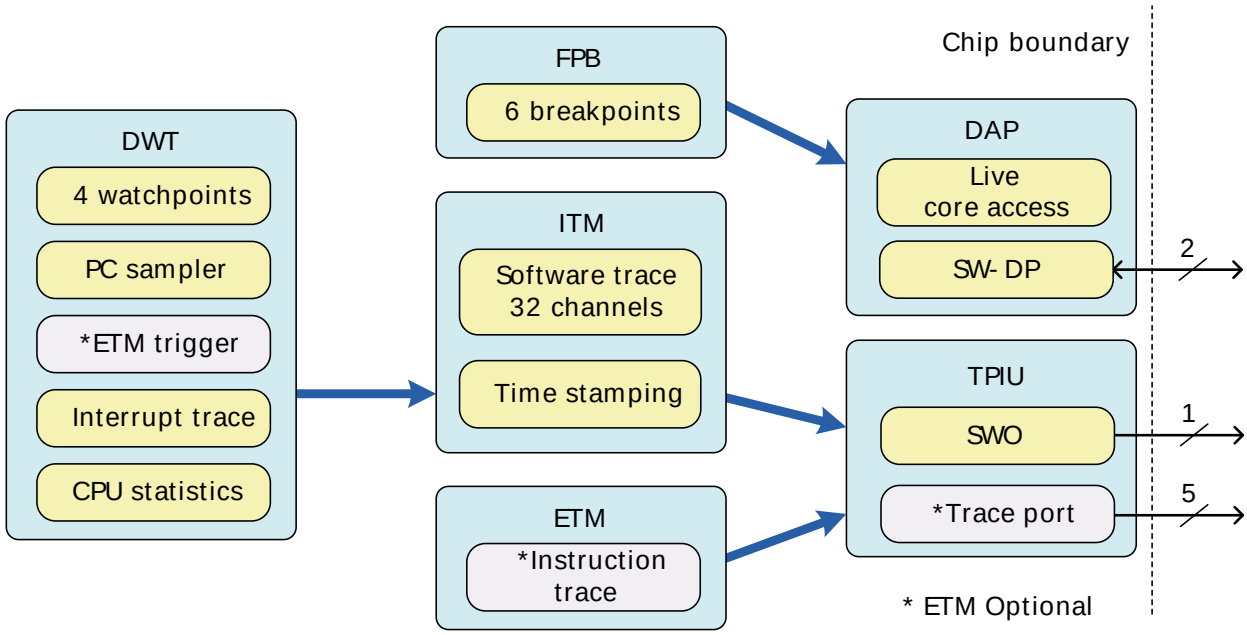
All EFM32 devices also include the Serial Wire Output interface which is a one wire interface for the Instrumentation Trace Macrocell (ITM) which is a CoreSight module that provides coarse-grained time stamping and software driven trace information.

Some device families, including Leopard Gecko, Giant Gecko and Wonder Gecko devices also include the CoreSight Embedded Trace Macrocell (ETM). This module provides full instruction trace over a separate high speed trace interface. Instruction trace allows the software developer to get a more complete view of what goes on inside the microcontrollers CPU.

2 EFM32 Debug and Trace System

An overview of all the different modules that makes up the CoreSight system inside the EFM32 is given in Figure 2.1 (p. 3) . A short description of the different modules are given in the following section.

Figure 2.1. Debug Overview



2.1 Serial Wire Debug Port (SW-DP)

The SW-DP provides a low pin count bi-directional serial connection to the Debug Access Port (DAP) with a reference clock signal for synchronous operation. This is the only connection from outside the EFM32 to access core registers and memory within the MCU. Through this two wire interface the debugger has real-time access to system/peripheral memory and debug configuration registers. Access is even available without halting the processor.

The communication over the SW-DP uses a three phase protocol:

- A host-to-target packet request.
- A target-to-host acknowledge response.
- A data transfer phase, if required. This can be target-to-host or host-to-target, depending on the request made in the first phase.

More information on the CoreSight Serial Wire Debug interface and protocol can be found here:

- **http://infocenter.arm.com/help/topic/com.arm.doc.ddi0314h/DDI0314H_coresight_components_trm.pdf**
- **http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0314h/Babcfacj.html**

2.2 Instrumentation Trace Macrocell (ITM)

The ITM is a an application-driven trace source that supports printf() style debugging to trace operating system and application events. The ITM is responsible for generating packets based on information from multiple sources. Software can write directly to ITM stimulus registers to generate packets. The

Data Watchpoint and Trace (DWT) module can also generate packets that are formatted and output by the ITM. Finally the ITM can generate timestamps using the 21 bit counter within the ITM module. The packets generated are sent out over the Serial Wire Output (SWO) interface.

2.3 Serial Wire Output (SWO)

The Serial Wire Output module formats and sends the data from the ITM over a one wire connection to the debugger. Typically data is transmitted in bytes with either a Manchester or UART protocol.

2.4 Embedded Trace Macrocell (ETM)

The ETM module provides full instruction trace for the CPU. Data transferred through the Trace Port Interface Unit (TPIU), can be reconstructed to full program execution. Reconstruction, typically by debug software running on a computer, results in cycle accurate execution history for parts of the code. The amount of trace data captured depends on the trace buffer size in the debugger.

2.5 Data Watchpoint and Trace (DWT)

The data watchpoint and trace unit contains several comparators and counters that can be configured to generate events for the ITM and ETM modules. The most common features include data breakpoints, data/interrupt triggers and PC (Program Counter) sampling. This module, combined with the ITM can perform low-bandwidth data trace.

2.6 Flash Patch and Breakpoint unit (FPB)

This module consist of several address matching tags. When a match occurs the access can be rerouted from flash to a special part of the SRAM. This permits patching flash locations for breakpointing and quick fixes or changes. This is the unit that handles hardware breakpoints for code.

More information on the different CoreSight components can be found in the Cortex-M3 Technical Reference Manual from ARM:

- **http://infocenter.arm.com/help/topic/com.arm.doc.ddi0337i/DDI0337I_cortexm3_r2p1_trm.pdf**

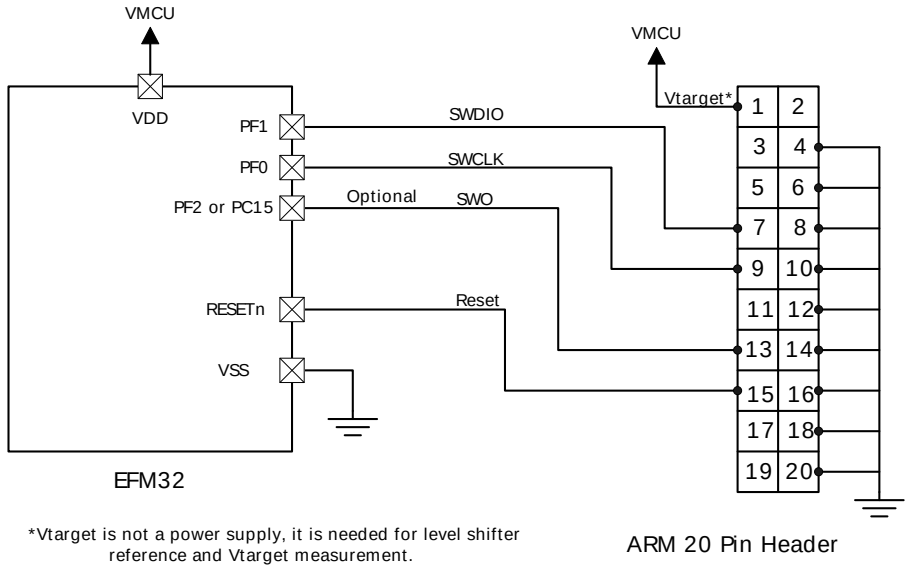
3 Debug Connection with SW-DP

SW-DP is a packet based protocol with a unidirectional clock signal and a bi-directional data signal. The master, which is usually the debugger, drives the clock signal. Both master and slave can drive the data-line.

3.1 Serial Wire Hardware Connection

The minimum hardware connection needed for full debug functionality is illustrated in Figure 3.1 (p. 5) .

Figure 3.1. Minimum serial wire debug connection.



Make sure that the debug lines are kept short. Especially the chip unlock feature of the EFM32 relies on high speed communication over the serial wire debug interface. The Serial Wire Output (SWO) line is optional and not needed for plain code uploading and stop/go debugging, but it is needed for the more advanced features such as instrumentation trace. The reset signal is also optional, but highly recommended. Without it, the EFM32 lock/unlock feature will not work and it can be difficult to access the chip if the code for some reason disables clocks or debug pins etc.

For a detailed description of the hardware design considerations for the EFM32, please see application note AN0002 EFM32 Hardware Design Considerations.

3.2 Debugging with the Energy Micro Kits

Both the Energy Micro starter kits and development kits include a full featured Segger J-Link™ debug interface to take full advantage of the serial wire debug features of the EFM32 microcontrollers (SW-DP and SWO). The kits can be configured to debug either the EFM32 part mounted on the kit itself, or it can be used to program and debug an EFM32 mounted on your own board. As long as you are working with EFM32 microcontrollers, you do not need any other debug adapter.

Note

Not all of the Energy Micro kits include a full J-Trace enabled debugger (needed for ETM Instruction Trace). Please refer to the kit documentation for your kit.

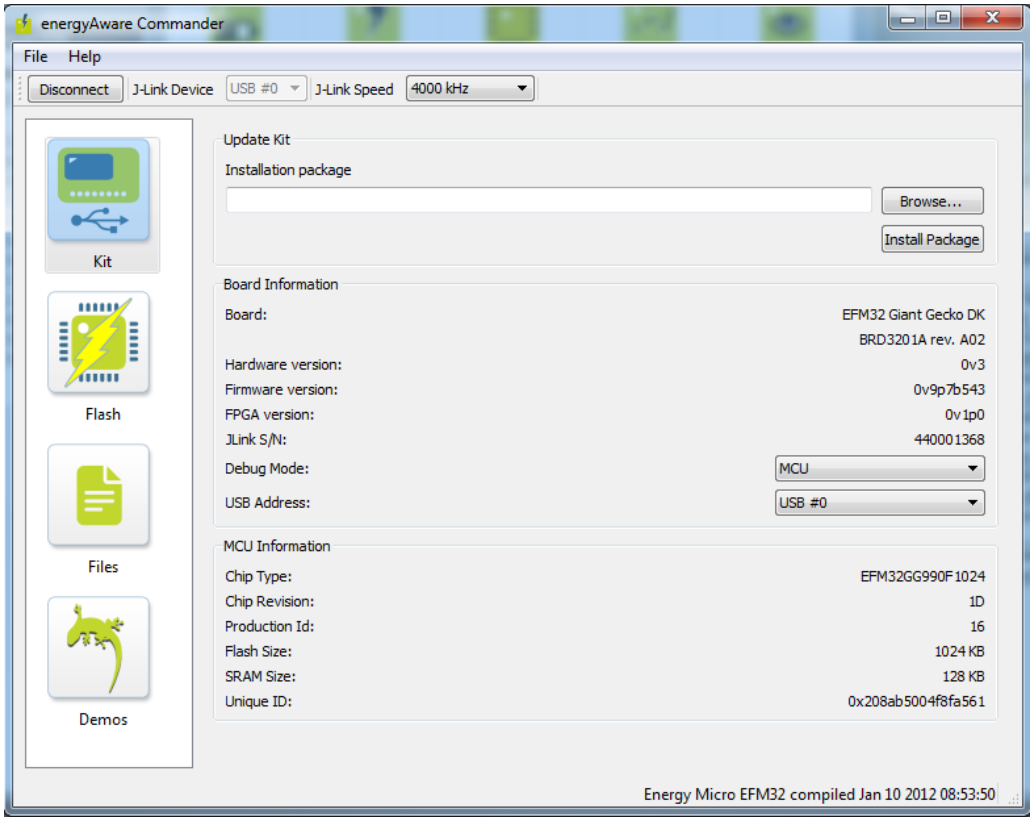
3.3 Software Tools

Software tools, typically in the form of an IDE (Integrated Development Environment) is needed to take full advantage of the EFM32's debug features. The following section describes how to set up a debug-session with one of our kits in combination with an IDE from either IAR or ARM (µVision IDE from Keil, from now on called referred to as Keil).

3.3.1 Upload Code and Debug with Keil

1. Make sure your kit is connected to the computer and correctly identified as a Segger J-Link device. Also make sure that the kit debug mode is configured correctly. You can use the Energy Aware Commander to change debug mode and check if the J-Link can contact the target EFM32 device. Figure 3.2 (p. 6) illustrates correct Energy Aware Commander output when the J-Link on the kit is working.

Figure 3.2. eAcommander connected correctly to computer.



2. In Keil, check that the project debug options and J-Link Target Driver options are configured correctly. See Figure 3.3 (p. 7) and Figure 3.4 (p. 7) .

Figure 3.3. Keil debug options.

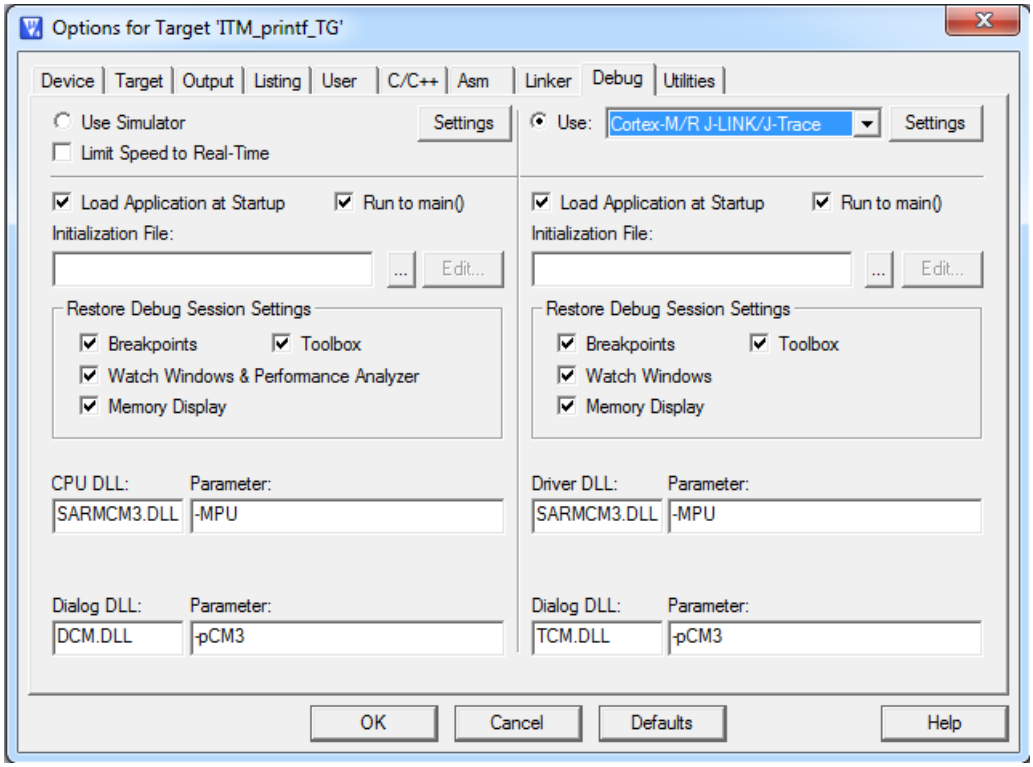
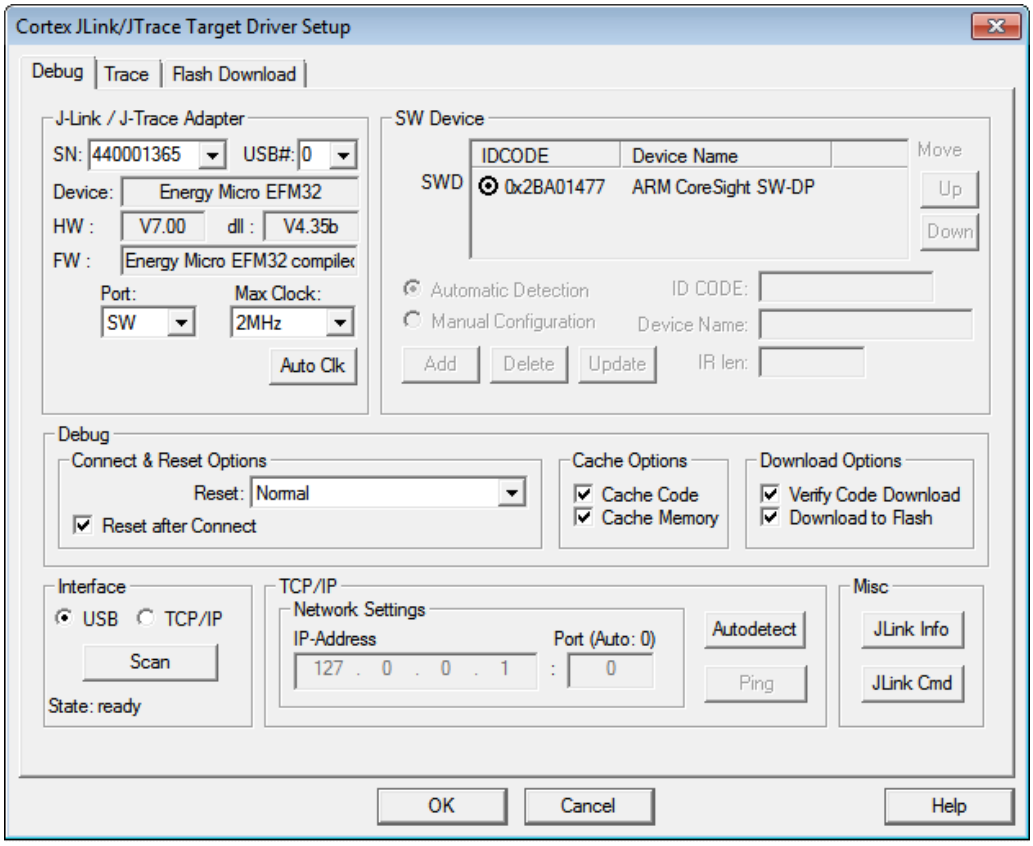
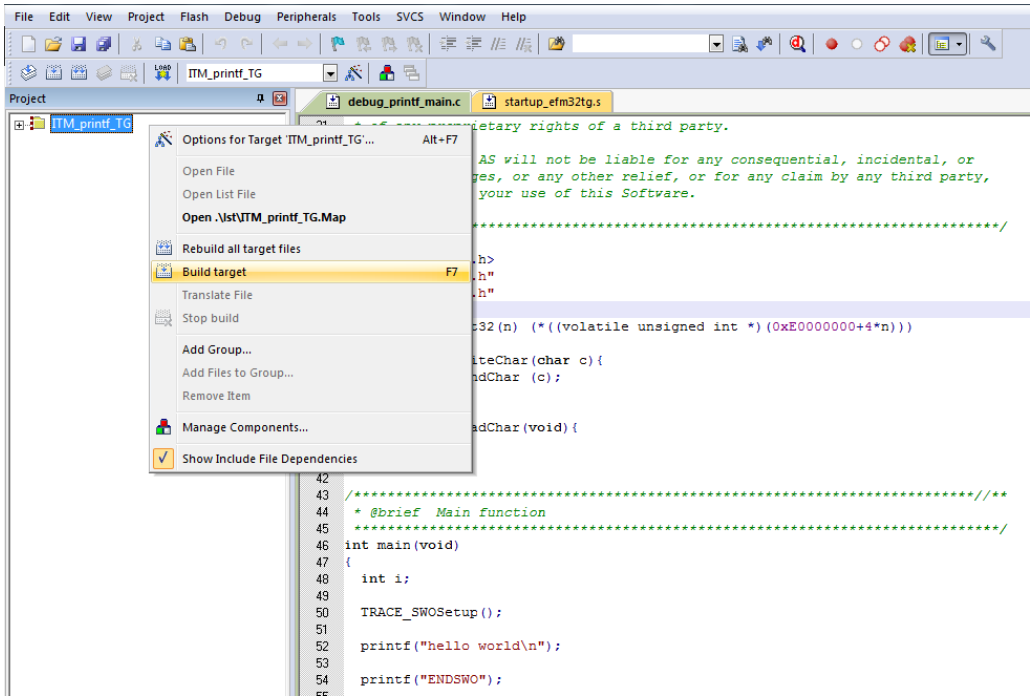


Figure 3.4. Keil J-Link Target Driver options.



3. Build target by right clicking project or pressing F7.

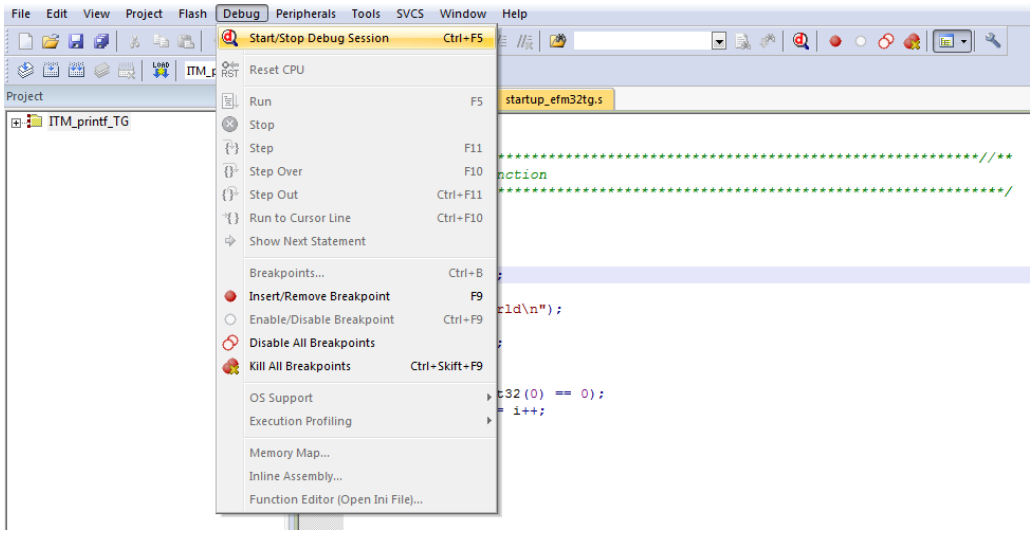
Figure 3.5. Keil build target.



4. Start the debug session by pressing Ctrl+F5.

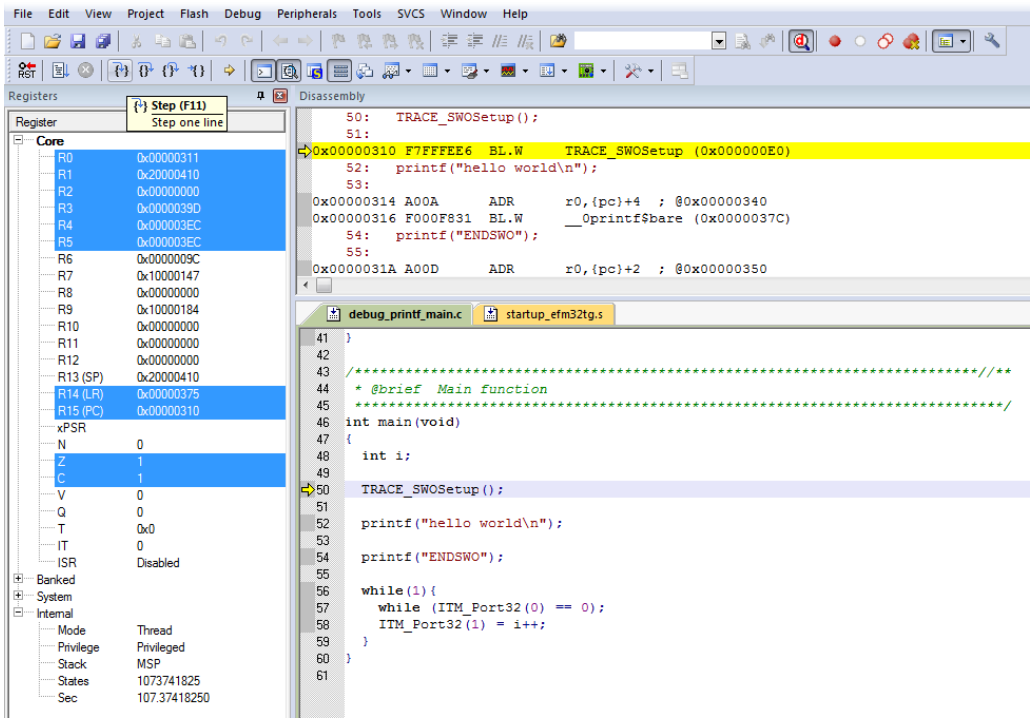
   Note: If you have not selected the "Update Target before debugging" in the options->utilities menu, you have to upload any new code before starting the debug session.

Figure 3.6. Keil start or stop debug session.



5. You are now in a debug session, single stepping and other debug feature are available in the top left corner.

Figure 3.7. Keil in debug session.



For more information regarding debugging in Keil uVision, please refer to the uVision documentation:

- **http://www.keil.com/product/brochures/uv4.pdf**

3.3.2 Upload Code and Debug with IAR

1. Make sure your kit is connected to the computer and correctly identified as a Segger J-Link device. Also make sure that the kit debug mode is configured correctly. You can use the Energy Aware Commander to change debug mode and check if the J-Link can contact the target EFM32 device. Figure 3.2 (p. 6) illustrates correct Energy Aware Commander output when the J-Link on the kit is working.

2. In IAR, check that the project options regarding debugger and J-Link are configured correctly. See the different screenshots below:

Figure 3.8. IAR debugger setup settings.

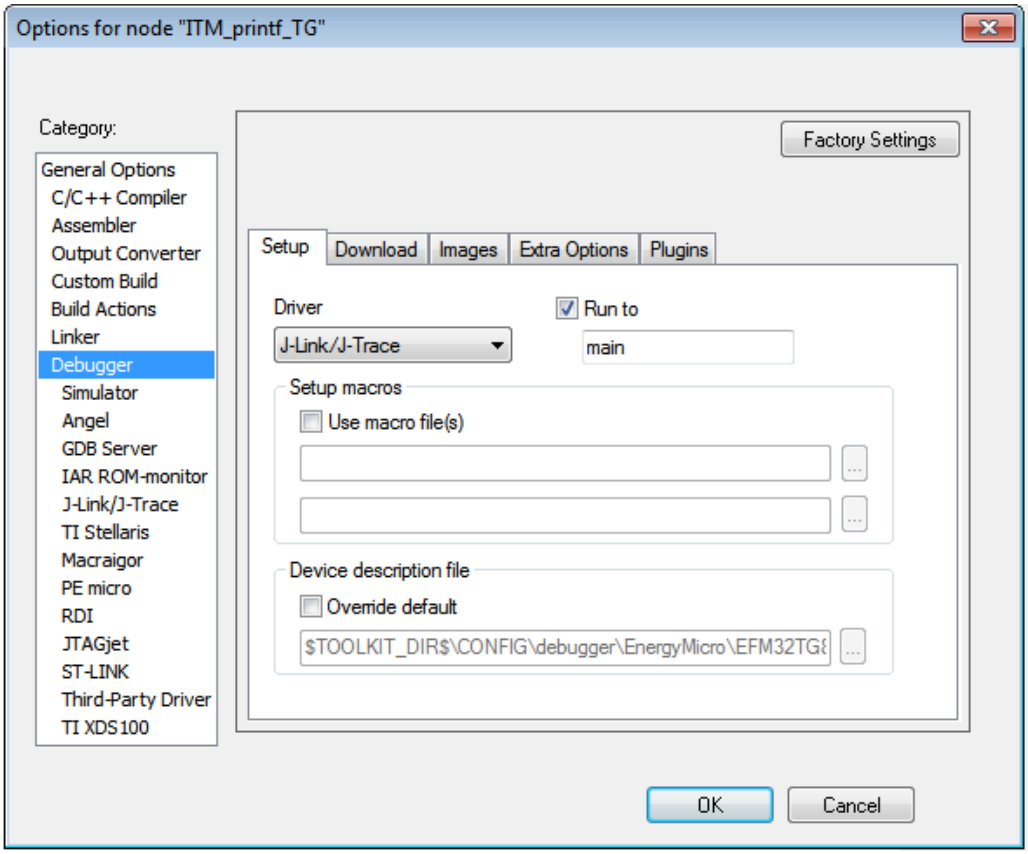


Figure 3.9. IAR debugger download settings.

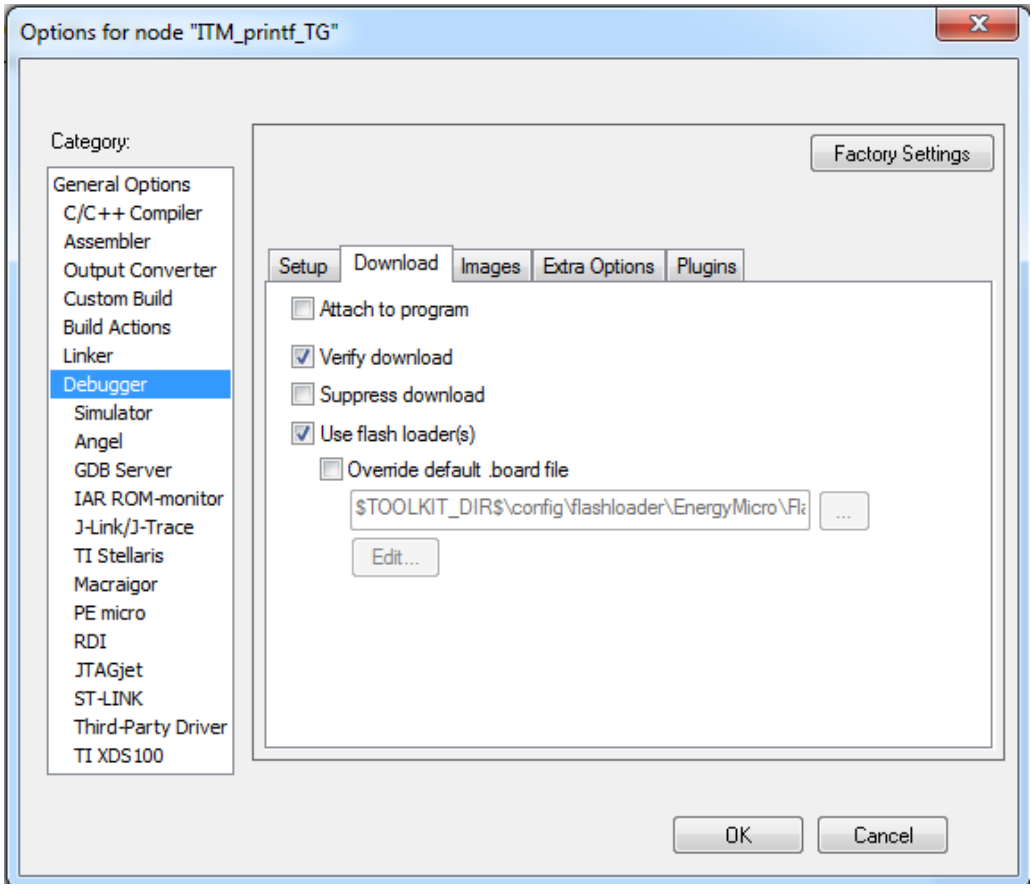


Figure 3.10. IAR J-Link setup settings.

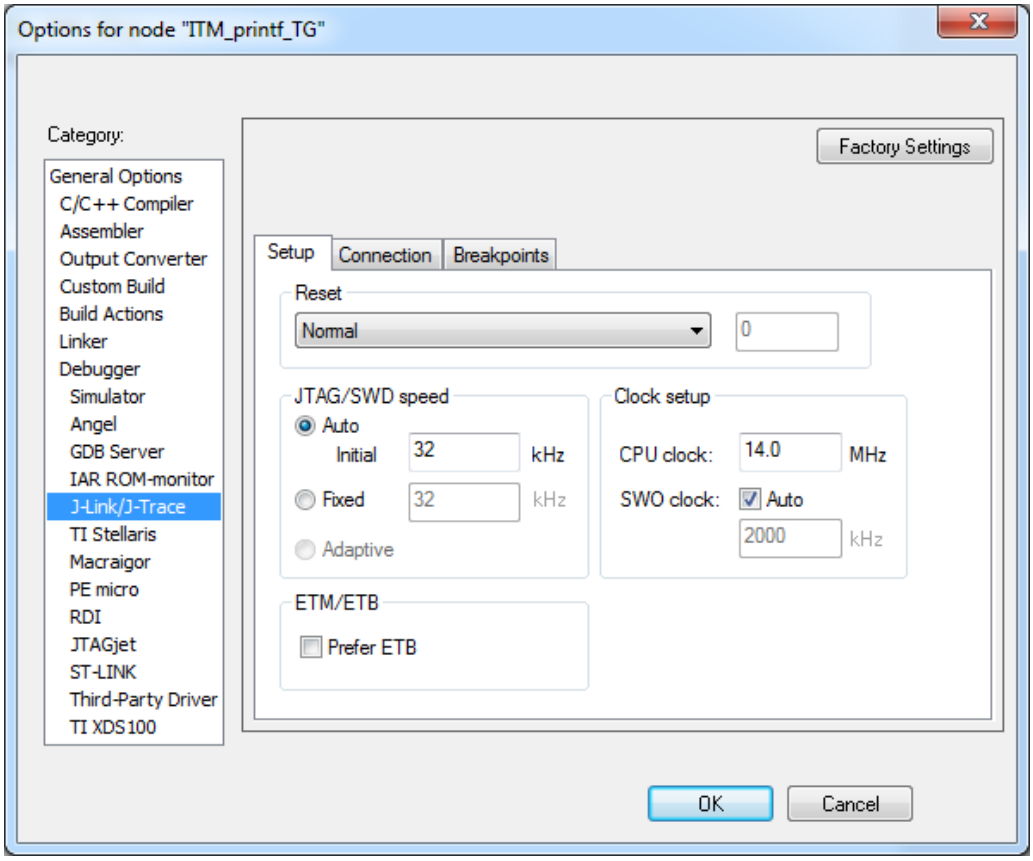
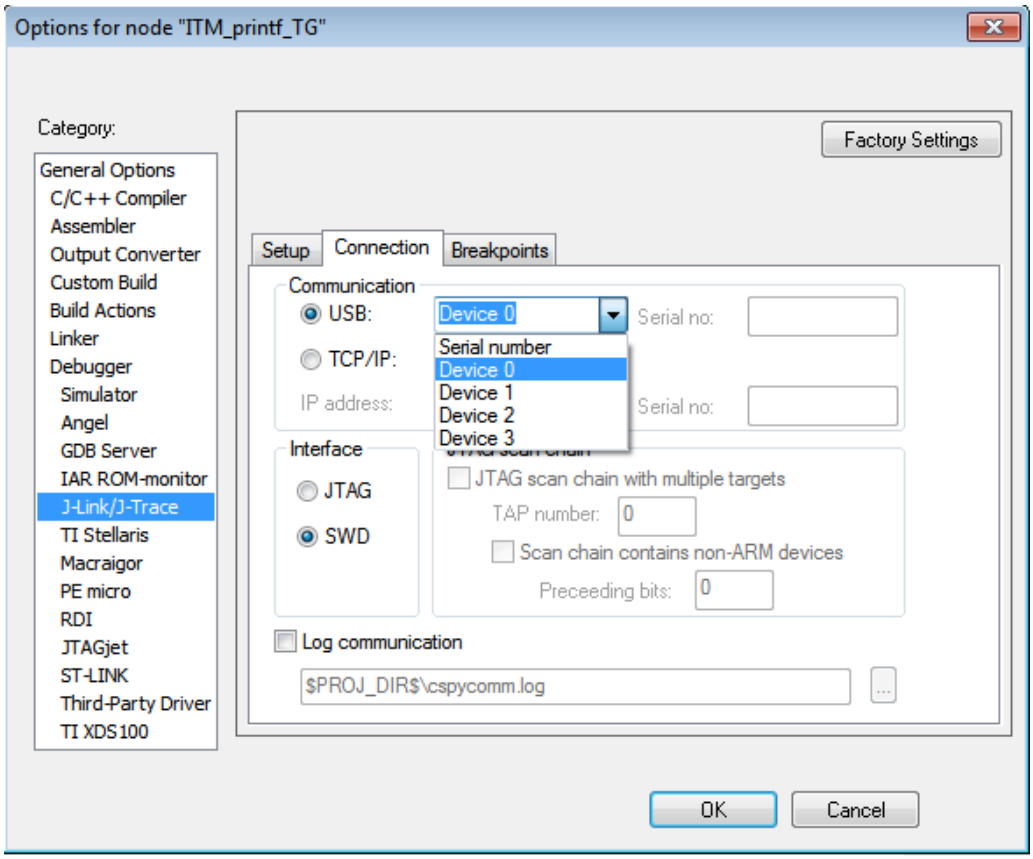
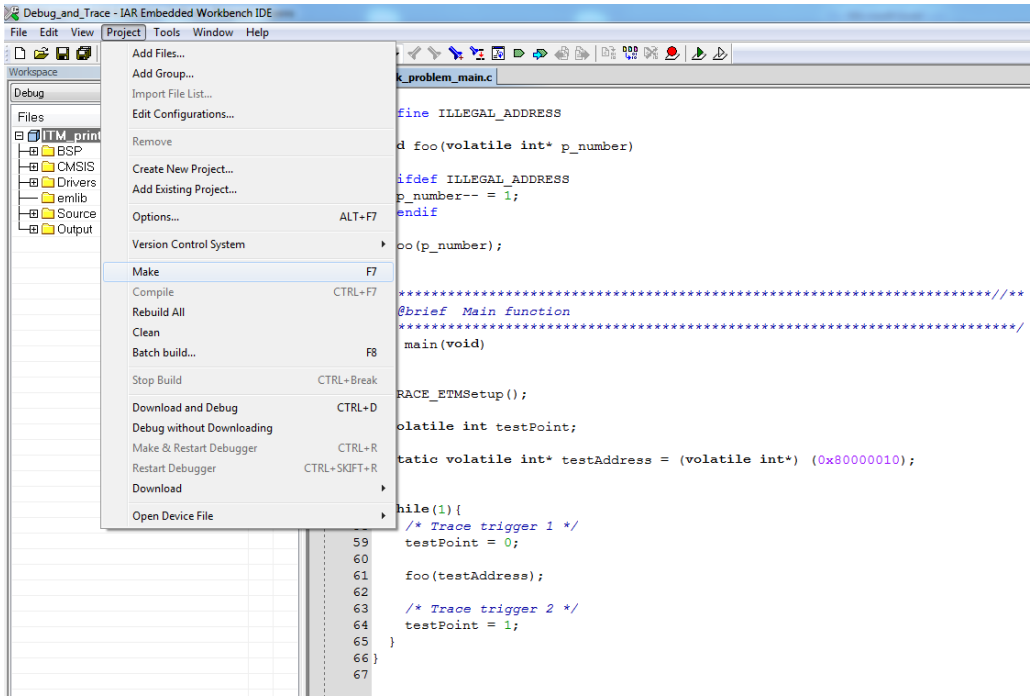


Figure 3.11. IAR J-Link connection settings.



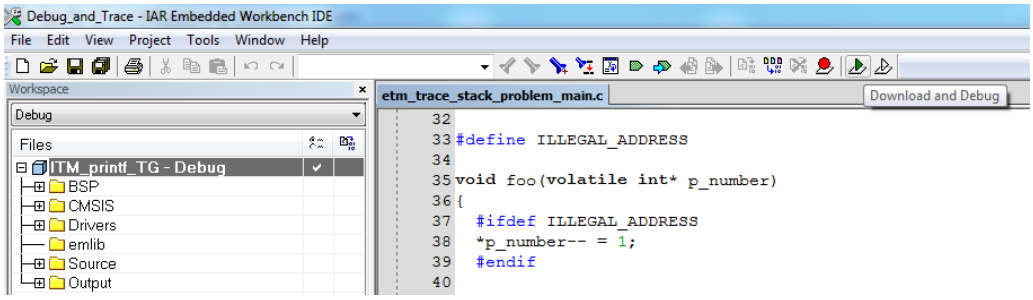
3. Build target by selecting Make from the project-menu or pressing F7.

Figure 3.12. IAR build project.



4. Download code and start the debug session by the small green arrow or Ctrl+D. See screenshot Pressing this button will also do a build if there are changes since the last build.

Figure 3.13. IAR upload code and start debugging.



5. You are now in a debug session, single stepping and other debug feature are available in the top left corner.

For more information regarding debugging in IAR, please refer to the IAR documentation:

- **ftp://ftp.iar.se/WWWfiles/arm/webic/doc/EWARM_DebuggingGuide.ENU.pdf**

3.4 Serial Wire Output

In addition to the two SW-DP pins which allows normal debug operations and programming of the EFM32. The microcontroller also includes a SWO (Serial Wire Output) interface. SWO is a one wire output from the ITM (Instrumentation Trace Module), see Figure 2.1 (p. 3) . By having the software writing to the ITM-registers, data can be sent to the debugger through this one wire interface. This enables debug printf messages to be sent without setting up a separate serial output connection.

The ITM also supports time stamping of events like interrupt handling and program counter samples. This enables simple code tracing and profiling. The bandwidth is not high enough to capture all the activity

going on inside the CPU. Still time stamped samples of the program counter and interrupt execution can be valuable information in a debug situation.

3.4.1 SWO printf in IAR

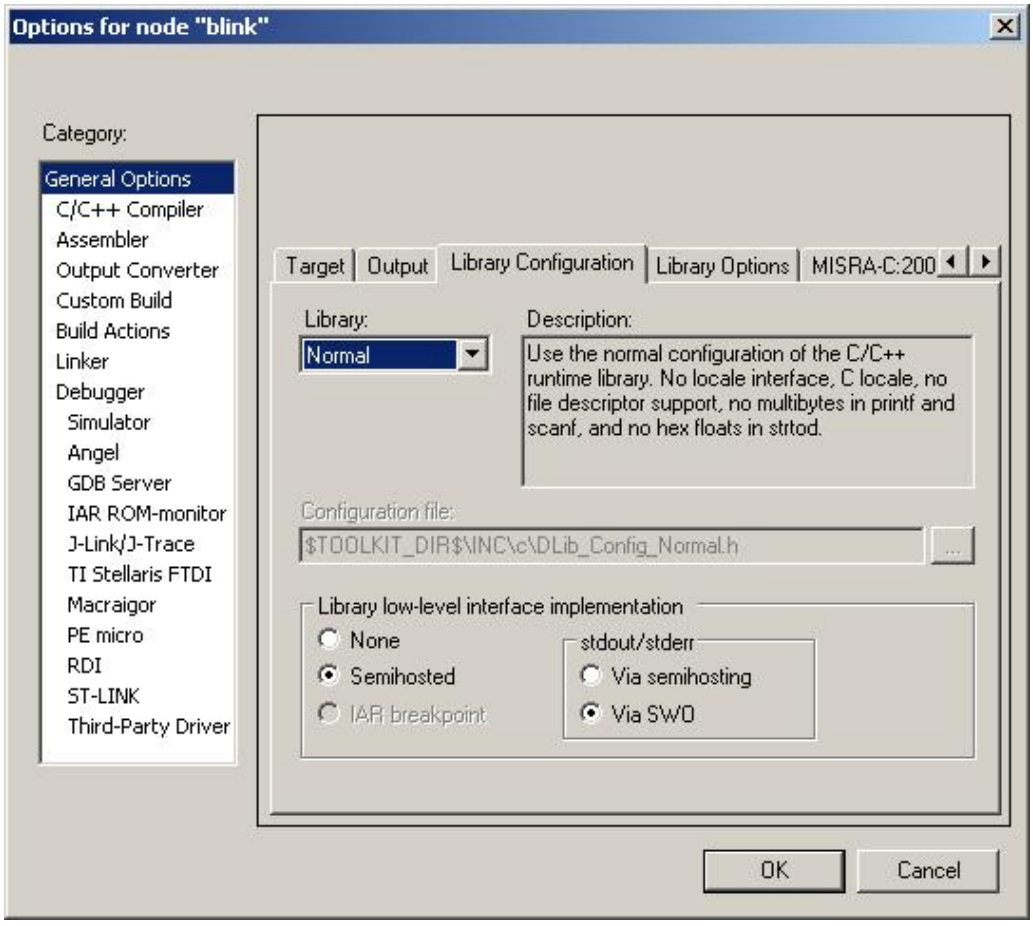
The following walk through demonstrates how to send `Hello world` through the SWO interface to the IAR IDE terminal output.

1. Add `#include <stdio.h>` to the beginning of the file where the printf statement is located.
2. Enable the EFM32 SWO Output.

   The simplest way of enabling the SWO line in the EFM32 is by using the setupSWO function. You can find this function in the energyAware Profiler, in the left pane, when you open up the program. Once this is executed, the SWO output will be enabled on the correct pin for the Development Kit or Starter Kit.

   The content of the setupSWO function is also available as part of the board support package for each kit. The software examples supplied with this application note includes the `TRACE_SWOSetup()` function from there which does the same thing as the setupSWO in the profiler.
3. Configure the project in IAR to receive and transmit data through SWO:

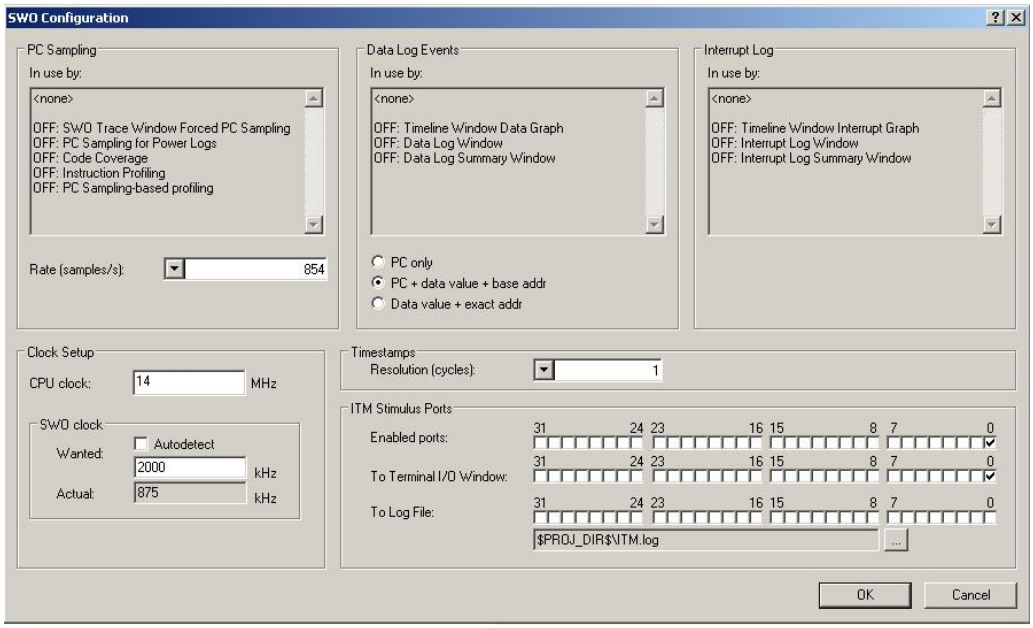
   Figure 3.14. IAR project options for SWO printf.



4. Write `printf("Hello world");` in your code after you have enabled the SWO output.
5. Compile the code and download it to the Starter Kit/Development Kit. Start the debug session.
6. Once you are in debug mode, you should set the correct clock speed (14 MHz) and enable one or more ITM ports in J-link->SWO Configuration, Figure 3.15 (p. 14). Please note that the clock

speed is 14 MHz regardless of your core clock frequency. The debug clock is derived from the AUXHFRCO which runs at 14 MHz by default.

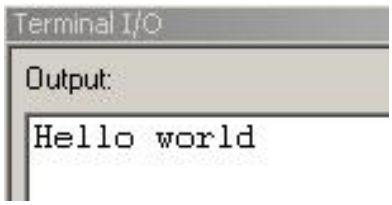
Figure 3.15. SWO configuration in IAR.



7. Open up View->Terminal I/O.

8. When you hit Go you should see the printf statement show up as below:

Figure 3.16. IAR terminal output.



3.4.2 SWO printf in Keil

The following walk through demonstrates how to send `Hello world` through the SWO interface to the Keil IDE terminal output.

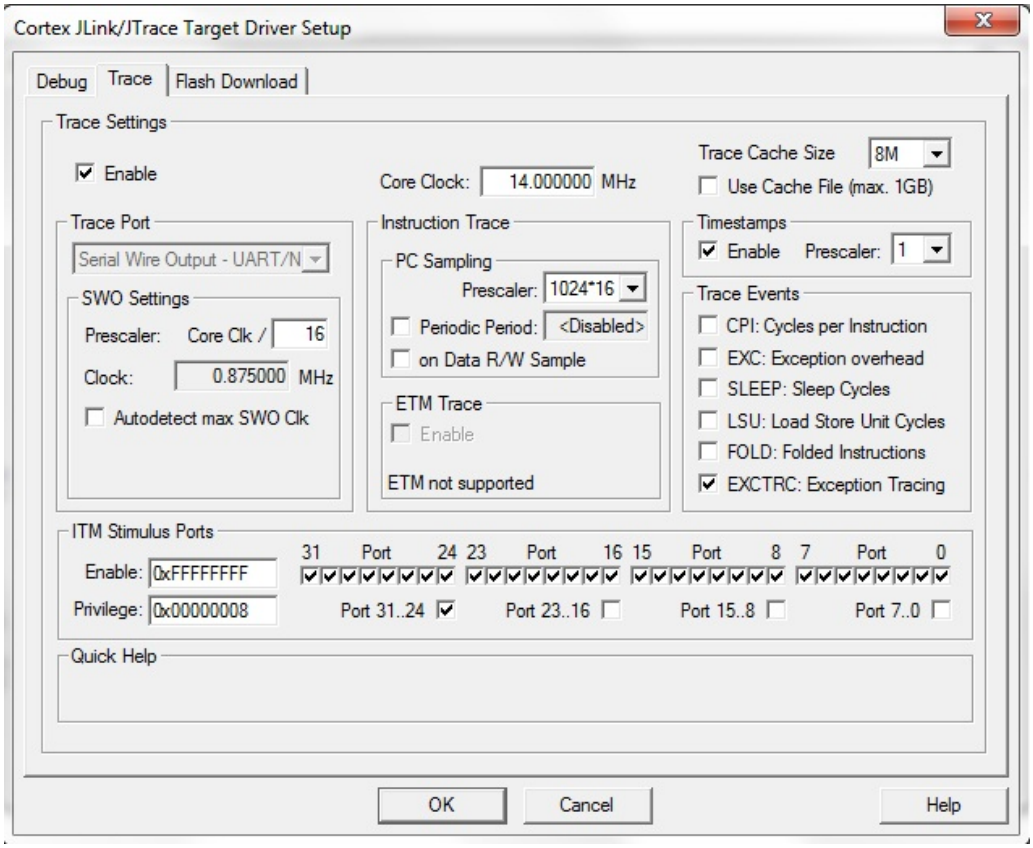
1. Add `#include <stdio.h>` to the beginning of the file where you want to write your printf statement.

2. Enable SWO Output.

   The simplest way of enabling the SWO line in the EFM32 is by using the setupSWO function. You can find this function in the energyAware Profiler, in the left pane, when you open up the program. Once this is run, the SWO output will be enabled to the correct pin on the Development Kit or the Starter Kit.

3. In the project options in Keil, go to the Debug tab and Press the Settings button next to the debugger selection (Should say Cortex-M/R J-LINK/J-Trace). Go to the Trace tab and check the Enable box as well as setting Core Clock to 14 MHz and the Prescaler under SWO Settings to Core Clock / 16. Please note that the clock speed is 14 MHz regardless of your core clock frequency. The debug clock is derived from the AUXHFRCO which runs at 14 MHz by default.

Figure 3.17. Keil debugger options for SWO printf.



4. Write `printf("Hello world");` in your code after you have enabled the SWO output.

5. To enable the ARM compiler to send printf commands through the SWO interface you need to add the following lines to your code:

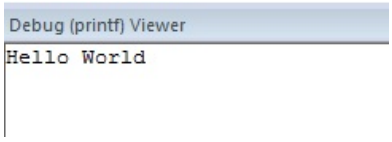
```
struct __FILE { int handle; /* Add whatever you need here */ };
FILE __stdout;
FILE __stdin;

int fputc(int ch, FILE *f) {
 ITM_SendChar(ch);
 return(ch);
}
```

6. Compile the code and download it to the Starter Kit/Development Kit. Enter a debug session.

7. Open the printf-viewer by going to View->Serial Windows->Debug (printf) Viewer

8. When you hit Run you should see the printf statement show up as below:

Figure 3.18. Keil terminal output.



3.5 Alternative Capture of SWO Output

The software trace data transmitted by the printf function, or by other means through the SWO-output, must not necessarily be read in a terminal window within the IDE itself. Some debuggers support reading the SWO output directly in a separate terminal window. The Energy Aware Commander software from Energy Micro also supports reading printf SWO output directly with the command line feature '`--readswo`'.

3.6 Other SWO-features

The Instrumentation Trace Module with SWO output can be used for several other features than printf debugging. Some features include the Data Watchpoint and Trace module which can monitor the value of variables. Time Stamping feature which is useful for measuring execution time of a piece of code and Interrupt Trace which gives information about interrupt execution and timing. Documentation and software examples for the other SWO features can be found in the IAR CoreSight application note referred to in Section 5.4 (p. 22) .

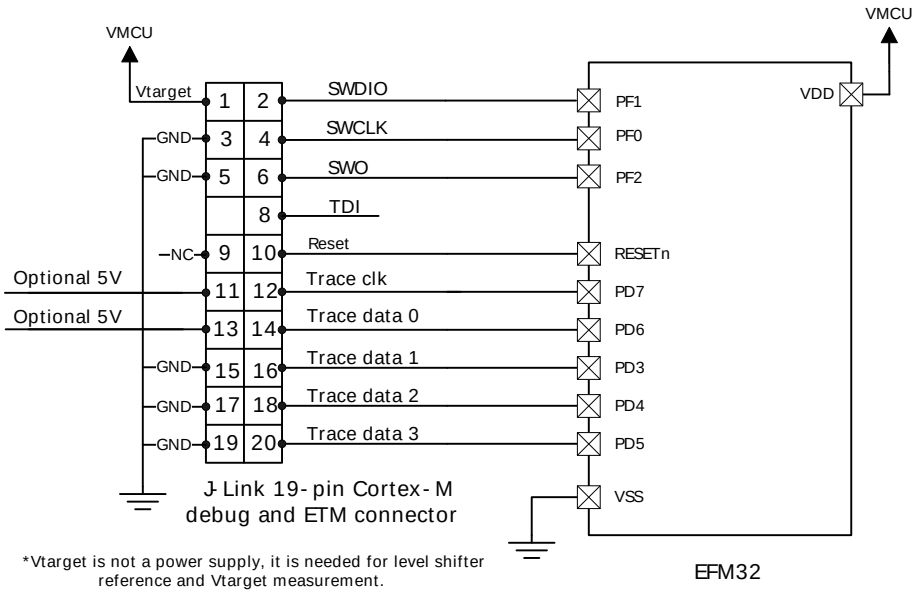
4 Full Instruction Trace

Some device families, including Leopard Gecko, Giant Gecko and Wonder Gecko devices include the CoreSight Embedded Trace Macrocell (ETM). Through a separate 5 wire interface the ETM is a debug component that enables full reconstruction of actual program execution. The ETM is designed to be a high-speed, but still low-power debug tool that supports instruction trace.

4.1 Hardware Connection

The EFM32's hardware connection for instruction trace normally uses 5 additional pins compared to the 2 pins of the serial wire debug interface. Figure 4.1 (p. 17) illustrates the pinout of the connector chosen on the development kits. It is a 1.27mm pitch 20 pin connector which is smaller than the standard 2.54 debug connector. It includes both the normal debug interface and the instruction trace connection. No additional connector is needed for programming/debugging if the trace-connector is used.

Figure 4.1. Typical 5 wire ETM trace connection.



4.2 ETM Instruction Trace Setup

How to start a debug session with ETM instruction trace enabled. Instructions given for both Keil and IAR.

4.2.1 Instruction Trace in IAR

1. If you are using the DK3650/3750 as J-Trace, make sure that ETM trace is enabled on the kit itself. This can only be configured through the kit CFG-menu on the TFT-display.
2. Make sure your project includes the `TRACE_ETMSetup()` at startup. This enables the ETM trace inside the EFM32 and also enables the correct GPIO-pins for trace to function on the development kit.

   Note: If you are using your own board with trace routed to other pins, please change this ETMSetup function accordingly.
3. Start a debug session and then open the J-Trace window (J-Link -> ETM Trace), function trace can also be opened from here. If trace is turned on the small ETM icon in the upper left corner should be green. Clicking this will also open the trace settings.

Figure 4.2. Open ETM Trace window in IAR.

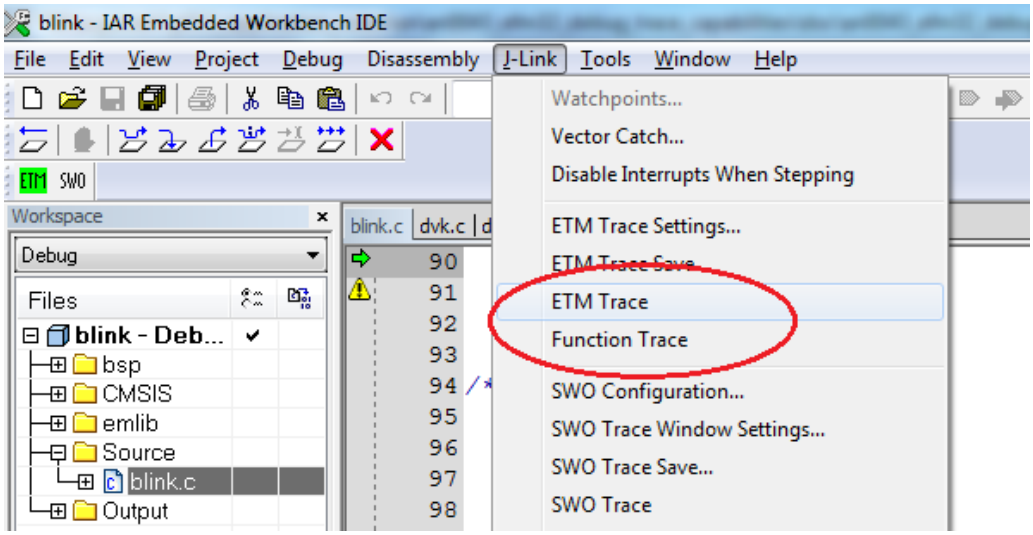
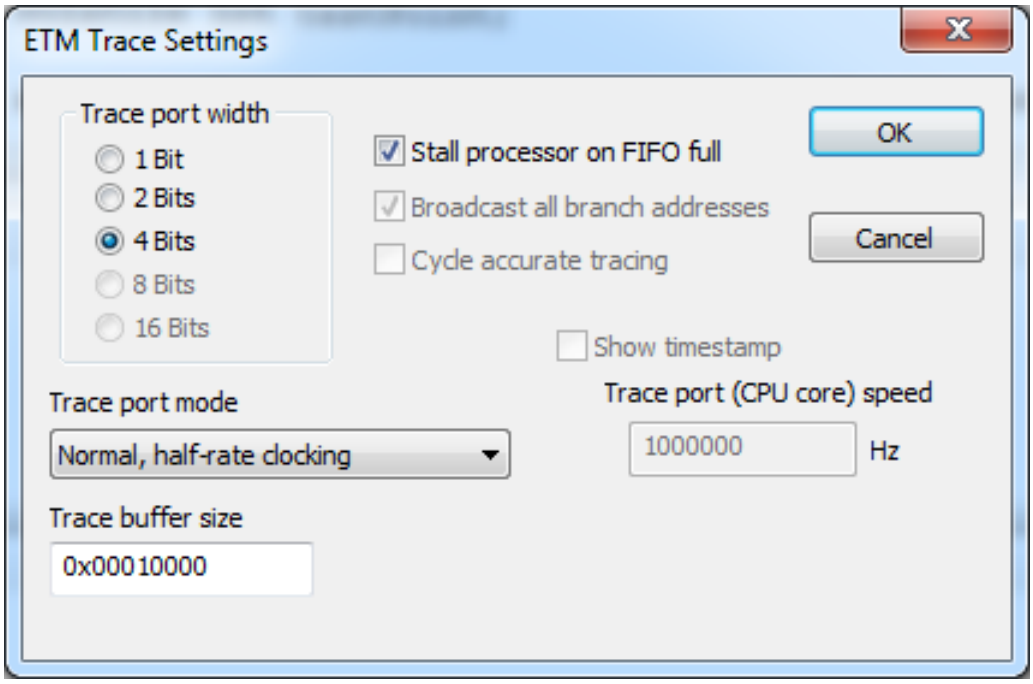
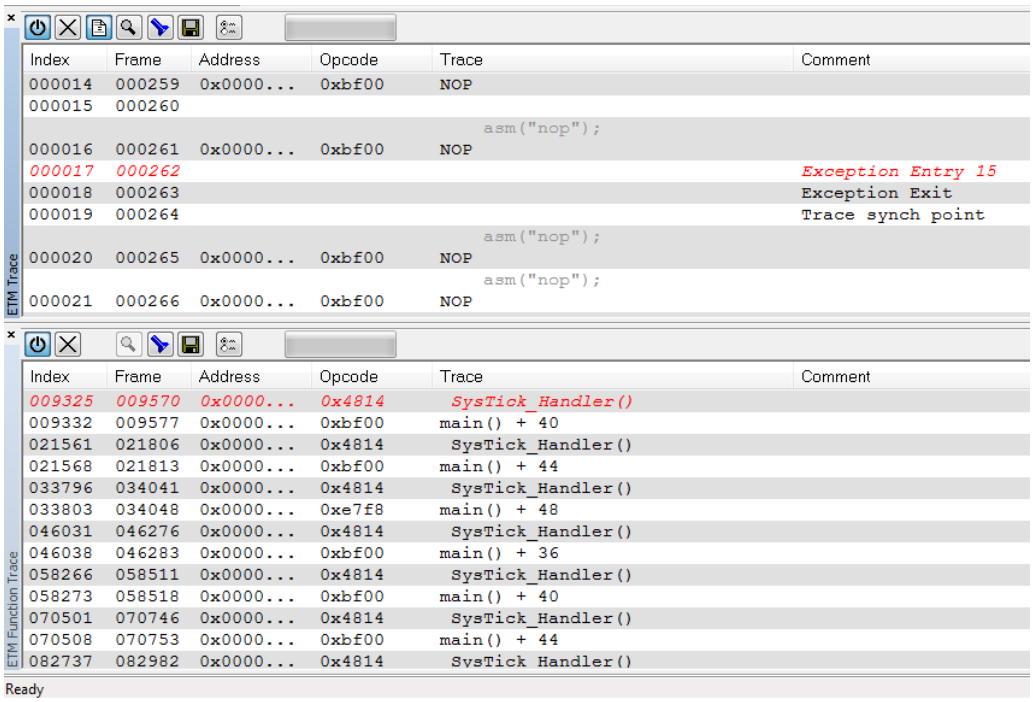


Figure 4.3. ETM Trace settings in IAR.



4. You are now in a debug session with instruction trace enabled. You can single step and should see instructions appear in the trace window.

Figure 4.4. IAR instruction and function trace windows.



5. You can right click on a line of code and enable trace start/stop breakpoints, this is an important feature which helps to limit the amount of trace data when debugging a problem in a specific part of the code.

Figure 4.5. IAR toggle breakpoints for trace start and trace stop.

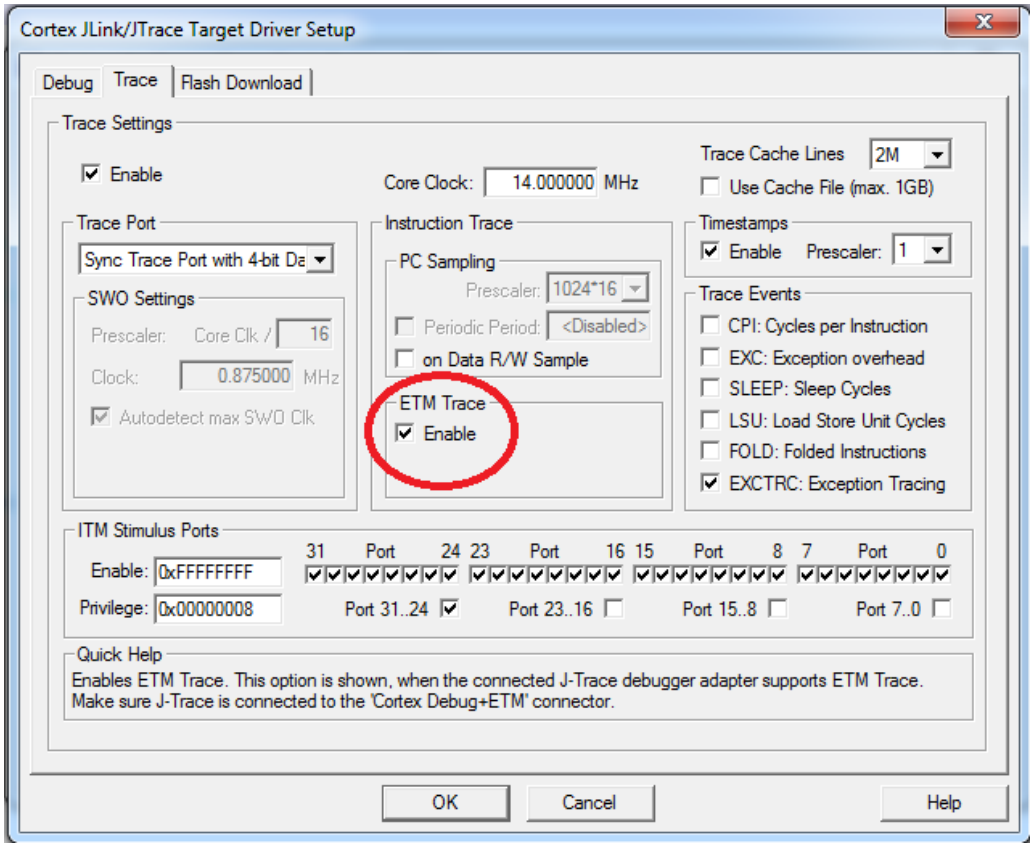
4.2.2 Instruction Trace in Keil

1. If you are using the DK3650/3750 as J-Trace, make sure that ETM trace is enabled on the kit itself. This can only be configured through the kits CFG-menu on the kits own TFT-display.

2. Make sure your project includes the `TRACE_ETMSetup()` at startup. This enables the ETM trace inside the EFM32 and also enables the correct GPIO-pins for trace to function on the development kit. Note: If you are using your own board with trace routed to other pins, please change this ETMSetup function accordingly.

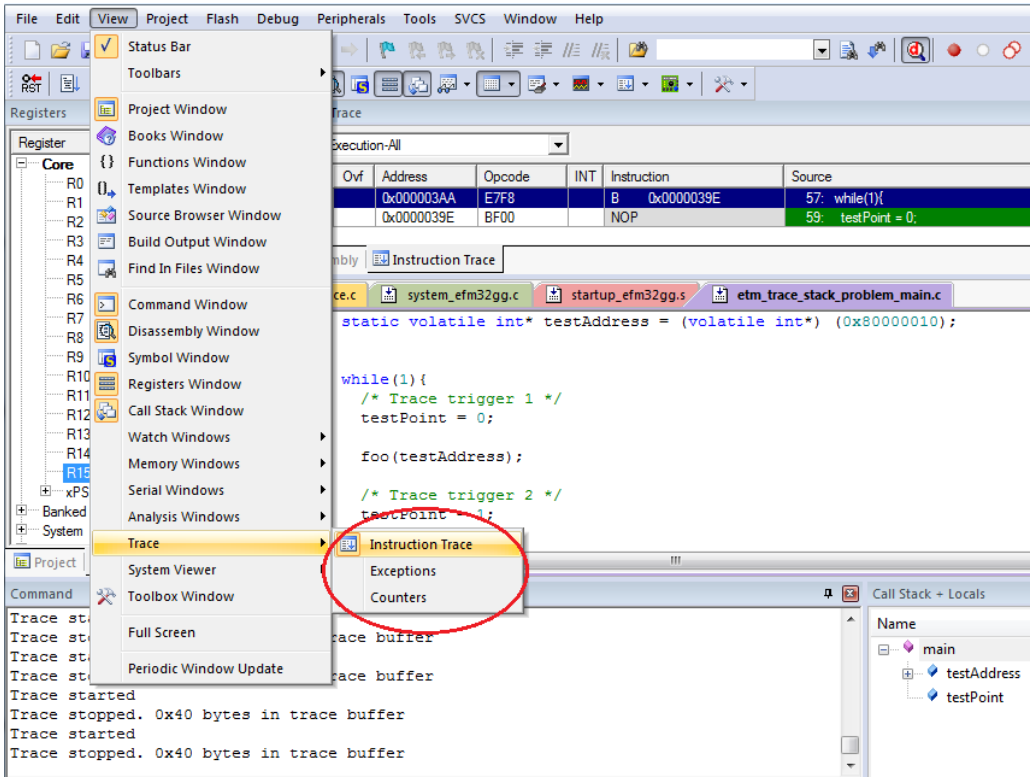
3. Check that Trace is configured correctly in the Cortex Jlink/JTrace Target Driver Setup, note that you need to enable trace separately in the trace setup window.

Figure 4.6. ETM trace settings in Keil.



4. Start a debug session and then open the instruction trace window to see the trace data. (View -> Trace -> Instruction Trace).

Figure 4.7. Keil instruction trace window.



5. You are now in a debug session with instruction trace enabled. You can single step and should see instructions appear in the trace window.

5 Software Examples

This application note includes software examples for both the Tiny Gecko and Giant Gecko EFM32 microcontrollers. The software examples demonstrate how to configure the EFM32 to perform both SWO printf, ITM trace and full ETM trace. (Please note that a debugger with J-Trace is only available on some kits, for example the DK3750/DK3650.)

5.1 ITM printf Output

This example uses the `retargetio.c` driver to make the CMSIS function `ITM_SendChar(c)` the default used by `printf()` to print characters.

The example also demonstrates how one can use the other ITM-channels. The `ITM_SendChar(char c)` only uses channel 0. By defining a separate `ITM_Port32(n)` one can write 32 bit words to any of the 32 instrumentation trace channels. Notice that the software needs to wait until the channel buffer is ready before it can be written to. The specific ITM ports must also be enabled in the SWO-configuration in the IDE. If just SWO output is needed without an IDE, the ITM ports must be enabled by the software running on the EFM32. Please see the `TRACE_SWOSetup()` function.

5.2 Trace and ITM printf Output

This example is exactly the same as the previous one, except that it is configured for the giant gecko and includes the `TRACE_ETMSetup()` function.

5.3 Trace Problem

This example demonstrates how one can use the instruction trace feature (included with the DK3750/DK3650 development kits) to debug software problems. The problems created on purpose in this example generates a hard fault in two different ways. In this example it is relatively easy to spot where the bugs are without instruction trace debugging. But since such problems usually occur deep within larger software project, it can be invaluable information to know which instructions were executed right before the hard fault occurs.

One of the bugs introduced is a stack overflow caused by recursion. The other bug is an illegal memory access, caused by reading from a memory address which lies outside the defined memory block for the external bus interface (EBI). The illegal address hard fault can be selected by defining the `ILLEGAL_ADDRESS` define.

In both cases it is convenient to introduce a breakpoint in the hard fault handler. If not, the amount of trace data after the hard fault occurs will be huge and it can cause trace buffer overflow.

Because of the amount of RAM on the Giant Gecko device, it will take a few seconds before the stack overflow causes a hard fault.

5.4 More Examples

More information and several software examples can be found in the IAR CoreSight Debug and Trace tutorial:
http://www.iar.com/Global/Resources/Developers_Toolbox/Building_and_debugging/Using_CoreSight_Trace_Techniques_on_Cortex-M3-M4/Using_CoreSight_Trace_Techniques_on_Cortex-M3-M4.pdf

6 Revision History

6.1 Revision 1.03

2013-09-03

New cover layout

6.2 Revision 1.02

2013-05-08

Added software projects for ARM-GCC and Atollic TrueStudio.

6.3 Revision 1.01

2012-11-12

Adapted software projects to new kit-driver and bsp structure.

6.4 Revision 1.00

2012-10-01

Initial revision.

A Disclaimer and Trademarks

A.1 Disclaimer

Silicon Laboratories intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Laboratories products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Laboratories reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Laboratories shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products must not be used within any Life Support System without the specific written consent of Silicon Laboratories. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Laboratories products are generally not intended for military applications. Silicon Laboratories products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

A.2 Trademark Information

Silicon Laboratories Inc., Silicon Laboratories, the Silicon Labs logo, Energy Micro, EFM, EFM32, EFR, logo and combinations thereof, and others are the registered trademarks or trademarks of Silicon Laboratories Inc. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.

B Contact Information

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701

Please visit the Silicon Labs Technical Support web page:
http://www.silabs.com/support/pages/contacttechnicalsupport.aspx
and register to submit a technical support request.

Table of Contents

List of Figures

silabs.com


ZERO
ARM Cortex-M0+
TINY
ARM Cortex-M3
GECKO
ARM Cortex-M3
LEOPARD
ARM Cortex-M3
GIANT
ARM Cortex-M3
WONDER
ARM Cortex-M4